*Final Report for:*

# Configuring Embeddable Adaptive Computing Systems for Multiple Application Domains with Minimal Size, Weight, and Power

DARPA Contract F30602-97-2-0297

Technical Point of Contact:

Dr. John K. Antonio, Director
School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019-6151
fax: 405-325-4044
tel: 405-325-4397
antonio@ou.edu
www.cs.ou.edu/~antonio

October 2002

## Abstract

A main objective of this effort was to demonstrate the advantages of integrating field programmable gate array (FPGA), digital signal processor (DSP), and general purpose processor (GPP) computing technologies together to perform computations required by embedded radar applications of interest to DARPA. Particular emphasis was placed on minimization of the overall size, weight, and power (SWAP) of the computational platform. To facilitate the practical use of integrated systems containing different types of computing technologies, configuration techniques and tools were developed for aiding the system designer in determining optimal specifications of system requirements for given instances of the applications considered. A prototype FPGA/DSP/GPP-based system was developed to illustrate the feasibility of the research.

## Table of Contents

**List of Figures**

**Acknowledgments**

I would first like to acknowledge and thank the graduate students that worked on this project. Every publication associated with the project was co-authored with one or more graduate assistants and was based on research conducted in conjunction with a PhD dissertation or MS thesis.

Jeff Muehring (initially an MS student and later a PhD student) conducted research in the area of determining optimal configurations for SAR (synthetic aperture radar) processing. His work included the application of mathematical programming techniques for determining optimal multiprocessor configurations for SAR. His techniques centered on the concept of making the proper trade-off between processing hardware and memory so as to minimize the overall power consumption of the system, while satisfying throughput requirements. Jeff also made contributions in the domain of FPGA (field programmable gate array) design by proposing a new way of implementing two-dimensional signal processing tasks (including SAR) using very deep FPGA pipelines. In addition, Jeff played an important role in designing and implementing the hardware portions of the custom interfaces between the FPGA and DSP/GPP subsystems of the constructed prototype platform.

Jack West (initially an MS student and later a PhD student) conducted research in the area of minimizing communication time for STAP (space-time adaptive processing) executing on a multiprocessor. By minimizing communication overhead, he demonstrated that less hardware is required for given instances of STAP, thereby reducing SWAP (size, weight, and power). The first phase of Jack's work included the development of a simulator for Mercury's RACEway® interconnection network. This fast and efficient simulator was used in the second phase of his work in which genetic algorithm approaches were developed for solving the communication scheduling problem. Jack also played important roles in developing the prototype system by helping with the STAP application software implementation and by designing and implementing the software portions of the custom interfaces connecting the FPGA and DSP/GPP subsystems.

Tim Osmulski (MS student) developed and implemented an analytical tool, in software, for estimating power consumption of a configured FPGA chip. This tool, which was the first of its kind, demonstrated that it is indeed possible to accurately predict FPGA power consumption by applying existing analytical approaches. The accuracy of Tim's tool was verified by comparing its predicted values with actual measured power consumption taken from an instrumented FPGA board.

Hongping Li (PhD student) developed a new analytical approach for estimating power consumption of circuits, including those implemented on a FPGA. His approach is based on a Markov chain signal model, and directly accounts for correlations present among the internal signals of the circuit. Hongping verified the accuracy of his approach using PSpice® based simulation studies. Hongping also lead the effort in implementing the parallel SAR application software on the multiprocessor system.

Nikhil Gupta (MS student) developed FPGA circuit designs to support core calculations required by STAP. His work demonstrated that 16-bit block floating point

arithmetic provides acceptable accuracy for many situations. The advantage of using block floating point arithmetic, instead of standard floating point, is the significant reduction in the size and power consumption of the corresponding circuits. His research illustrated that if the values of the input data are approximately uniformly distributed, then the block floating point approach delivers acceptable accuracy.

Brian Veale (initially an MS student and later a PhD student) conducted a study comparing different FPGA designs and implementations for an inner product co-processor. He studied two architectural approaches for the co-processor and two different types of arithmetic (integer and floating point) for a total of four combinations. For each implementation, he also studied the effect that employing different degrees of pipelining had on each design in terms of size, speed, and power consumption. His study of pipelining resulted in some counterintuitive results. In particular, while it is well known that increasing the degree of pipelining generally enables custom designs to be run at faster clock rates, the same is not always true for FPGA designs.

Sirirut Vanichayobon (PhD student) studied the power-speed trade-off for a class of circuits known as prefix circuits. These circuits are important in their own right, and are representative of the type of circuit often required in high-performance embedded applications. Through extensive analysis of a number of known prefix circuits, her work illustrates that the trade-off between power and speed is not always obvious to the circuit designer. Based on discoveries made through her research, some important guidelines for properly matching circuit characteristics with power and speed requirements are provided. Sirirut also lead the effort in implementing the parallel STAP application software on the multiprocessor system.

I would also like to acknowledge the work and contributions of faculty colleagues. Dr. Sudarshan Dhall served as co-PI on this project since the Fall of 1999; the time at which I became Director of Computer Science at the University of Oklahoma. Dr. Dhall made contributions in nearly all aspects of the project, and was particularly instrumental in guiding the research of graduate assistants Sirirut Vanichayobon and Hongping Li. Dr. Dhall's expertise in system modeling – and probabilistic techniques in particular – was extremely valuable. The project also benefited greatly by the contributions of Dr. S. Lakshmivarahan. It was Dr. Lakshmivarahan that originally proposed the topic of Sirirut's research, and he and Dr. Dhall served as co-advisors of her PhD committee.

Next, I would like to acknowledge the assistance and guidance of key defense personnel, starting with Rick Metzger of Rome Laboratory. Rick served as program manager for a related prior project I performed for Rome Laboratory, and it was this past work experience that enabled me to be successful in proposing and completing the present project for DARPA.

I would like to acknowledge the support and encouragement of José Muñoz, who served as the original program manager for DARPA's ACS (Adaptive Computing Systems) program. José provided valuable feedback and perspective throughout the contract period. I had the opportunity to meet with José and his staff frequently, including at annual reviews, PI meetings, and other professional conferences. He actively encouraged and facilitated interaction and collaboration among the PIs of different projects, which ensured that the ACS program was cohesive and integrated. The interactions with other PIs was stimulating, and served to accelerate and improve the quality and relevance of the results delivered by all project PIs.

Assisting José Muñoz with the management of this project was Ralph Kohler of Rome Laboratory. I met with Ralph on a regular basis at meetings and conferences and communicated with him frequently through e-mail and telephone correspondence. Ralph made a number of technical contributions and refinements to the project, and often served as a sounding board on behalf of the military. He related to me the actual needs of the war fighter, and these insights helped us to provide results that were more applicable than would have otherwise been possible. Ralph also helped me tremendously with the overall management and organization of the project.

Finally, I would like to thank Jules Bergmann of Rome Laboratory, who had the unenviable task of encouraging me to complete and submit this final report. Jules was most gracious and professional; he gently, but persistently, encouraged me to finish this report. I would not want to think when this report would have been delivered without the interaction and encouragement provided by Jules.

## Introduction

### *Organizational Structure of the Report*

A challenge in organizing this report was to provide sufficient detail to readers that desire it, while also providing a relatively high-level summary of the entire project. Published materials that resulted from this project currently include eleven conference/journal papers, two PhD dissertations, and five MS theses. The eleven published papers are included in printed form in the appendices of this report. It was natural to include copies of the papers in printed form and refer readers interested in further details to the dissertations and theses (which are available online) because the papers were generally derived from the dissertations and theses. It was infeasible to incorporate the dissertations and theses in printed form; there are over 800 pages associated with these documents. The report is organized hierarchically, as illustrated in Figure 1.

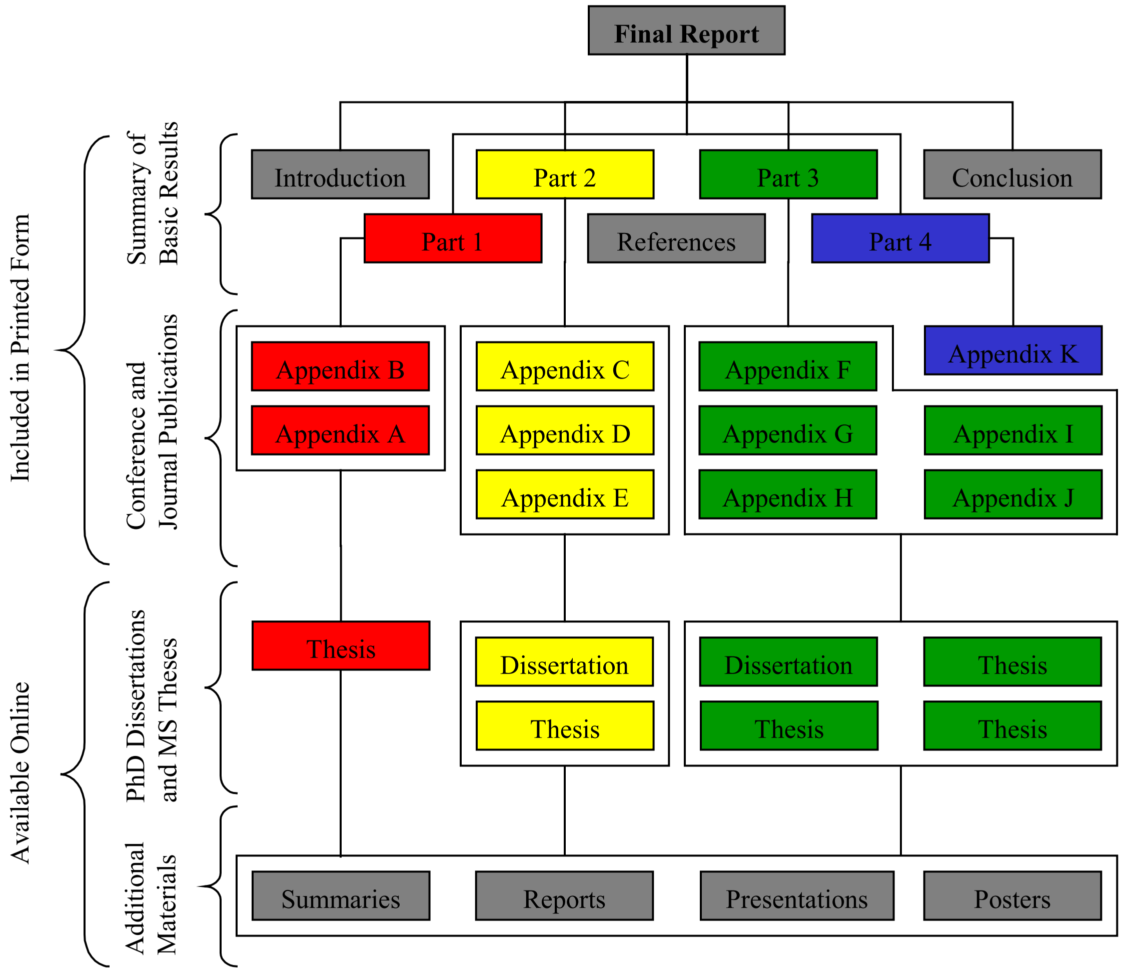


Figure 1. Organizational structure of the report.

The main body of the report provides a summary of basic results, and includes four major parts: (1) Optimal Multiprocessor Configuration for SAR; (2) Optimal Communication Scheduling for STAP; (3) FPGA Power Prediction and Applications; and (4) Hybrid FPGA/DSP/GPP Platform. Each of these parts is supported by a collection of published papers, theses, and dissertations produced during the project period. Copies of the published papers are included in the appendices of the report. References to these publications are labeled with a number followed by the letter of the appendix where a copy of the publication can be found. For example, reference label [1A] indicates that a copy of the referenced publication can be found in Appendix A. Due to size considerations, copies of theses and dissertations, such as reference [3], are not included in an appendix; however, online links for all references are provided in the list of references. For conference papers, links to the associated presentation materials are also provided within the list of references. As illustrated in Figure 1, additional materials are also available online, including annual project summaries, technical reports, and presentations and posters given at conferences and PI (principal investigator) meetings. Online links to additional materials are provided in the section entitled Additional Materials, which follows the References section.

Each major part is divided into subsections, and each subsection provides an overview of one or more published papers. Overviews of some of the conference papers (e.g., [15I] and [18K]) actually expand upon the publication by including content from the presentation materials associated with that publication. Readers not needing the level of detail found in these overviews are encouraged to first read the Acknowledgments section, which includes a paragraph on the work conducted by each student assistant. Of course readers requiring more detail are encouraged to pursue copies of the papers found in the appendices, online links of presentation materials found in the References section, and/or the online links found in the Additional Materials section.

## *Project Overview*

The advantages of using digital signal processing (DSP) chips for high-performance embedded signal processing applications have been demonstrated during the past decade. DSP chips often win over general purpose processors (GPPs) because their complexity (measured, for example, in terms of silicon area, number of transistors, or power consumption) is better matched to the highly regular and numerical-intensive computations required by many signal processing based embedded applications. However, it is now apparent that even DSP chips can be overkill for some computations found in common embedded military applications. That is, in some cases DSP chips are equipped with much more architectural complexity than is actually needed, resulting in inefficiencies and greater power consumption than absolutely necessary.

In this project, we investigated the advantages of integrating configurable hardware together with a multiprocessor DSP/GPP platform. The computational engine of the configurable hardware used in this project was comprised of FPGA chips. A primary goal of our project was to demonstrate that for given computational loads – associated with instances of embedded radar signal processing applications – the total size, weight, and power (SWAP) could be reduced by integrating FPGA-based components as part of the embedded computational platform.

Reconfigurable computing devices, such as FPGAs, can offer a cost-effective and more flexible alternative than the use of application specific integrated circuits (ASICs). FPGAs are especially cost-effective compared to ASICs when only a small number of the chip(s) are required. Another major advantage of FPGAs over ASICs is that they can be reconfigured to change their functionality while still resident in the system, which allows hardware designs to be changed similar to software, and dynamically reconfigured to perform different functions at different times.

A number of theoretical and empirical studies were conducted during the project period to understand and demonstrate the advantages and disadvantages of DSP/GPP versus FPGA technologies with respect to SWAP. A prototype heterogeneous FPGA/DSP/GPP-based platform was constructed using commercial off-the-shelf (COTS) components to demonstrate the utility of a hybrid system containing all three types of technologies. A number of systematic approaches and tools based on mathematical programming and modeling were developed to optimally configure FPGA/DSP/GPP-based platforms for applications in the radar signal-processing domain. The two major applications considered were SAR (synthetic aperture radar) and STAP (space-time adaptive processing).

The prototype system was constructed using COTS components from two vendors: Annapolis Micro Systems, Inc. and Mercury Computer Systems, Inc. We had excellent support from both companies, and we designed and implemented a custom interface to allow communication between two disparate product lines of these vendors. Implementation of a custom interface was necessary because at that time (1997-98) there were few interfacing standards among vendors such as the two we were working with and little customer demand (excluding us, of course!) for providing such an interface. The availability of products and support to more easily interface components from different vendors, including the two we worked with, is much better today. In fact, the output of our research, which illustrated the potential benefits of a hybrid FPGA/DSP/GPP platform, served as a catalyst for these industry sectors to invest significant resources and provide support and standards appropriate for interfacing their product lines.

## *Brief Descriptions of Major Parts of the Report*

Part 1: Optimal Multiprocessor Configuration for SAR – describes research for determining optimal multiprocessor configurations for instances of the SAR processing problem. The research was targeted at how to optimally configure a multiprocessor system for given instances of the SAR problem so that the resulting power consumption of the multiprocessor system is minimized. The key to the approach involved making the proper trade-off between the number of processors and amount of memory associated with the multiprocessor configuration. References associated with this work are [1A], [2B], and [3].

Part 2: Optimal Communication Scheduling for STAP – describes research for determining how to best schedule inter-processor communications of a parallel STAP algorithm mapped onto a Mercury Race Multiprocessor. The approach is based on a genetic algorithm, and the research also resulted in the development of a fast and accurate network simulator for the RACEway® interconnection network. References associated with this work are [4C], [5D], [6E], [7], and [8].

Part 3: FPGA Power Prediction and Applications – describes mathematical models and other approaches developed for predicting power consumption for FPGA circuits. We found that predicting power consumption for FPGAs was particularly difficult, as it strongly depends on precisely how the chip is configured and the "activity" characteristics of the input data being processed. Nevertheless, we generated new and important results and tools in this area. We also demonstrated the utility of using FPGA circuits for portions of the SAR and STAP applications. References associated with this work are [9F], [10], [11G], [12H], [13], [14], [15I], [16J], and [17].

Part 4: Hybrid FPGA/DSP/GPP Platform – describes a prototype hybrid platform that was constructed for this project. It includes the detailed design and development of the custom interfaces implemented to interconnect the disparate products of the two vendors. Some performance results are also included. The reference associated with this work is [18K].

## Part 1: Optimal Multiprocessor Configuration for SAR

*Overview of References [1A], [2B], and [3]*

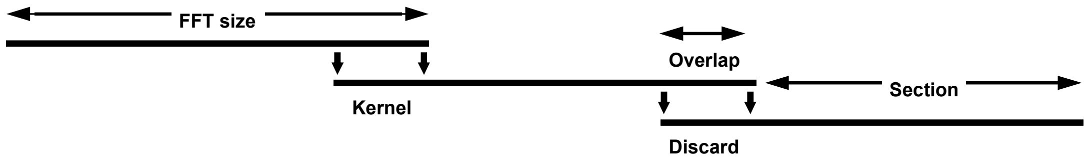
The real-time embedded application considered in this part, i.e., SAR, as well as many others of military interest, are characterized by a common theme: processing a continuous stream of data collected from radar sensors. The rate at which data samples flow from the sensor(s) to the computational platform is typically very high – often on the order of tens or hundreds of millions of samples per second and even higher. Furthermore, the number of calculations to be performed on each sample is typically at least 100 FLOPs (floating-point operations), which amounts to an overall computational throughput requirement ranging from at least one to ten billion FLOPs (and often much higher).

At the beginning of the contract period, approaches capable of providing a computational platform that could achieve these types of computational throughput rates typically involved a "pipeline of interconnected processors" style of architecture. Such an approach could be a valid and effective architecture in some cases. However, situations often arose in which the throughput requirements dictated that 100 or more SHARC® (or similar) DSP processors were required. In many situations, the associated level of power requirement for the computational platform alone posed a severe problem, because of the strict power budgets available on UAVs (unmanned aerial vehicles) and satellites where these systems are deployed.

In the paper [1A], we showed how a DSP/GPP-based multiprocessor system could be optimally configured using two types of processor/memory daughtercards to minimize overall power consumption for SAR applications. We showed that by careful (and often counterintuitive) selection of parameters associated with both the hardware (the number of daughtercards of two possible types) and the application software (a parameter known as the azimuth section size), an optimal configuration (one with minimal power consumption) can be derived based on the application of mathematical programming techniques.

Our approach centered on the derivation of two mathematical formulas for given instances of the SAR problem: one for the total numbers of processors required and the other for the total memory required. Both of these functions are dependent on the choice of the section size parameter. The derived functions dictate that if a small section size is used, then the associated memory requirements are small, but the processor requirements are high. On the other hand, a large section size was shown to result in a requirement for fewer processors, but more memory.

The reason a large section size implies that fewer processors are required is because only a small fraction of data is discarded during the calculation of the so-called sectioned fast convolutions (refer to Figure 2). This implies that the processors are being used with high efficiency when the section size is large. On the other hand, when a small section size is used, then more processors are required because a relatively large fraction of data is overlapped. From Figure 2, note that the overlapped data samples are actually processed twice. Although achieving high processor efficiency is a traditional objective, the trade-off is that implementing the associated large section sizes requires extra memory, and extra memory consumes extra power. It is this inherent trade-off between processor efficiency, memory, and section size that our approach optimized.



**Large Overlap/Section ratio ⇒ Small azimuth memory, large number azimuth processors**
**Small Overlap/Section ratio ⇒ Large azimuth memory, small number azimuth processors**

Figure 2. This diagram illustrates the method of performing sectioned fast convolutions on azimuth input data with a pre-stored kernel. Given that the kernel size is fixed, then if the section size is made large, a relatively small fraction of samples are discarded for each section, thus making processor efficiency high. Conversely, if the section size is small, then a relatively large fraction of samples must be discarded for each section, resulting in poor processor efficiency, but relatively small memory requirements.

The two daughtercards assumed to be available in our approach were: Type 1, which had six SHARC® processors and a total of 32MB of memory; and Type 2, which had two SHARC® processors and a total of 64MB of memory. Thus, our optimization procedure was based on minimizing total consumed power based on proper selection of three parameters: section size, number of Type 1 cards, and number of Type 2 cards. Note that allowing two daughtercards in the configuration put additional constraints on the types of configurations that were possible. Thus, in general, arbitrary numbers of processors and amounts of memory could not me configured. However, the underlying concept of trading the efficiency of processors for more memory was still present.

One interesting lesson learned from our study happened when we considered a situation in which only Type 1 cards were assumed to be available for configuring the system (recall that the Type 1 card is “processor rich” and “memory poor” as compared with the Type 2 card). For this case of configuring only with Type 1 cards, the optimization procedure selected *very* small section sizes – smaller than one would think to be reasonable. We had to think about why this was happening; it went against our intuition. After some thought, we realized the reason – the objective of our optimization, afterall, was to *minimize consumed power, not to maximize processor efficiency*. The mathematical programming procedure had no regard for processor efficiency; its only concern was to use the available resources (in this case a lot of processors, and not much memory) to *minimize total consumed power*. If that means inefficient use of the processors, then so be it.

Consider why it is generally not optimal to force our expectations about what “reasonable” processor efficiencies should be for the case discussed in the previous paragraph. To achieve such efficiencies may require substantial memory (refer to Figure

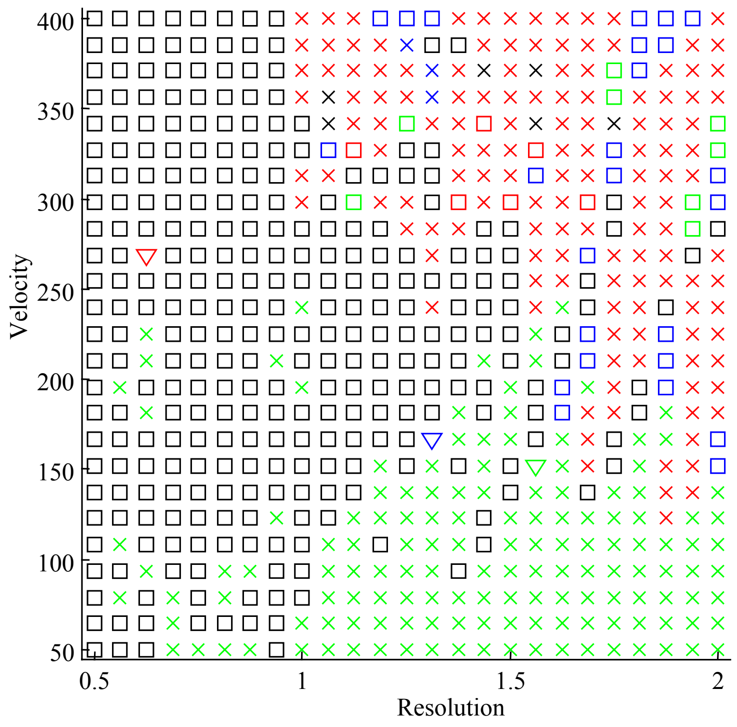
2). So, if “reasonable” processor efficiencies are forced into the configuration, then the number of cards required by the configuration must increase – not because more *processors* are required, but because more *memory* is required. In fact, some processors will be idle while the few “efficient ones” are working away – the resource being fully used is the memory. Recall that consumed power is in direct proportion to the number of cards in the configuration. This helped us understand a new interpretation for what our optimization procedure was actually doing: piecing together the “pre-configured silicon” cards available in the most power efficient way possible. Forget about the importance of processor efficiencies that we study/teach in our parallel processing courses!

References [2B] and [3] further refine the results of [1A]. The most notable refinement involves the concept of configuring a compute node. In the Mercury system, a compute node (CN) is an entity on a daughtercard consisting of one or more compute elements (CEs). A compute element, in this context, is a SHARC® processor. In our study, the Type 1 cards were populated with CNs in which each CN contains 3 CEs; and the Type 2 cards were populated with CNs in which each CN contains 2 CEs. In [2B] and [3], we defined formulations to our optimization problem in which the utilization of each CN is determined by the optimization procedure.

Figure 3 illustrates optimal configurations for a wide range of SAR operating points. The horizontal resolution axis represents the desired SAR image resolution in meters, and the vertical velocity axis is the speed of the vehicle (e.g., UAV) in meters/sec. The legend on the right side of the figure indicates two possible choices ($X$ and $Y$) for CN configurations. The value of $X_T$ and $Y_T$ indicate the card Type (1 or 2) selected for the $X$ and $Y$ configurations. For example, the red square symbol ‘□’ is associated with the use of card Type 1 for the $X$ configuration (i.e., $X_T = 1$) and card Type 2 for the $Y$ configuration (i.e., $Y_T = 2$). Furthermore, for the $X$ configuration, one CE (for each CN) is utilized for range processing (i.e., $X_r = 1$) and two CEs are used for azimuth processing (i.e., $X_a = 2$). Similarly, for the $Y$ configuration, none of the CEs are used for range processing, and both CEs (for each CN) are used for azimuth processing (because $Y_r = 0$ and $Y_a = 2$). For the sake of comparison, consider now the configurations associated with the blue times symbol ‘×’ where both the $X$ and $Y$ configurations use the Type 1 card, but the utilization of the CNs for $X$ and $Y$ are distinct. The number of configured CNs, and thus the total number of cards of each type, is also provided by the optimization procedure, but is not shown on Figure 3.

Although subtle, perhaps, this part of the work is extremely important because it cuts to the heart of a bigger issue. The most fundamental questions of interest for these types of systems should not necessarily be expressed in terms of processor efficiencies, or even processors or memories at all; what is important is the “configuration of the silicon,” i.e., how can it be configured to minimize SWAP. The mixing of the two card types we studied is only a rough approximation to this general concept of “configurable silicon.” With two discrete card types available, many, but not anywhere near all, possible combinations of processors and memories can be configured. But remember, processors and memory are not the only things we can build out of silicon. More specialized functional units can also be built.

Parts 3 and 4 of this report deal with a key aspect of the project – namely, is it always necessary to configure silicon as discrete processor and memory modules? Could it be that silicon configurations consisting of modules or functional units less complex than processors and memories are also possible, and have superior SWAP characteristics in some situations? Before getting to the answers to these questions, the next part of this report deals with optimizing the SWAP performance of a multiprocessor implementation for STAP. Although Part 2 is similar to Part 1 in the sense that only processors and memories (and not reconfigurable computing) are assumed in the computing platform, the mechanism for minimizing SWAP in the STAP application centers around effective use of the interconnection network that supports interprocessor communication.



| | $X_T X_r X_a$ | $Y_T Y_r Y_a$ |
|---|---|---|
| × | 1 1 2 | |
| × | 2 1 1 | |
| × | 1 1 2 | 1 2 1 |
| × | 1 1 2 | 2 0 1 |
| □ | 1 1 2 | 2 0 2 |
| □ | 1 1 2 | 2 1 1 |
| □ | 1 2 1 | 2 0 2 |
| □ | 1 3 0 | 2 0 2 |
| ▽ | 1 3 0 | 2 1 1 |
| ▽ | 2 0 2 | 2 1 1 |
| ▽ | 2 1 1 | 2 2 0 |

Figure 3. Optimal CN Configurations of the CN-constrained Model [2B].

## Part 2: Optimal Communication Scheduling for STAP

*Overview of References [4C], [5D], [6E], [7], and [8]*

The work here develops and evaluates a genetic-algorithm-based (GA-based) optimization technique for the scheduling of messages for a class of parallel embedded signal processing techniques known as space-time adaptive processing (STAP). The GA-based optimization is performed off-line, resulting in static schedules for the compute nodes of the parallel system. These schedules are utilized for the on-line implementation of the parallel STAP application. The primary motivation and justification for devoting significant off-line effort to solving the formulated scheduling problem is the resulting reduction of hardware resources required for the actual on-line implementation. Studies illustrate that reductions in hardware requirements of around 50% can be achieved by employing the results of the proposed scheduling techniques. This reduction in hardware requirement is of critical importance for STAP, which is typically an airborne application in which the size, weight, and power consumption of the computational platform are often severely constrained.

For an application implemented on a parallel and embedded system to achieve required performance, it is important to effectively map the tasks of the application onto the processors in a way that reduces the volume of inter-processor communication traffic. It is also important to schedule the communication of the required message traffic in a manner that minimizes network contention so as to achieve the smallest possible communication times.

Mapping and scheduling can both – either independently or in combination – be cast as optimization problems, and optimizing mapping and scheduling objectives can be critical to the performance of the overall system. For embedded applications, great importance is often placed on determining minimal hardware requirements that can support a number of different application scenarios. This is because there are typically tight constraints on the amount of hardware that can be accommodated within the embedded platform. Using mappings and schedules that minimize the communication time of parallel and embedded applications can increase the overall efficiency of the parallel system, thus leading to reduced hardware requirements for a given set of application scenarios.

The work here focuses on using a GA-based approach to optimize the scheduling of messages for STAP algorithms. STAP is an adaptive signal processing method that simultaneously combines signals received from multiple elements of an antenna array (the spatial domain) and from multiple pulses (the temporal domain) of a coherent processing interval. The focus of this research assumes STAP is implemented using an element-space post-Doppler partially adaptive algorithm; refer to references [6E], [7], and [8] for details.

STAP involves signal processing methods that operate on data collected from a set of spatially distributed sensors over a given time interval. Signal returns are composed of range, pulse, and antenna-element digital samples; consequently, a three-dimensional (3-D) data cube naturally represents STAP data. A distributed memory multiprocessor machine is assumed here for the parallel STAP implementation. The core processing requirement proceeds in three distinct phases of computation, one associated with each dimension of the STAP data cube. After each phase of processing, the data must be re-distributed across the processors of the machine, which represents the communication requirements of this parallel application. Thus, there are two primary phases of inter-processor data communication required: one between the first and second phases of processing and one between the second and third phases of processing. After all three phases of processing are complete for a given STAP data cube, a new data cube is input into the parallel machine for processing.

A proposed GA-based approach is used to solve the message-scheduling problem associated with each of the two phases of inter-processor data communication. This GA-based optimization is performed off-line, and the results of this optimization are static schedules for the compute nodes of the parallel system. These schedules are used within the on-line parallel STAP implementation. The results of the study show that significant improvements in communication time performance are possible using the proposed approach for scheduling. It is then shown that these improvements in communication time translate to reductions in required hardware for a class of scenarios. Performance of the mappings and schedules are evaluated based on a Mercury RACEway® network simulator developed under this project and described in references [4C] and [7].

For this work, the STAP data cube is partitioned into sub-cube bars of vectors where each vector is mapped onto a given CN (compute node), refer to [6E] for more details. A two-dimensional process set, as described in [8], defines the mapping of data onto CNs for each computational phase. Additionally, the process set defines the communication pattern for the required "distributed corner turns" of the STAP data cube.

Summarizing the results published in [6E] and [8], it is demonstrated that off-line GA-based message scheduling can significantly improve the communication performance in a parallel system. When compared to baseline and randomly generated schedules, the GA-based schedules are significantly superior – typically reducing communication times by between 20% and 50%, see [8] for details.

Interestingly, it is shown that the best *mapping* – defined as a mapping that minimizes a mapping objective function – is not always the best choice in terms of minimizing overall communication time. In particular, as the number of CNs is increased, optimal mappings that require only one phase of communication generally report higher overall communication times than those good (but not optimal) mappings that require two non-trivial phases of communication.

The optimization of mapping and scheduling, either independently or in combination, is critical to the performance of the STAP application for embedded parallel systems. For such systems, great significance is placed on minimizing overall execution time, which includes both computation and communication components. Such reductions in execution time also translate into improved hardware efficiency and thus reduced hardware requirements, which is often critical.

Through extensive numerical studies, it is shown in [6E] and [8] that the GA-based optimization approaches can yield mappings and schedules that greatly improve the online performance and reduce the hardware requirements of the parallel embedded system. Examples are provided that illustrate the optimal mapping and scheduling methodologies of [6E] and [8] can produce hardware savings of 50% and more when compared to typical solutions to the mapping and scheduling problems that might be employed by practitioners. Because of limitations on the size of problems that were executed/simulated, systems up to a size of only 32 processors were investigated. However, from the trends observed in overall completion times, it is apparent that even more significant savings in hardware/power requirements are realizable for STAP applications that require substantially larger systems having hundreds or even thousands of processors.

## Part 3: FPGA Power Prediction and Applications

We discovered during the project period that predicting power consumption for an FPGA is a very difficult task. There were no commercially available tools that accurately predicted power consumption for any of the existing FPGAs. Thus, a major focus of this part of the work involved the development of accurate methods for predicting FPGA power consumption. References generated by this project in the area of power prediction include [9F], [10], and [11G], which are overviewed in Section 3.1.

In addition to trying to understand and predict FPGA power consumption, we also studied the types of computations that could be effectively mapped onto FPGAs. In theory, given enough gates, one could imagine configuring an FPGA board to behave as a microprocessor. Thus, again in theory, an FPGA board could be used to perform any type of calculation. However, based on the available technology, this would be extremely impractical. Our goal was to therefore use FPGAs to devise useful modules that are much less complex than a microprocessor, thereby reducing the SWAP overhead inherent when computations are performed only on microprocessors and/or DSPs. So, one of our aims was to characterize the types of computations that can be practically implemented in FPGAs. References produced in the area of mapping applications onto FPGAs include [12H], [13], [14], [15I], [16J], and [17], and these are overviewed in Section 3.2.

### 3.1 FPGA Power Prediction

*Overview of References [9F] and [10]*

The work published in [9F] and [10] describes a practical and accurate power prediction tool for the Xilinx® 4000-series FPGA. The utility of the tool is that it enables FPGA circuit designers to evaluate the power consumption of their designs without resorting to the laborious and expensive empirical approach of instrumenting an FPGA board/chip and/or taking actual power consumption measurements. Preliminary evaluation of the tool indicates that an error of less than 5% is usually achieved when compared with actual physical measurements of power consumption.

The tool, which is implemented in Java, takes as input two files: (1) a *configuration file* associated with an FPGA design and (2) a *pin file* that characterizes the signal activities of the input data pins to the FPGA. The configuration file defines how each CLB (configurable logic block) is programmed and defines signal connections among the programmed CLBs. The configuration file is a text file that is generated using a Xilinx® M1 Foundation Series utility called *ncdread*. The pin file is also a text file, but is generated by the user. It contains a listing of pins that are associated with the input data for the configured FPGA circuit. For each pin number listed, probabilistic parameters are provided which characterize the signal activity for that pin.

Based on the two input files, the tool propagates the probabilistic information associated with the pins through a model of the FPGA configuration and calculates the activity of every internal signal associated with the configuration. The activity of an internal signal *s*, denoted $a_s$, is a value between zero and one and represents the signal's relative frequency with respect to the frequency of the system clock, *f*. Thus, the average frequency of signal *s* is given by $a_s f$.

Computing the activities of the internal signals represents the bulk of computations performed by the tool. Given the probabilistic parameters for all input signals of a configured CLB, the probabilistic parameters of that CLB's output signals are determined using a mathematical transformation. Thus, the probabilistic information for the pin signals is transformed as it passes through the model of the configured logic, defined by the configuration file. However, the probabilistic parameters of some CLB inputs may not be initially known because they are not directly connected to pin signals, but instead are connected to the output of another CLB for which the output probabilistic parameters have not yet been computed (i.e., there is a feedback loop). For this reason, the tool applies an iterative approach to update the values for unknown signal parameters. The iteration process continues until convergence is reached, which means that the determined signal parameters are consistent based on the mathematical transformation that relates input and output signal parameter values, for every CLB.

The average power dissipation due to a signal *s* is modeled by $\frac{1}{2} C_{d(s)} V^2 a_s f$, where $d(s)$ is the Manhattan distance the signal *s* spans across the array of CLBs, $C_{d(s)}$ is the equivalent capacitance seen by the signal *s,* and *V* is the voltage level of the FPGA device. The overall power consumption of the configured device is the sum of the power dissipated by all signals of the configured FPGA.

For the study conducted in [9F], a total of 70 power measurements were made using five different configuration files and fourteen different data sets. Descriptions of these configuration files and data sets are given in [9F]. Each of the configuration files used take a total of 32-bits of data as input. The first three configurations (fp_mult, fp_add, int_mult) each take two 16-bit operands on each clock cycle, and the last two (serial_fir and parallel_fir) each take one 32-bit complex operand on each clock cycle. The 32 bits of input data are numbered as 0 through 31, and two key parameters are used to characterize these bits: an *activity factor, a* and a *probability factor, p*. As mentioned earlier, the activity factor of an input bit is a value between zero and one and represents the signal's relative frequency with respect to the frequency of the system clock, *f*. The probability factor of a bit represents the fraction of time that the bit has a value of one.

Figure 4 shows plots of the measured power for all combinations of the configuration files and data sets considered. For all cases, the clock was run at $f = 30$ MHz. With the exception of the fp_mult configuration file, the most active data set file (number 6) is associated with the highest power consumption. Also, the least active data set file (number 5) is associated with the lowest power consumption across all configuration files. There is somewhat of a correlation between the number of components utilized by each configuration and the power consumption; however, it turned out that even though the serial_fir implementation is slightly larger than parallel_fir, it consumes less power. This is likely due to the fact that the parallel_fir design requires a high fan-out (and thus high routing capacitance) to drive the parallel multipliers.

In addition to the graph shown in Figure 4, additional figures are provided in [9F] that overlay estimates of power consumption predicted by the tool developed in this project.

As mentioned above, predicted values of power were generally within 5% of actual measured values.

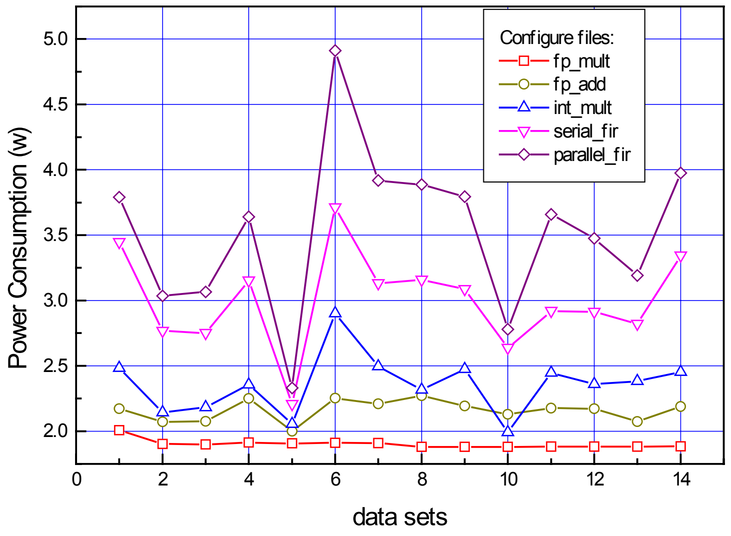


Figure 4. Measured power consumption of the configuration files and data sets from [9F].

*Overview of Reference [11G]*

The method used by the above tool to compute signal activities was based on a previously published approach from another research group. That approach has some difficulties, primarily related to its time complexity. In [11G], a new analytical approach was developed by us for calculating signal activities. Our approach is based on a Markov chain signal model, and directly accounts for correlations present among the signals. We verified the accuracy of the approach by comparing signal activity values calculated using our approach with corresponding values produced through simulation studies. It was also demonstrated that the proposed approach is much more computationally efficient than competing approaches. In addition to describing the new approach for calculating signal activities, [11G] also provides a comprehensive review of past approaches, including the approach implemented for the tool described in [9F] and [10].

## 3.2 FPGA Applications

*Overview of References [12H] and [13]*

In references [12H] and [13], techniques for mapping portions of space-time adaptive processing (STAP) computations onto FPGAs are described. The output of STAP is a weighted sum of multiple radar returns, where the weights for each return in the sum are calculated adaptively and in real-time. The most computationally intensive portion of most STAP approaches is the calculation of the adaptive weight values, which typically constitutes over 90% of all the computations needed in adaptive processing. Calculation of the weights involves solving a set of linear equations based on an estimate of the covariance matrix associated with the radar return data. The traditional approach for computing the adaptive weights is based on a direct method called QR-decomposition. This method has a fixed computational complexity, which depends on the size of the equation matrix and provides the exact solution. An alternative approach based on an iterative method called Conjugate Gradient was investigated, which allows for trading off accuracy for reduced computational complexity. The two approaches are analyzed and compared in [13]. The results show that the Conjugate Gradient approach can reduce the computations needed at the cost of reduced accuracy in some cases.

Existing computational strategies for STAP typically rely exclusively on the use of multiple DSPs and/or GPPs. An alternative strategy is proposed in [12H] and [13], which makes use of FPGAs as vector co-processors that perform inner product calculations. Two different "inner-product co-processor" designs are introduced for use with a host DSP or GPP. The first has a multiply-and accumulate structure and the second uses a reduction-style tree structure having two multipliers and an adder. For a fixed clock rate, the second design can provide a higher throughput, but requires more computation from the host (to perform the final summation of the partial sums).

In the work of [12H] and [13], the two inner-product co-processors were implemented using a block floating point format, which is much simpler to implement than standard floating point units. We also investigated overall accuracy of block floating point versus full floating point. It was demonstrated that the block floating point co-processors produce acceptable accuracy results for input data distributions that are uniformly distributed. Poor results are obtained, however, for cases where one or a few of the elements are much larger than the rest of the numbers. This is because the block-floating-point architecture normalizes all the exponents to the maximum exponent by shifting out the least significant bits of the mantissa so that all the exponents are equal, and then all the operations are integer arithmetic operations (based on the resulting mantissas), which are much easier to perform than general floating-point operations. The shifting out of the bits produces inaccuracy in the computations. For all the ranges of numbers considered, if the numbers are uniformly distributed, then the exponent distribution has an increasing exponential shape with a majority of the numbers close to the maximum value in the exponent domain. This results in a small number of bits from the mantissas of the numbers being shifted out, on the average. Another important point is that the multiply implementation uses a 15-bit mantissa, which implies that the mantissa of the input floating-point number is truncated to 15 bits from 23 bits, which itself introduces some inaccuracies.

### *Overview of Reference [14]*

In reference [14], further studies of inner-product co-processor designs were conducted. In contrast to the inner product designs of [12H] and [13], which were based on a block floating point format, both floating point and integer formats were used in [14], both using 16-bit formats. The studies demonstrated that inner-product co-processors, for both integer and floating-point data, could fit into current (at that time) FPGA technology and achieve significant speed and throughput. The results of the implementations show that it is feasible and beneficial under certain circumstances to implement floating-point and integer operations in FPGAs (i.e., such as when a custom data format can be used, as with the SHARC® DSP which can convert back and forth between IEEE 32-Bit floating point and the SHARC® DSP 16-bit floating point formats).

The studies in [14] also considered the advantages and disadvantages of employing different degrees of pipelining in the inner product designs. One interesting (and somewhat counterintuitive) outcome related to pipelined versions of the designs was that adding more pipeline stages did not always allow for an increased clock speed at which the circuit could be executed. This was due to the fact that adding in the pipeline stages also added more overall complexity, which made it more difficult for the place-and-route routines of the FPGA design tool to find good implementations. Thus, as more pipelined stages were added, critical signal lengths sometimes increased, dictating that the clock rate actually had to be decreased. Estimates of power consumption were also evaluated for all designs considered in [14].

*Overview of Reference [15I]*

Two major contributions are presented in [15I]. First, it is shown that the core computations from the SAR application, including both the range compression and azimuth processing phases, can be structured as a single deep computational pipeline that can be implemented directly on an array of FPGAs. Past results for high-throughput SAR processing (e.g., refer to [1A], [2B], and [3]) typically assume the computations are to be mapped onto a distributed memory multiprocessor system in which a subset of the available compute elements (CEs) is assigned to perform range processing and the remaining CEs perform azimuth processing. In this type of traditional approach, a number of processed range vectors are sent from the range CEs to the azimuth CEs where they are buffered in memory. After a prescribed number of compressed range vectors are present in the memory space of the azimuth CEs, azimuth processing commences on the azimuth CEs. Because of the significant intermediate buffer storage required by this approach, and the associated placing and fetching of data in this memory space by the range and azimuth CEs, respectively, this type of SAR implementation is generally not thought to be "purely streaming." However, as is presented in [15I], these computations (both phases) can in fact be structured as a single computational pipeline, which can be directly mapped onto an array of FPGAs.

In the proposed approach, no intermediate memory buffer is required between the two phases of computation. Instead, within the structure of the computational pipeline are long segments of delay elements that effectively provide the intermediate storage associated with the more traditional approach. Figure 5 illustrates the structure of the computational pipeline. In the figure, small values of parameters are used for the purpose minimizing the size of the pipeline, while still illustrating its basic structure. Realistic parameters values would be on the order of thousands, resulting in a pipeline with millions of registers. Further details on sizing analysis and hardware comparisons between a deep pipeline implementation versus a multiprocessor implementation are provided in the online link to the presentation materials for reference [15I].

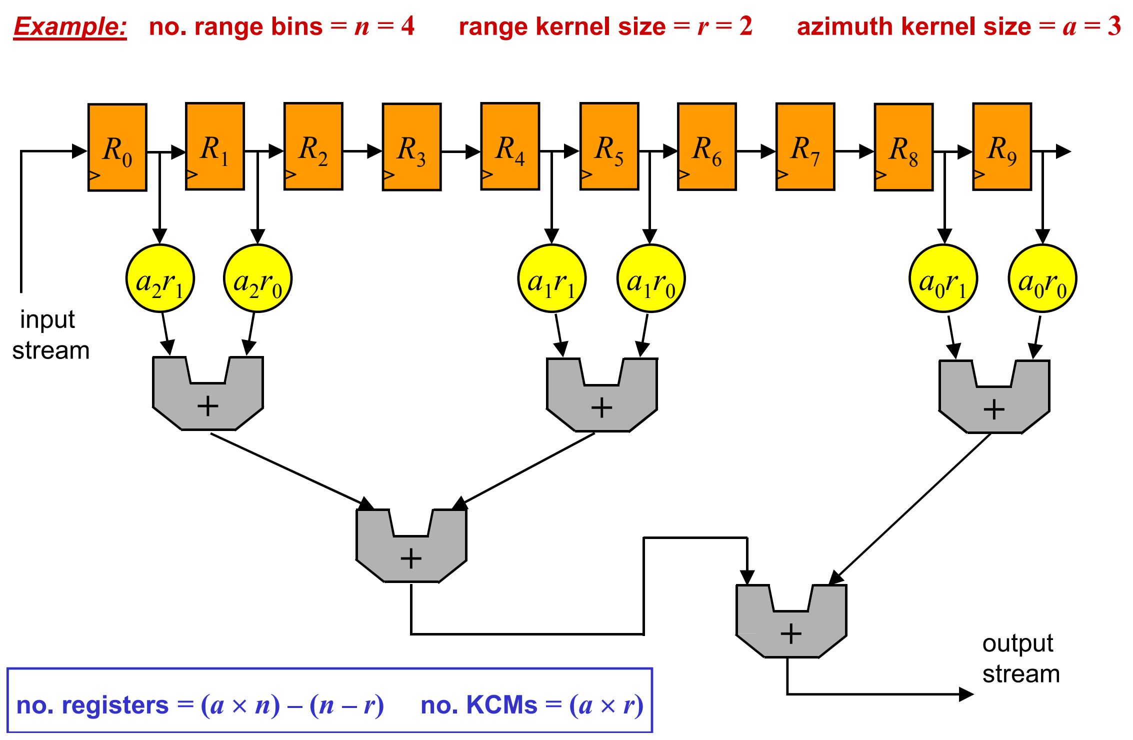

Example: no. range bins = $n$ = 4 range kernel size = $r$ = 2 azimuth kernel size = $a$ = 3
$R_0$
$R_1$
$R_2$
$R_3$
$R_4$
$R_5$
$R_6$
$R_7$
$R_8$
$R_9$
$a_2r_1$
$a_2r_0$
$a_1r_1$
$a_1r_0$
$a_0r_1$
$a_0r_0$
input stream
+
output stream
no. registers = $(a \times n) - (n - r)$ no. KCMs = $(a \times r)$

Figure 5. Structure of the deep pipeline.

One potential advantage of the proposed approach is that data need not be continuously stored and then fetched from a separate memory module by CEs (which, incidentally, can require significant power consumption). Instead, the data streams continuously through a long computational pipeline. Within this pipeline are the taps of the FIR (finite impulse response) implementations of both the range and azimuth processing, interspersed with segments of delay elements. Although the resulting pipeline may be thousands of stages long for practical values of SAR parameters, it is a viable approach because end-to-end latencies on the order of 1 millisecond are typically acceptable, provided that the required throughput is achieved.

The second contribution presented in [15I] demonstrates how signal activity parameters of incoming data can be transformed, before the data are processed by a computational pipeline, as a means of reducing overall power consumption. The key to understanding this approach is the realization that the activity levels of the input signals to the computational pipeline dictate its level of power consumption. The activity of a given input signal (i.e., bit position) is defined as the fraction of time that the signal transitions relative to the system clock. We demonstrated that increasing/decreasing the signal activities of input data to a pipelined circuit implemented on an FPGA also increases/decreases the power consumption of the circuit. In [15I], we introduce a concept for how the activities of the input data can be transformed (pre-processed) so that the resulting (transformed) signals that are input into the computational pipeline have activity values that are well-matched with the pipelined circuit in terms of minimizing consumed power. At the end of the computational pipeline, an inverse transformation is applied to the output values to convert them back to their proper (and meaningful) representation. This concept is illustrated in Figure 6. The approach is based on two fundamental assumptions: (1) that the power consumption of the computational pipeline is significantly higher than that of the computational structures implemented to perform the transform and inverse transformation of the data and (2) that the computations performed within the computational pipeline are linear and time invariant.

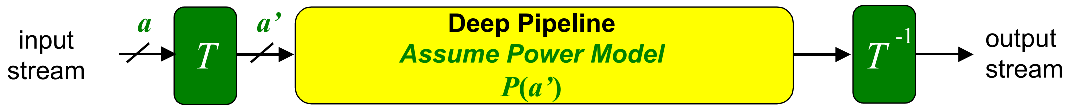


Figure 6. Using activity transformations to minimize power consumption.

*Overview of References [16J] and [17]*

References [16J] and [17] present a comparative study of different parallel prefix circuits from the point of view of power-speed trade-off. The prefix circuit plays an important role in many applications such as the carry-look-ahead adder, ranking, packing, and radix sort. The power consumption and the power-delay product of seven parallel prefix circuits were compared. By assuming a linear capacitance model, combined with PSpice® simulations, we investigated the power consumption in the parallel prefix circuits. The degrees of freedom studied include different parallel prefix architectures and voltage scaling. The results show that the use of the linear output capacitance assumption provides power estimates that are consistent with those obtained using PSpice® simulations. It was found that the divide-and-conquer prefix circuit, which is the fastest circuit considered, consumes the most power. Also – according to PSpice® simulations – the power-delay product of the LYD (Lakshmivarahan-Yang-Dhall) prefix circuit was the best (i.e., lowest) among the circuits studied, while the power-delay product of the divide-and-conquer was the highest. This study demonstrates the importance of careful analysis of the speed-power trade-off when considering architectural choices for implementing a given computational function in hardware.

## Part 4: Hybrid FPGA/DSP/GPP Platform

*Overview of Reference [18K]*

The prototype platform was developed to demonstrate the advantages and trade-offs associated with the combined use of different hardware technologies for two embedded radar-processing applications, namely SAR and STAP. The primary metrics of interest are size, weight, and power utilizations. The developed system can be configured with FPGAs, DSPs, and/or GPPs. Although the prototype system was not evaluated through fielded studies, experiments involving continuous input streams at relatively high rates were conducted in the laboratory using unprocessed radar data as input.

The FPGA components of the prototype system are commercially available WildOne™ and WildForce™ boards (from Annapolis Microsystems) populated with 4000-series Xilinx® parts. The WildForce™ boards each have four 4085-series FPGAs plus one control FPGA. The DSP/GPP components of the system are within a Mercury Race Multicomputer configured with both SHARC® and PowerPC® CNs. The Mercury system can be configured with up to eight PowerPC® nodes and eight SHARC® compute nodes (each SHARC® CN actually contains three SHARC® DSP chips).

An overview of the overall architecture is depicted in Figure 7. A more detailed view of the major components of the hybrid system are illustrated in Figure 8, and a photograph of the actual prototype system is provided in Figure 9.

The source PC is responsible for initially loading unprocessed radar data (from disk) into a circular buffer within its main memory. Once the input data is loaded into the circular buffer, the source PC then continuously (and repeatedly) streams this data into the front-end FPGA subsystem, denoted as (F) in Figures 7, 8, and 9. It was necessary to locate the input data in a large main memory buffer in order to achieve realistic data throughput rates, which would otherwise not be possible if the data were streamed directly from the disk of the source PC. All of the Annapolis FPGA boards are PCI-based and reside on the data source and/or data sink PCs. A total of four WildForce™ boards are available, and zero or more of these may reside on the source and sink PCs. The source and sink PCs also contain one WildOne™ board each. The WildOne™ boards are not used for computation; they handle the data communication (through the PCI bus) between the PCs and the FPGA subsystems. The data communication among all FPGA boards is through two types of 36-bit wide connectors, one called systolic and one called SIMD.

The data communication between the front-end FPGA subsystem (F) and the DSP/GPP subsystem is a custom interface developed using the systolic connector from Annapolis and the RIN-T input device from Mercury. Similarly, the data communication between the DSP/GPP subsystem and the back-end FPGA subsystem (B) is through a custom interface developed using the ROUT-T output device from Mercury and the systolic connector from Annapolis. More details on the design of the interfaces between the Mercury and the front- and back-end subsystems are provided in Figures 10 and 11, respectively.

## Hybrid FPGA/DSP/GPP Prototype Architecture

**Block Diagram**

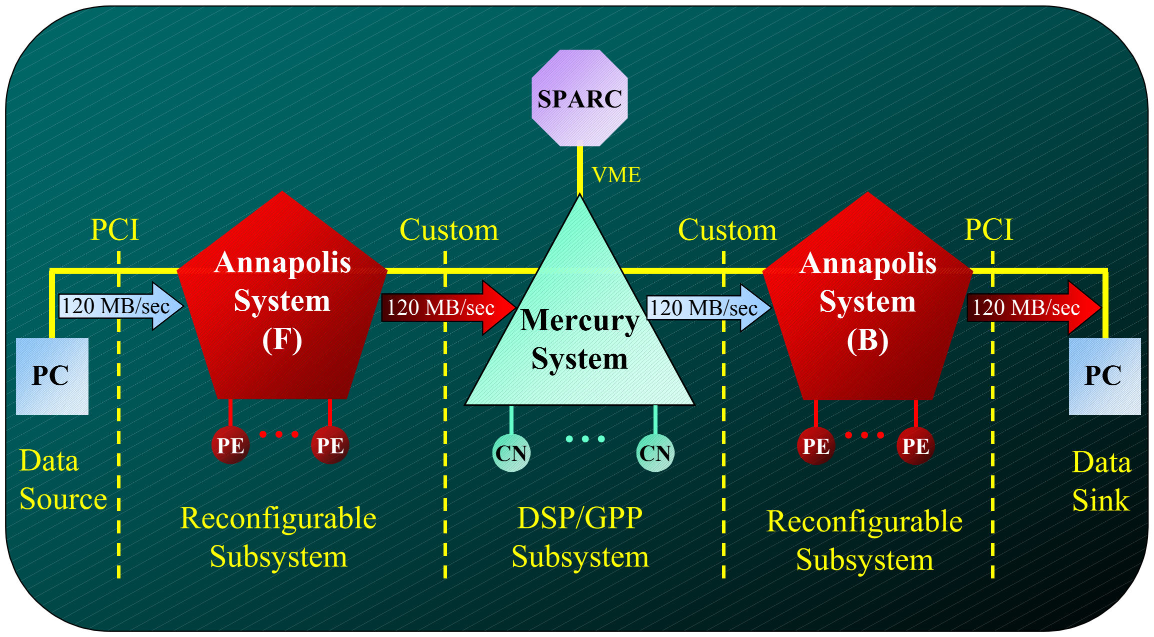


Figure 7. Block diagram of the FPGA/DSP/GPP prototype architecture.

Design and implementation of the interface connecting the Mercury to the back-end FPGA subsystem (B), shown in Figure 11, was particularly challenging. The clock signal used to strobe the data from the Mercury was not programmable; it was fixed at 33 MHz. It turned out that the input impedance of the back-end FPGA subsystem was not very well matched with the output of the Mercury subsystem. As a result, the maximum clock rate possible was only about 8Mhz, or about one-fourth of fixed 33Mhz clock available. So, we implemented a scheme in which four copies each data word was transmitted from the Mercury, which effectively reduced the clock rate by a factor of four. We also had to include a packing scheme, which encoded two bits of each transmitted word to enable detection of the boundary between groups of copied data. This was necessary because the actual number of copies of each word received by the back-end GPGA subsystem was unpredictable, and varied between two and four. More details on this scheme can be found at the online link to the presentation materials for reference [18K].

Figures 12 and 13 illustrate how the major computational components of the SAR and STAP applications can be mapped onto the prototype system. A candidate mapping is defined by assigning the computations of each major component to one or both of the symbols shown in each block (which correspond to one of the FPGA or DSP/GPP subsystems defined in Figure 7). Using SAR to illustrate, one mapping would be to perform all of the range compression on the front-end FPGA subsystem (F) and then perform all azimuth processing on the DSP/GPP subsystem. Another possible mapping is defined by using the FPGA subsystems and the DSP/GPP for both components of computation. It is also possible to use only the DSP/GPP subsystem for both components of computations.

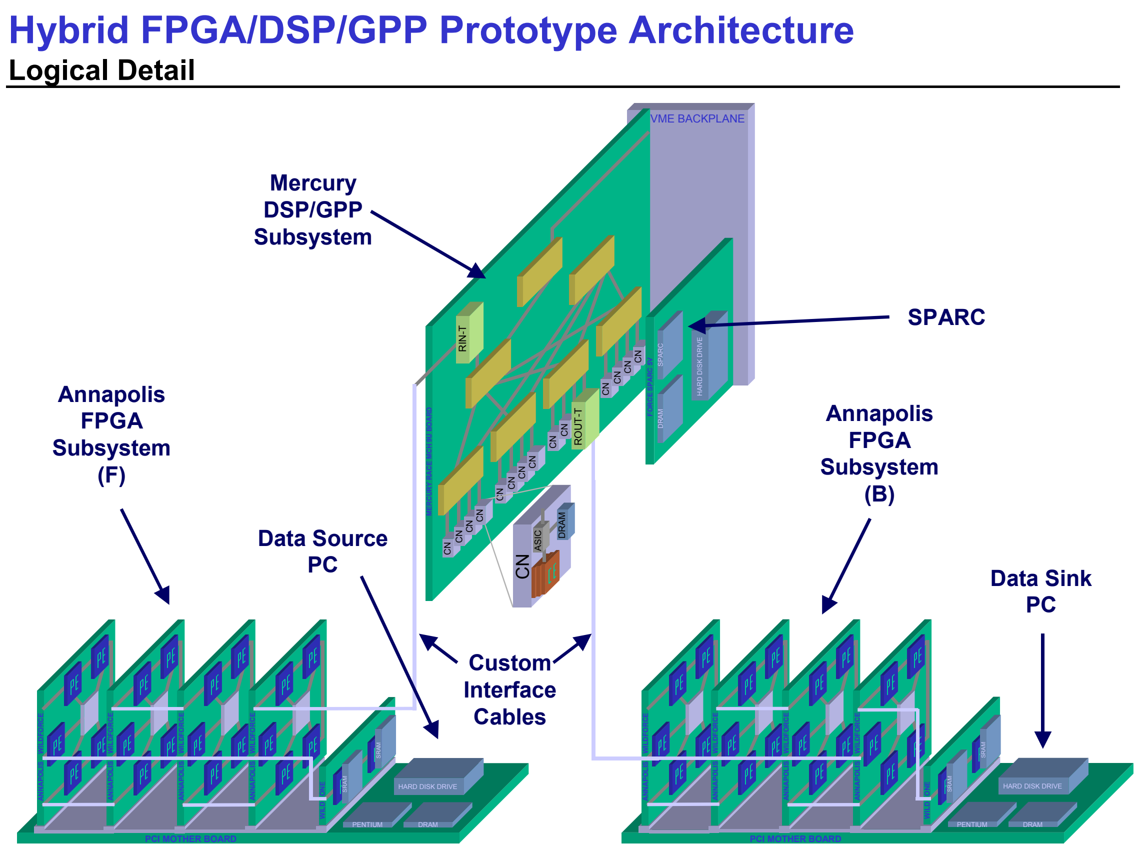


Figure 8. Detail of the FPGA/DSP/GPP prototype architecture.

The SAR studies were designed by adapting the RASSP (Rapid Prototyping of Application Specific Signal Processors) benchmark developed originally by Lincoln Laboratory at MIT. The benchmark, which was originally implemented in serial C code, was first modified to execute on the parallel DSP/GPP subsystem. A data-streaming component was also added so that input data can be sent continuously from the data source of the prototype system. Core computations from the range compression and azimuth processing components were implemented for the FPGA subsystems, as described earlier in Part 3 of this report.

An overview of SAR processing flow is provided in Figure 14. The data distribution scheme for SAR is illustrated in Figure 15. For the case shown in the figure, a total of eight CNs were utilized: two SHARC® CNs (one for input and the other for output) and six PowerPC® CNs (two for range processing and four for azimuth processing). A detailed timing diagram is shown in Figure 16. Note from this figure that the processing is well balanced and that the amount of idle time for each CN is relatively small. A summary of time and throughput results are provided in Figure 17. Note that the required input and output throughputs realized for this particular study, 0.71 Mbytes/sec and 1.42 Mbytes/sec, are well within the maximum capacity supported by the custom interfaces of 60 Mbytes/sec and 31 Mbytes/sec (refer to Figures 10 and 11). This implies that the constructed prototype system is capable of processing much more intensive instances of SAR processing.

**Hybrid FPGA/DSP/GPP Prototype Architecture**

**Photograph**

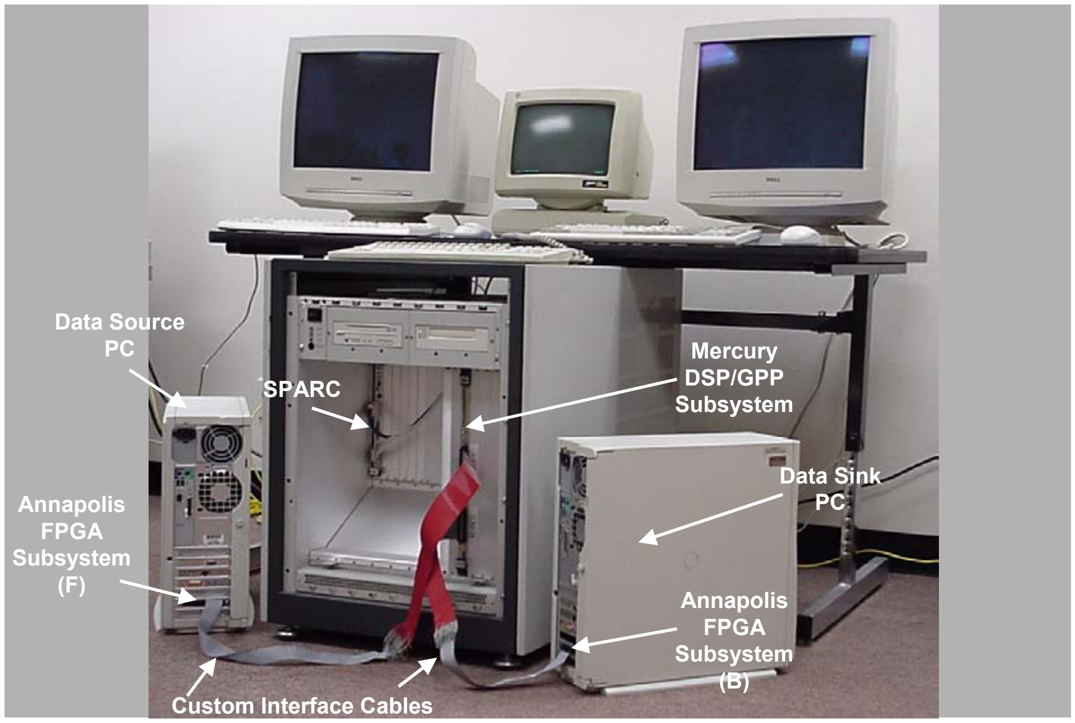

Data Source PC
SPARC
Mercury DSP/GPP Subsystem
Data Sink PC
Annapolis FPGA Subsystem (F)
Annapolis FPGA Subsystem (B)
Custom Interface Cables

Figure 9. Photograph of the FPGA/DSP/GPP prototype architecture.

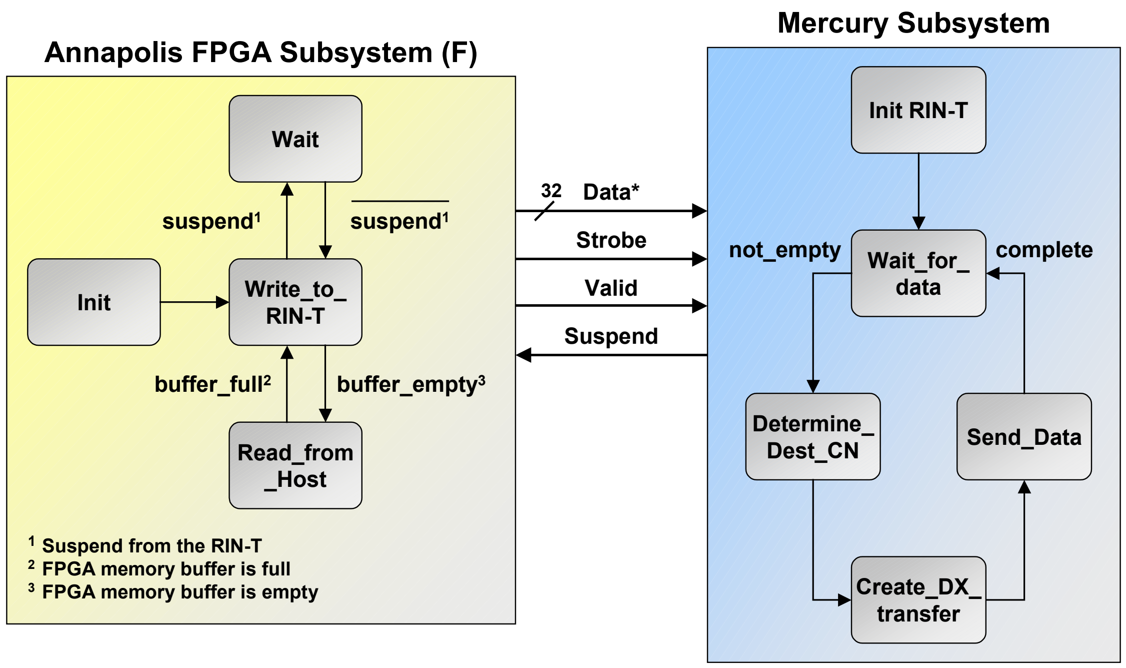
The STAP studies were designed by adapting the RT_STAP (Real Time STAP) benchmark developed originally at the MITRE Corporation. This benchmark was already implemented for parallel execution on a PowerPC-based Mercury system. This implementation was expanded to also enable execution on SHARC® compute nodes. The same basic data streaming component that was developed for SAR was also adapted to enable the STAP input data to be sent continuously from the data source. Core computations from the range compression and weight computation components from the STAP processing flow were implemented for the FPGA subsystems.

Similar to the figures associated with SAR, an overview of the scheme used to stream STAP processing is provided in Figure 18. Note from the figure that two SHARC® compute nodes are used for I/O and eight PowerPC® are used to actually perform the

STAP computations (for the particular instance of STAP considered). Unlike SAR, where CNs are dedicated exclusively to one particular phase of the computation, in the STAP implementation all CNs work on all three phases of computation. Figure 19 illustrates the three phases of computation required by STAP and the two communication phases (i.e., re-partitioning of the data cube) between the three phases. A space-time diagram is provided in Figure 20 followed by a summary of obtained throughput results in Figure 21. As was the case for SAR, note from Figure 21 that the required input and output throughputs realized for this particular study are well within the maximum capacity supported by our custom interfaces.

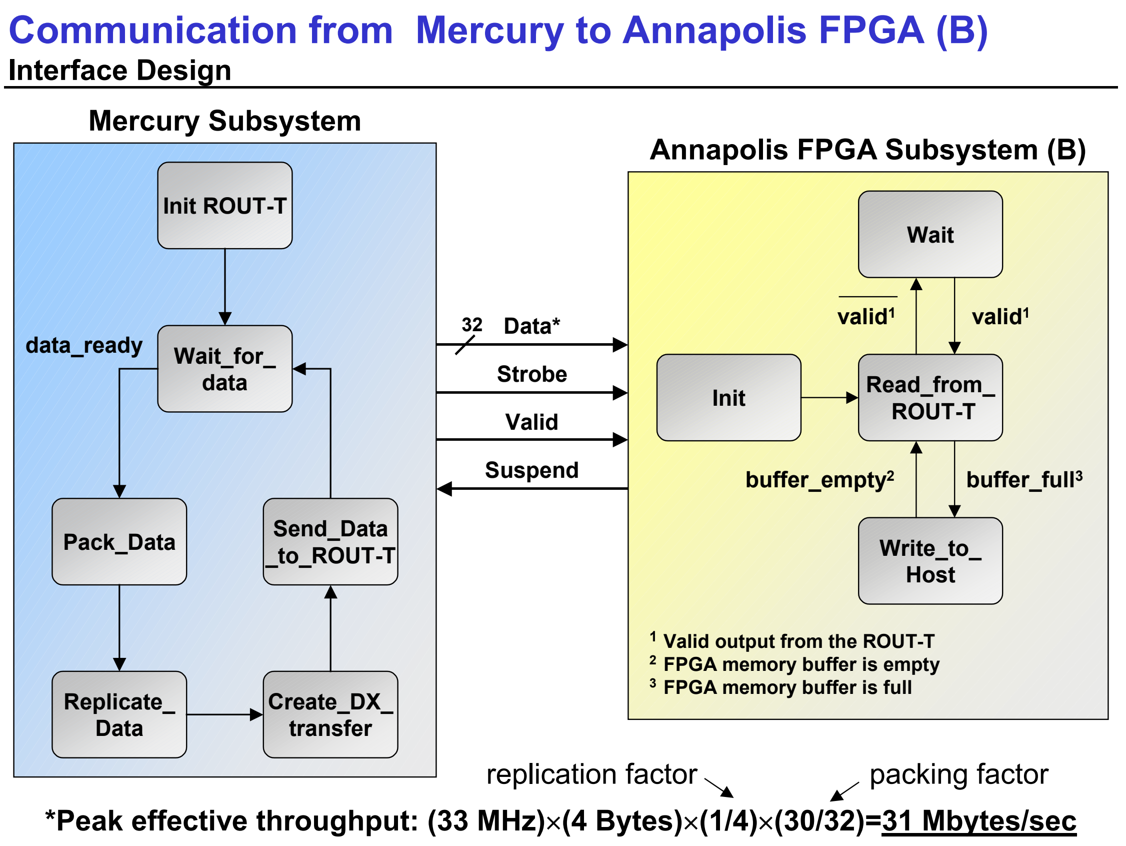
## Communication from Annapolis FPGA (F) to Mercury

**Interface Design**



***Peak throughput achieved to date: (15 MHz) × (4 Bytes) = <u>60 Mbytes/sec</u>**

Figure 10. Interface Design: Communication from Annapolis FPGA (F) to Mercury.



**Communication from Mercury to Annapolis FPGA (B)**

**Interface Design**

***Peak effective throughput: (33 MHz)×(4 Bytes)×(1/4)×(30/32)=31 Mbytes/sec**

Figure 11. Interface Design: Communication from Mercury to Annapolis FPGA (B).

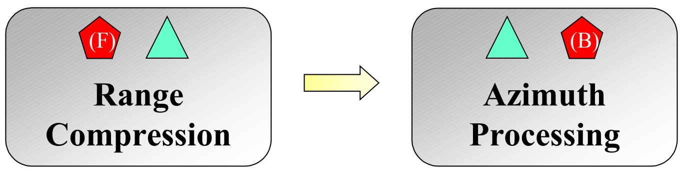


Figure 12. Illustration of how the major computational components of SAR processing can be mapped onto the hybrid system.

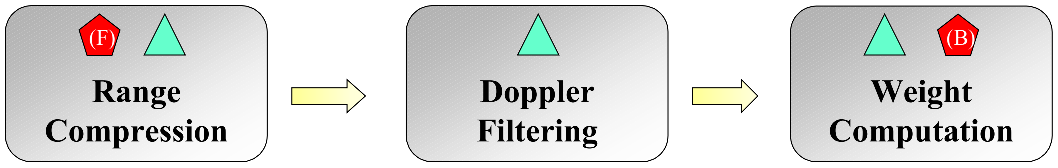


Figure 13. Illustration of how the major computational components of STAP processing can be mapped onto the hybrid system.

## SAR Processing Flow*

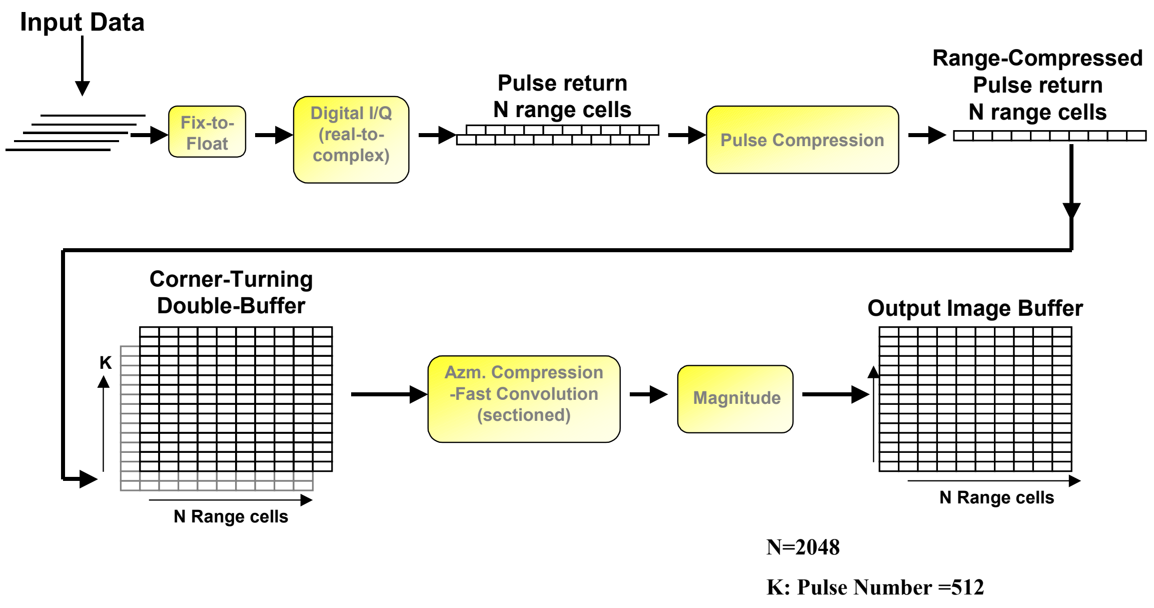

Input Data
Fix-to-Float
Digital I/Q (real-to-complex)
Pulse return N range cells
Pulse Compression
Range-Compressed Pulse return N range cells
Corner-Turning Double-Buffer
K
N Range cells
Azm. Compression -Fast Convolution (sectioned)
Magnitude
Output Image Buffer
N Range cells
N=2048
K: Pulse Number =512

*Figure Derived from:T. Einstein, “Realtime Synthetic Aperture Radar Processing on the RACE Multicomputer,” App. Note 203.0, Mercury Computing Sys, 1996.

Figure 14. Figure 14. SAR Processing Flow.

## Data Distribution for Parallel SAR Processing on Mercury

**Using 6 PPC CNs for Processing and 2 SHARC CNs for I/O**

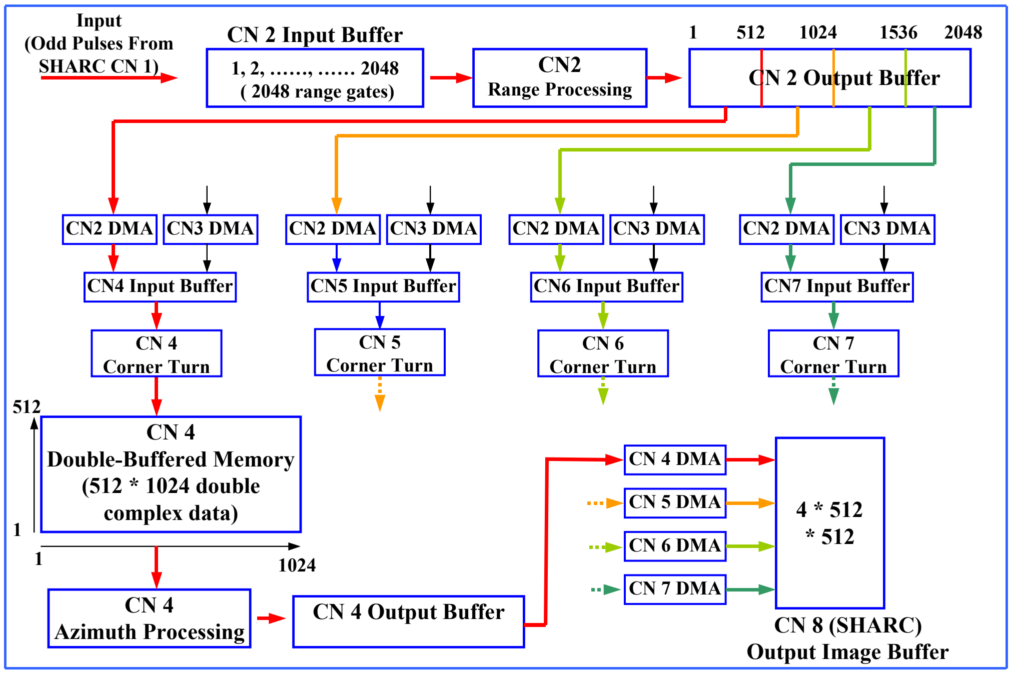


Figure 15. Data distribution for Parallel SAR Processing on Mercury.

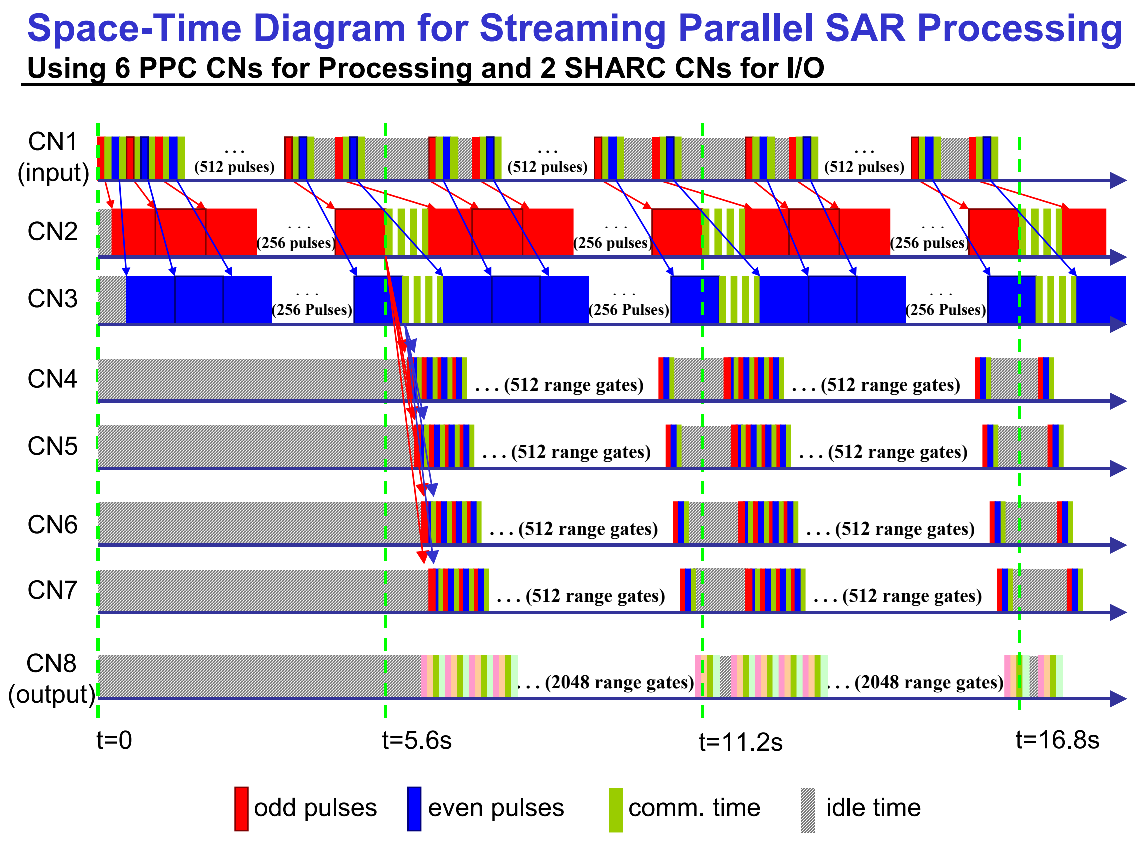
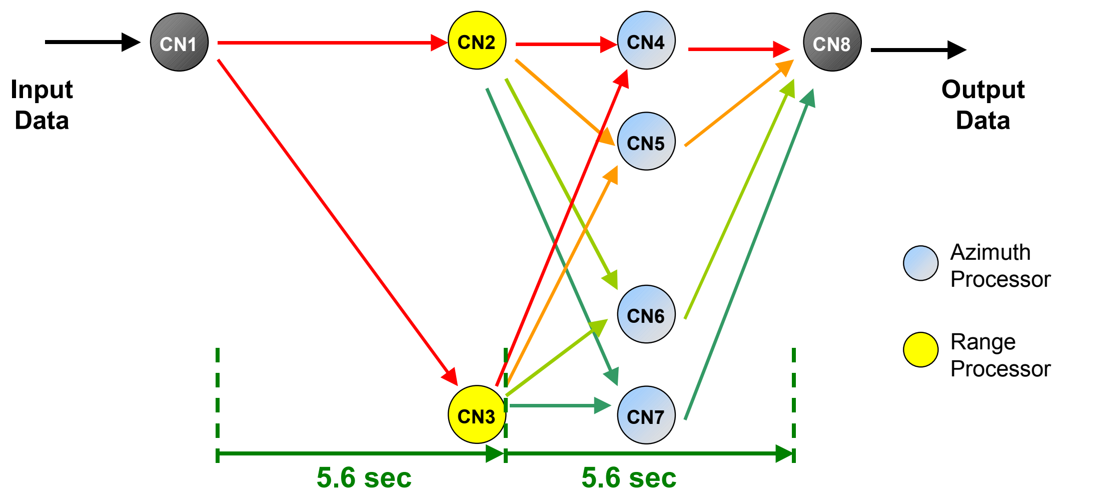


Figure 16. Space-time diagram for streaming parallel SAR processing.

## Streaming Parallel SAR Processing Throughput Requirements

**Using 6 PPC CNs for Processing and 2 SHARC CNs for I/O**



| | | | |
|---|---|---|---|
| Input Data Size | = 512 × 2 × 2032 × 2<br>= 4 MBytes | Output Data Size | = 512 × 2048 × 2 × 4<br>= 8 Mbytes |
| **Input Throughput** | **= 4 MBytes/5.6 sec**<br>**= <u>0.71 MBytes/sec</u>** | **Out Throughput** | **= 8 MBytes/5.6 sec**<br>**= <u>1.42 MBytes/sec</u>** |

Figure 17. Throughput requirements achieved for streaming parallel SAR processing.

## Streaming Parallel RT_STAP on Mercury Subsystem

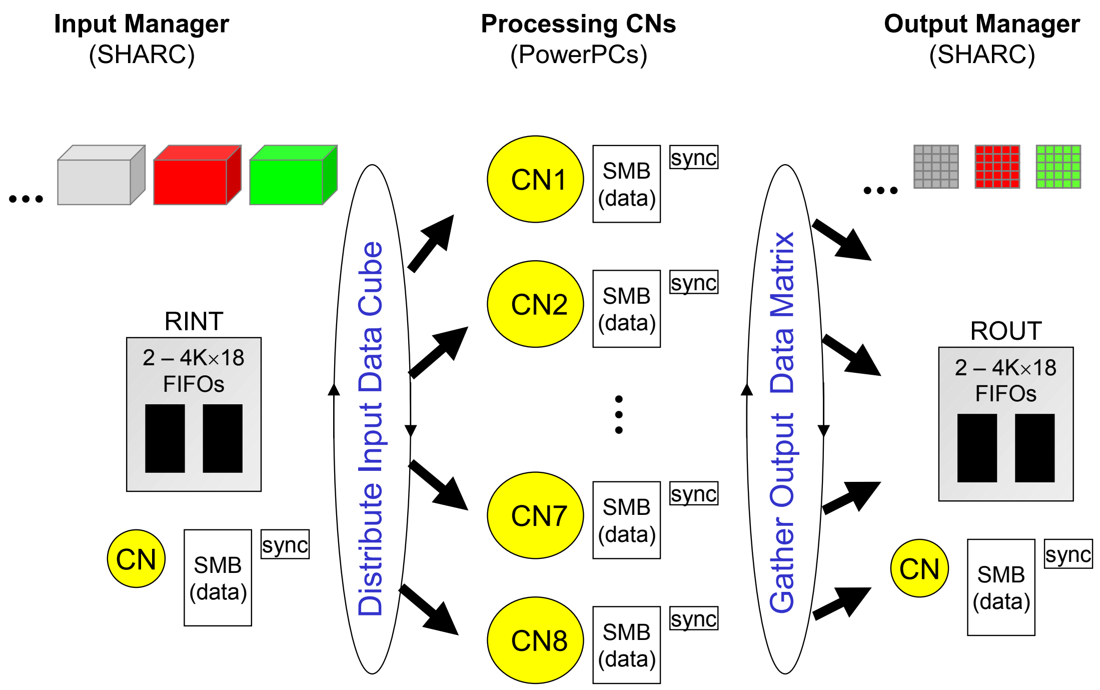


Figure 18. Streaming parallel RT_STAP on Mercury Subsystem.

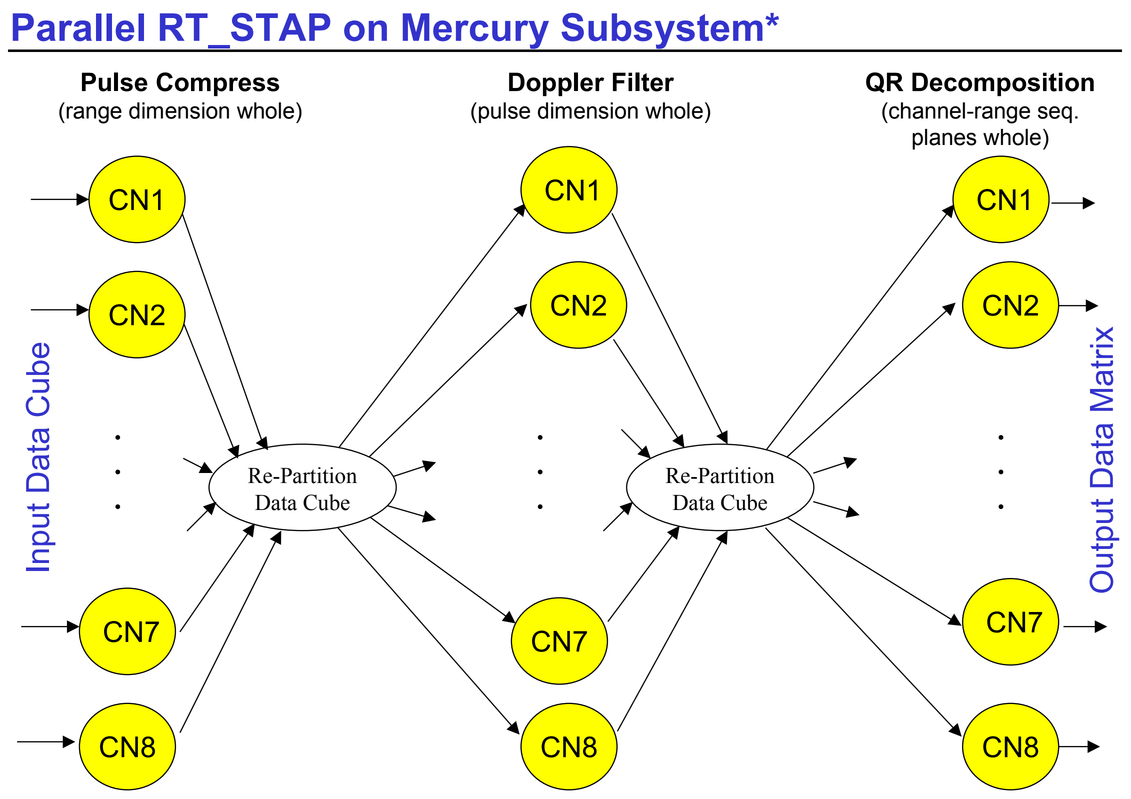


*Figure Derived from:M. Skalabrin and T. Einstein, "STAP Processing on Multiprocessor Systems: Distribution of 3-D Data Sets and Processor Allocation for Efficient Interprocessor Communication," ASAP Workshop, Mar. 1996.

Figure 19. Parallel RT_STAP on Mercury Subsystem.

## Space-Time Diagram for Parallel RT_STAP

**Using 8 PPC CNs for Processing and 2 SHARC CNs for I/O**

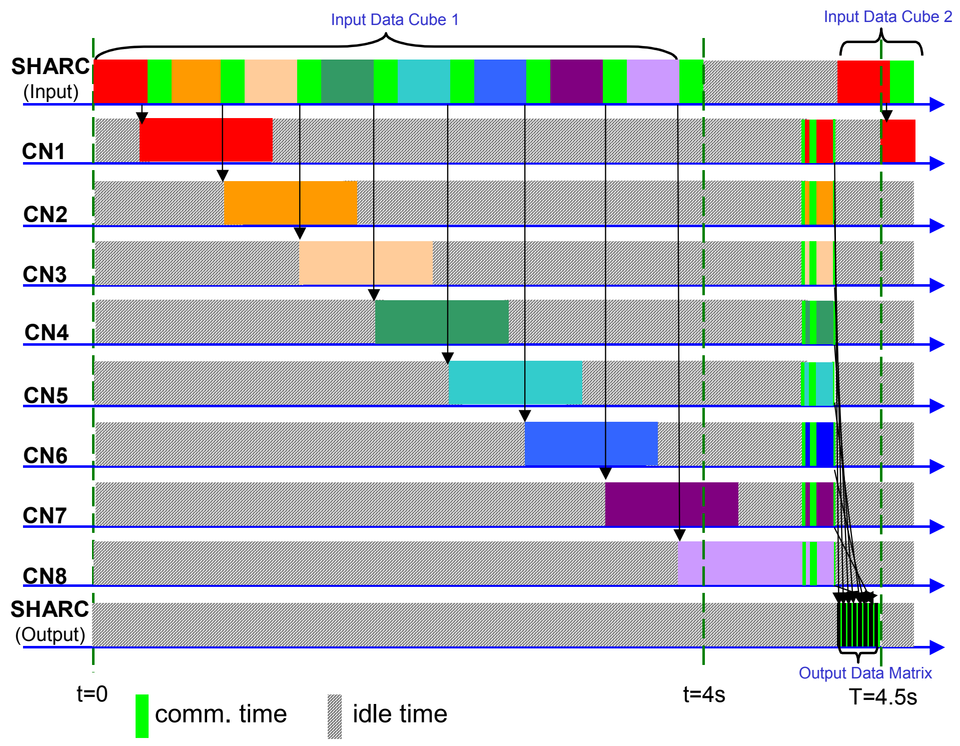
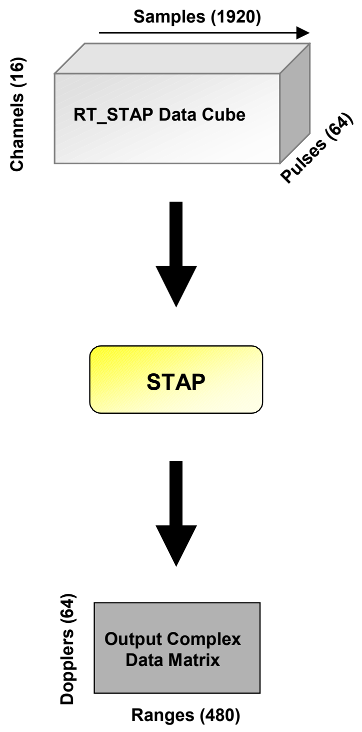


Figure 20. Space-time diagram for parallel RT_STAP.

## Throughput Requirements for Medium Case Parallel RT_STAP

**Using 8 PPC CNs for Processing and 2 SHARC CNs for I/O**



| Function | Time |
|---|---|
| Distribute Input Data | 4 sec |
| Pulse Compress | 299.48 msec |
| First Rotation | 21.18 msec |
| Doppler Filter | 25.32 msec |
| Second Rotation | 112.48 msec |
| QR Decomposition | 99.36 msec |
| Gather Output Data | 23 msec |
| **Total Time** | **4.5 sec** |

Input Data Size = 16 × 64 × 1920 × 2 = 4 MBytes
Output Data Size = 64 × 480 × 8 = 0.25 MBytes

**Input Throughput** **= 4 Mbytes/4.5 sec**
**= 0.89 Mbytes/sec**

**Output Throughput** **= 0.25 Mbytes/4.5 sec**
**= 0.056 Mbytes/sec**

Figure 21. Throughput requirements achieved for the medium case parallel RT_STAP.

## Conclusion

### *Technology Transfer*

Technology transfer took place along five main paths: (1) the DARPA Adapted Computing Systems (ACS) community through PI (Principal Investigator) meetings and other conferences (plus communications with PIs and program managers in related areas); (2) the employees and technical support contacts at Mercury Computer Systems, Inc.; (3) the employees and technical support contacts at Annapolis Micro Systems, Inc.; (4) contacts with various defense contractors such as Northrop Grumman; and (5) the academic high-performance embedded computing research community.

Regarding path (1), we worked with DARPA and other PIs associated with related projects to ensure efficient transfer of information and technology. We attended all PI meetings and helped support DARPA in presenting the results of this effort for further program funding.

For paths (2) and (3), we consulted with the vendors on a regular basis, especially during the period of time in which the prototype system was being constructed. We kept both vendors informed on the current status of the prototype throughout the project. The success of our project sparked interaction between the two vendors in terms of defining and refining interface standards for interconnecting their products. These new standards, which were not available at the time we were constructing our prototype, make it much easier to construct an FPGA/DSP/GPP system such as the one implemented for this project.

The transfer along path (4) was important because it enabled our proposed approaches to be considered and evaluated by defense systems designers and end-users. Also, staying in close contact with major defense contractors and other contractors that were part of the ACS program, ensured that the approaches and systems we developed were realistic.

As indicated by path (5), it was important to keep the academic research community informed about our developments. The publications that resulted from this project have made an impact and serve to illustrate the types of research of interest to DARPA. It also illustrated that there is an abundance of basic, fundamental research to be done on the way to solving important problems of military interest.

### *Deliverables*

This project delivered an abundance of results of both practical and theoretical importance. Many of these results have been published as journal and conference papers, and copies of these papers are provided in the appendices of this report. Online links to delivered publications, presentation materials, dissertations, theses, and additional materials are provided in the References and Additional Materials sections of the report. Associated with each publication is one or more tool or technique of immediate practical importance to practitioners in the area of embedded high-performance systems design and implementation. Also delivered was a prototype platform in which the three technologies of interest (FPGA, DSP, and GPP) were integrated into a single high-performance computational engine. This platform served as a test bed in which experimental tests, evaluations, and assessments associated with the research were conducted.

The theme of the project was to focus on techniques and systems for minimizing power consumption requirements for two particular radar-processing applications. In addition to providing results along these lines, many of the techniques and results delivered are applicable to a much broader set of problems that arise in high-performance, SWAP-constrained embedded systems.

## References

[1A] Jeffrey T. Muehring and John K. Antonio, "Optimal Configuration of an Embedded Parallel System for Synthetic Aperture Radar Processing," *Proceedings of the International Conference on Signal Processing Applications & Technology*, Boston, MA, Oct. 1996, pp. 1489-1494.
Location: **Appendix A** and http://www.cs.ou.edu/~antonio/pubs/conf033.pdf

[2B] Jeffrey T. Muehring and John K. Antonio, "Optimal Configuration of Compute Nodes for Synthetic Aperture Radar Processing," *Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC '98)*, in *Lecture Notes in Computer Science 1388: Parallel and Distributed Processing,* edited by Jose Rolim, sponsor: IEEE Computer Society, Orlando, FL, USA, Apr. 1998, pp. 987-993.
Location: **Appendix B** and http://www.cs.ou.edu/~antonio/pubs/conf035.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf035.pdf

[3] Jeffrey T. Muehring, *Optimal Configuration of a Parallel Embedded System for Synthetic Aperture Radar Processing*, Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock, TX, Dec. 1997.
Location: http://www.cs.ou.edu/~antonio/pubs/muehring_thesis.pdf

[4C] Jack M. West and John K. Antonio, "Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System," *Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC '98)*, in *Lecture Notes in Computer Science 1388: Parallel and Distributed Processing,* edited by Jose Rolim, sponsor: IEEE Computer Society, Orlando, FL, USA, Apr. 1998, pp. 979-986.
Location: **Appendix C** and http://www.cs.ou.edu/~antonio/pubs/conf036.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf036.pdf

[5D] Jack M. West and John K. Antonio, "A Genetic Algorithm Approach to Scheduling Communications for a Class of Parallel Space-Time Adaptive Processing Algorithms," *Proceedings of the 5th International Workshop on Embedded/Distributed HPC Systems and Applications (EHPC 2000)*, in *Lecture Notes in Computer Science*, *IPDPS 2000 Workshops*, sponsor: IEEE Computer Society, Cancun, Mexico, May 2000, pp. 855-861.
Location: **Appendix D** and http://www.cs.ou.edu/~antonio/pubs/conf042.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf042.pdf

[6E] Jack M. West and John K. Antonio, "A Genetic-Algorithm Approach to Scheduling Communications for Embedded Parallel Space-Time Adaptive Processing Algorithms," *Journal of Parallel and Distributed Computing*, Vol. 62, No. 9, Sept. 2002, pp. 1386-1406.
Location: **Appendix E** and http://www.cs.ou.edu/~antonio/pubs/jour016.pdf

[7] Jack M. West, *Simulation of Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System*, Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock, TX, Aug. 1998.
Location: http://www.cs.ou.edu/~antonio/pubs/west_thesis.pdf

[8] Jack M. West, *Processor Allocation, Message Scheduling, and Algorithm Selection for Space-Time Adaptive Processing*, Doctoral Dissertation, Department of Computer Science, Texas Tech University, Lubbock, TX, Aug. 2000.
Location: http://www.cs.ou.edu/~antonio/pubs/west_diss.pdf

[9F] Timothy Osmulski, Jeffrey T. Muehring, Brian Veale, Jack M. West, Hongping Li, Sirirut Vanichayobon, Seok-Hyun Ko, John K. Antonio, and Sudarshan K. Dhall, "A Probabilistic Power Prediction Tool for the Xilinx 4000-Series FPGA," *Proceedings of the 5th International Workshop on Embedded/Distributed HPC Systems and Applications (EHPC 2000)*, in *Lecture Notes in Computer Science*, *IPDPS 2000 Workshops,* sponsor: IEEE Computer Society, Cancun, Mexico, May 2000, pp. 776-783.
Location **Appendix F** and http://www.cs.ou.edu/~antonio/pubs/conf041.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf041.pdf

[10] Timothy A. Osmulski, *Implementation and Evaluation of a Power Prediction Model for a Field Programmable Gate Array*, Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock, TX, May 1998.
Location: http://www.cs.ou.edu/~antonio/pubs/osmulski_thesis.pdf

[11G] Hongping Li, John K. Antonio, and Sudarshan K. Dhall, "Fast and Precise Power Prediction for Combinational Circuits," *Proceedings of the IEEE Symposium on VLSI*, sponsor: IEEE, Tampa, FL, Feb 2003, pp. 254-259.
Location **Appendix G** and http://www.cs.ou.edu/~antonio/pubs/conf046.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf046.pdf

[12H] Nikhil D. Gupta, John K. Antonio, and Jack M. West, "Reconfigurable Computing for Space-Time Adaptive Processing" *Proceedings of the Sixth Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 1998, pp. 335-336.
Location: **Appendix H** and http://www.cs.ou.edu/~antonio/pubs/conf037.pdf

[13] Nikhil D. Gupta, *Reconfigurable Computing for Space-Time Adaptive Processing*, Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock, TX, August 1998.
Location: http://www.cs.ou.edu/~antonio/pubs/gupta_thesis.pdf

[14] Brian F. Veale, *Study of Power Consumption For High-Performance Reconfigurable Computing Architectures*, Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock, TX, August 1999.
Location: http://www.cs.ou.edu/~antonio/pubs/veale_thesis.pdf

[15I] Jeffrey T. Muehring and John K. Antonio, "Minimizing Power Consumption using Signal Activity Transformations for Very Deep FPGA Pipelines," *Proceedings of the Military and Aerospace Applications for Programmable Devices and Technologies Conference (MAPLD 2000),* sponsors: NASA and Johns Hopkins University/Applied Physics Laboratory, Laurel, MD, Sep. 2000.
Location **Appendix I** and http://www.cs.ou.edu/~antonio/pubs/conf044.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf044.pdf

[16J] S. Vanichayobon, Sudarshan K. Dhall, S. Lakshmivarahan, and John K. Antonio, "Power-speed Trade-off in Parallel Prefix Circuits," *Proceedings of ITComm 2002, High-Performance Pervasive Computing Conference*, sponsor: SPIE, Boston, MA, July/Aug. 2002, pp. 109-120.
Location **Appendix J** and http://www.cs.ou.edu/~antonio/pubs/conf045.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf045.pdf

[17] Sirirut Vanichayobon, *Power-Speed Trade-Off in Parallel Prefix Circuits*, Doctoral Dissertation, School of Computer Science, University of Oklahoma, Norman, OK, 2002.
Location: http://www.cs.ou.edu/~antonio/pubs/sirirut_diss.pdf

[18K] Jack M. West, Hongping Li, Sirirut Vanichayobon, Jeffrey T. Muehring, John K. Antonio, and Sudarshan K. Dhall, "A Hybrid FPGA/DSP/GPP Prototype Architecture for SAR and STAP," *Proceedings of the Fourth Annual High Performance Embedded Computing Workshop*, sponsors: U.S. Navy and Defense Advanced Research Projects Agency (DARPA), MIT Lincoln Laboratory Publications, Group 18, Lexington, MA, Sep. 2000, pp. 29-30.
Location: **Appendix K** and http://www.cs.ou.edu/~antonio/pubs/conf043.pdf
Presentation Materials: http://www.cs.ou.edu/~antonio/pubs/p-conf043.pdf

**Additional Materials**

*Annual Reviews and Kickoff Presented to DARPA*

Fall 1999 Annual Review: http://www.cs.ou.edu/~antonio/pubs/p-ann_rev99acs.pdf
Fall 1998 Annual Review: http://www.cs.ou.edu/~antonio/pubs/p-ann_rev98acs.pdf
Fall 1997 Kickoff: http://www.cs.ou.edu/~antonio/pubs/p-kickoff97acs.pdf

*PI Meeting Presentations and Posters*

Presentation, Spring 2000 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/p-sp00acs.pdf
Poster, Spring 2000 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/poster-sp00acs.ppt

Presentation, Fall 1999 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/p-fall99acs.pdf
Poster, Fall 1999 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/poster-fall99acs.ppt

Poster, Spring 1999 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/poster-sp99acs.ppt

Poster, Fall 1998 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/poster-fall98acs.ppt

Poster 1, Spring 1998 PI Meeting:
http://www.cs.ou.edu/~antonio/pubs/poster1-sp98acs.ppt
Poster 2, Spring 1998 PI Meeting:
http://www.cs.ou.edu/~antonio/pubs/poster2-sp98acs.ppt

Presentation, Fall 1997 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/p-fall97acs.pdf
Poster, Fall 1997 PI Meeting: http://www.cs.ou.edu/~antonio/pubs/poster-fall97acs.pdf

*Technical Report*

Hongping Li, John K. Antonio, and Sudarshan K. Dhall, "Fast and Precise Power Prediction for Combinational Circuits," University of Oklahoma, School of Computer Science, Technical Report No. CS-TR-02-001, Nov. 2002, 42 pages. http://www.cs.ou.edu/~antonio/pubs/tr013.pdf (expanded content of [11G]).

## List of Acronyms

| | |
|---|---|
| ACS | Adaptive Computing Systems |
| ASIC | application specific integrated circuit |
| CE | compute element |
| CLB | configurable logic block |
| CN | compute node |
| COTS | commercial off the shelf |
| DARPA | Defense Advanced Research Projects Agency |
| DSP | digital signal processor |
| FLOP | floating-point operation |
| FPGA | field programmable gate array |
| GA | genetic algorithm |
| GPP | general-purpose processor |
| IEEE | Institute of Electrical and Electronics Engineers |
| MIT | Massachusetts Institute of Technology |
| PC | personal computer |
| PCI | peripheral component interconnection |
| PI | principal investigator |
| RASSP | rapid prototyping of application specific signal processors |
| RT_STAP | real-time space-time adaptive processing |
| SAR | synthetic aperture radar |
| SHARC® | "super" Harvard architecture |
| STAP | space-time adaptive processing |
| SWAP | size, weight, and power |
| UAV | unmanned aerial vehicle |

**Appendix A:** Jeffrey T. Muehring and John K. Antonio, "Optimal Configuration of an Embedded Parallel System for Synthetic Aperture Radar Processing," *Proceedings of the International Conference on Signal Processing Applications & Technology*, Boston, MA, Oct. 1996, pp. 1489-1494.

# Optimal Configuration of an Embedded Parallel System for Synthetic Aperture Radar Processing *

Jeffrey T. Muehring and John K. Antonio
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409-3104
{jmuehrin, antonio}@cs.ttu.edu

*Abstract–The creation of a synthetic aperture radar (SAR) image involves processing radar return signals in real-time using a computing platform on board the aircraft that houses the SAR system. In such environments, it is important to minimize the total power consumption of all onboard systems. This is especially true for applications that utilize small unmanned aircraft or satellites. In this paper, a mathematical optimization technique is formulated – based on nonlinear programming – for determining the optimal (i.e., minimal consumed power) configuration of an onboard parallel computing platform for SAR processing. The target hardware for this study is a Mercury Race System that is assumed to be configurable using a combination of two types of daughtercards: one type has six processors and a total of 32MB of memory; the other type has two processors and a total of 64MB of memory.*

## 1 Introduction

Because radar is a ranging instrument, the resolution associated with a single radar return depends on the width of the transmitted pulse; the shorter the pulse, the higher the resolution. However, generating short radar pulses requires high power [2]. In many applications where very high-resolution radar images are desired, there are hard constraints on the allowable size, weight, and power of the radar system (e.g., satellites and unmanned aircraft). Thus, radar systems that can generate extremely narrow pulses are not feasible in such applications because of their associated large size and/or high power requirements.

Synthetic aperture radar (SAR) is a processing technique for achieving high-resolution images from relatively small and low-power radar systems. Specifically, SAR involves the processing of multiple low-resolution radar returns to emulate a high-resolution return. Typical applications for SAR include ground surveillance and terrain mapping. Advantages of using SAR instead of optical imaging techniques include radar's immunity to weather and lighting conditions. Image resolutions for typical SAR applications can range from 50 m down to 0.5 m [3]. Due to space limitations, detailed background information on the theoretical foundations of SAR processing is not included here; however, there are numerous excellent books on the topic (e.g., see [2]).

In addition to the size and power associated with the radar equipment itself, the size and power of the computing platform used to perform the SAR processing can also become significant. Minimizing the power of the computing platform used for SAR processing, for a given radar system, is the focus of this paper.

SAR processing can be parallelized and performed on an embedded parallel computing platform. As a first step toward deciding how to configure such a computing platform, the aggregate required processing throughput associated with a given set of system parameters can be derived (see [3] for details). However, the throughput requirement alone does not uniquely specify how to configure the embedded computer. As described in more detail in Section 2, the computational strategy assumed here involves using the technique of sectioned fast convolution [5]. The choice of the "section size" used in this technique dictates the relative efficiency of the processors used and the amount of memory required. In general, a large section size implies better computational efficiency at the expense of requiring more memory. To further complicate the issue, there are practical constraints on how the embedded computer can be configured. Specifically, the number of processors and amount of memory for a configuration must be realizable by using combinations of different types of daughtercards. In this paper, two types of daughtercards are assumed to be available: one that has six processors and a total

*This work was supported by Rome Laboratory under grant number F30602-96-1-0098.

of 32MB of memory and one that has two processors and a total of 64MB of memory.

The proposed formulation involves the derivation of a parameterized objective function that defines the power consumption of the embedded computer. This objective function depends on radar-dependent parameters, application-dependent parameters, processor-dependent parameters, a software-dependent parameter, and configuration-dependent parameters (i.e., the number of daughtercards of each type). For a fixed set of radar-, application-, and processor-dependent parameters, values of the software- and configuration-dependent parameters are determined that minimize the derived objective function (i.e., the consumed power of the embedded computer).

The rest of the paper is organized in the following manner. In Section 2, an overview of the basic computational strategy is provided and mathematical relationships among the underlying parameters are derived. Based on these mathematical relationships, the proposed optimization problem is formulated in Section 3. A solution technique for the proposed optimization problem and numerical studies are included in Section 4 to illustrate the utility of the proposed approach.

## 2 Computational Framework

The basic computational framework assumed here is the same as that described in [3]. The description given here is an overview; for more details refer to [3].

Processors are divided into range and azimuth processors. That is, every processor is dedicated exclusively to the processing of data either in the range or azimuth direction. The range direction is perpendicular to the line of flight and the azimuth direction is parallel to the line of flight.

After radar returns have been sampled and converted to digital signals, samples are typically read into memory at a rate of 5-50 Msamples/s [3]. By visualizing memory as a 2-dimensional grid, a row of memory contains the returns from a single radar pulse, whereas a column contains returns of different pulses from the same range. Memory is therefore sequentially filled a row at a time. When a sufficient number of rows have been filled, this data is sent to a range processor. These blocks of data are sent to the range processors in a round-robin fashion. After a number of range processors have processed data, the conglomerate block of data is "corner-turned," or matrix-transposed, and then sent to the azimuth processors. Note that the number of range and azimuth processors need not be the same. The matrix transposition of the data dictates that the azimuth processors receive the range-processed rows as columns and the unprocessed columns of the azimuth direction as rows.

Processing of the samples in the range direction primarily involves convolving the data with a reference kernel. The most efficient method of performing this convolution is with the use of FFTs, which is known as a fast convolution [5, 7]. It is assumed that the entire vector of range samples for a given pulse return is processed as a single section of data.

The azimuth processors perform similar operations on the data as the range processors (i.e., fast convolution) but with one important difference: the length of the data stream in the azimuth direction is indefinite whereas in the range direction it is of a fixed length. Therefore the data cannot be convolved as a single entity in the azimuth dimension. Sectioned fast convolution [5] provides a method for processing data streams of indefinite length. For such a data stream, the data is divided into sections of arbitrary length. A section is then convolved with the prestored kernel as in the case of a regular fast convolution. However, overlapping the sections by an amount equal to the kernel size and performing fast convolutions on each overlapped section yields the same result as if the entire data stream were convolved at once. But there is a price to be paid in computational efficiency for using this method. A portion (of length equal to the kernel size) of each convolution resultant must be discarded. Therefore, computational efficiency decreases as the ratio of the section of new data to the kernel size decreases.

Besides memory, another limiting factor to the size of the new data to be convolved is the $O(N \lg N)$ time complexity of the standard FFT algorithm. An important objective is to balance computational efficiency with memory requirements. For instance, selecting a section size that maximizes computational efficiency alone, without regard for concomitant memory requirements, may be unfavorable due to high power consumption of the required memory. Accounting for this tradeoff is an important aspect of the model presented in this section.

A fast convolution consists of an $N$-point FFT, $N$ complex multiply operations, and an $N$-point inverse-FFT, where $N$ is the number of data points to be processed, including any overlap. The complexity of this computational load is therefore $L = O(N \lg N + N)$. The exact number of floating point operations generally depends on processor- and implementation-specific details. For the purposes of this paper, SHARC processors are assumed, for which the exact number of floating point operations is given by [3]:

$$L = 10N \lg N + 6N.$$

The computational load per sample is obtained by dividing $L$ by the number of new data points processed, which reflects the efficiency of the calculation. For range processing this load per sample, $\phi_r$, due to the fast convolution is given by

$$\phi_r = \frac{10F_r \lg F_r + 6F_r}{S_r},$$

where $F_r$ is the FFT size for the range and $S_r$ is the number of points in the range to be processed. These two values can differ because of the stipulation in the FFT algorithm that requires the FFT size to be a power of two (i.e., $F_r = 2^k$). Although this implies some inefficiency, it is usually still faster than using a direct convolution algorithm based on the exact sequence length.

The number of range points $S_r$ is equal to the range swath $R_s$ divided by the desired resolution $\delta$ (this is an intuitive result based on the physical interpretations of $R_s$ and $\delta$). Using this expression, the equation for $\phi_r$ becomes

$$\phi_r = \frac{\delta F_r(6 + 10 \lg F_r)}{R_s}.$$

Similarly, the azimuth processing load per sample due to the fast convolution is given by

$$\phi_a = \frac{F_a(6 + 10 \lg F_a)}{S_a},$$

where $F_a$ is the azimuth FFT size and $S_a$ is the section length. It should be noted that for both range and azimuth processing, the reference kernels are prestored and dependent only upon physical parameters of the system.

To compute the number of processors required for both range and azimuth processing, the total computational load must be computed. The fast convolution comprises the majority of the load. However, several other operations are also involved, including fix-to-float conversion, complex signal formation, motion compensation, magnituding, and the matrix transpose already mentioned [3]. It is important to realize that different operations can take different amounts of time, even if they are considered to be a "single floating point operation." Therefore, calculating the total computational load requirement per data sample involves dividing the number of real operations per sample of each type by their respective tested throughputs for a given type of processor. This value multiplied by the sample rate yields the total number of processors required.

Range and azimuth processing have unique load requirements in addition to the fast convolution load and are noted by the constants $\alpha_r$ and $\alpha_a$, respectively. The required number of range processors is then defined by

$$P_r = Q(\alpha_r + \frac{\phi_r}{\gamma}), \qquad (1)$$

where $Q$ is the sample rate and $\gamma$ is the throughput in Mflops for a fast convolution based on the assumed processor type used. Similarly, the number of azimuth processors required is given by

$$P_a = Q(\alpha_a + \frac{\phi_a}{\gamma}). \qquad (2)$$

It can be shown that the sample rate is determined by the following equation [3]:

$$Q = \frac{vR_s}{\delta^2},$$

where $v$ is the velocity of the platform. If this expression is substituted for $Q$ and the expressions for $\phi_r$ and $\phi_a$ are also applied, then Eqs. (1) and (2) become

$$P_r = \frac{v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^2} \qquad (3)$$

$$P_a = \frac{vR_s(\alpha_a + \frac{F_a(6+10\lg F_a)}{\gamma S_a})}{\delta^2}. \qquad (4)$$

The total memory required for range processing is a product of the number of range processors, $P_r$, and the number of range samples, $S_r$. This value represents the number of complex range samples that are stored in memory at a given instant, each complex sample consisting of 16 bytes. Therefore the total range memory required is

$$M_r = 16P_rS_r, \qquad (5)$$

or equivalently,

$$M_r = \frac{16R_s v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^3}. \qquad (6)$$

Azimuth memory needs dominate total system memory, requiring a double-buffer (for the matrix transpose operation) and an output image buffer, both of size $S_r(S_a + K_a)$, where $K_a$ denotes the length of the azimuth reference kernel. The double-buffer must store complex values; the output image buffer stores reals. The total azimuth memory requirement in bytes is expressed as

$$M_a = 10S_r(S_a + K_a). \qquad (7)$$

The value of $K_a$ can be expressed in terms of basic parameters of the radar. Let $\lambda$ be the wavelength of the radar. The value for $K_a$ is derived in [3] to be:

$$K_a = \frac{\lambda R}{2\delta^2}.$$

Substituting this expression and $S_r = R_s/\delta$ into Eq. (7) yields

$$M_a = \frac{R_s(\lambda R + 2\delta^2 S_a)}{\delta^3}. \quad (8)$$

## 3 Formulation of an Optimal Configuration Problem

The final equations derived above for $P_r$, $P_a$, $M_r$, and $M_a$, given by Eqs. (3), (4), (6), and (8), depend on many different types of basic system parameters. These basic parameters can be divided into four major categories:

- radar-dependent parameters: $R$ (range), $R_s$ (range swath), and $\lambda$ (wavelength);
- application-dependent parameters: $\delta$ (desired resolution) and $v$ (platform velocity);
- processor-dependent parameters: $\alpha_r$, $\alpha_a$, and $\gamma$; and
- software-dependent parameter: $S_a$.

From Eqs. (3), (4), (6), and (8), it appears that there is also a dependence on the parameters $F_r$ (range FFT size) and $F_a$ (azimuth FFT size). However, recall that $F_r$ and $F_a$ are functions of $S_r$ and $S_a + K_a$, respectively, and $S_r$ and $K_a$ can both be expressed in terms of basic radar- and application-dependent parameters.

For the purposes of this section, denote the total processor requirement $(P_r + P_a)$ and the total memory requirement $(M_r + M_a)$ as $P$ and $M$. To formulate an optimal configuration problem, it is assumed that all radar-, application-, and processor-dependent parameters are specified, and $S_a$ is to be determined. To emphasize this dependence solely on the parameter $S_a$, $P$ and $M$ are denoted by $P(S_a)$ and $M(S_a)$. The question that naturally arises is how to optimally choose the value of $S_a$? More fundamentally, how does the value of $S_a$ affect the resulting configuration of the computing platform and its value of consumed power? Recall that the desired objective is to minimize the total power consumption of the computing platform.

A possible (yet unrealistic) approach would be to model consumed power of the computing platform as

$$\kappa P(S_a) + \beta M(S_a),$$

where $\kappa$ and $\beta$ are constants that represent power requirements on a per processor and per byte of memory basis, respectively. Determining a value of $S_a$, say $S_a^*$, which minimizes this function could be used to define an optimal configuration – i.e., a configuration that has $P(S_a^*)$ processors and $M(S_a^*)$ bytes of memory.

Modeling total consumed power as described above is unrealistic because it allows configurations to have arbitrary numbers of processors and amounts of memory. This would require, in general, that such a configuration be realized at the chip-level, i.e., customized boards may have to be developed to support the derived optimal configurations.

In reality, it is more practical to constrain the set of configurations to those that are realizable using commercially available boards that contain differing numbers of processors and amounts of memory. For this study, the computing platform is assumed to be based on a Mercury Race System that is configurable using a combination of two possible types of daughtercards: (1) the S2T16B, which has a total of six SHARC processors and 32MB of memory and (2) the S1D64B, which has a total of two SHARC processors and 64MB of memory. Each of these card types has a corresponding maximum power consumption rating: the type 1 card is rated at 12.2 watts and the type 2 card is rated at 9.6 watts [6]. Under this framework, the total power consumption is modeled based on the number of cards of each type utilized.

Let $C_1$ and $C_2$ denote the number of type 1 and type 2 cards utilized, respectively. Thus, the function for total consumed power, denoted as $W$, is defined as

$$W = 12.2C_1 + 9.6C_2. \quad (9)$$

Next, two required constraint equations naturally follow based on the values of $P(S_a)$ and $M(S_a)$:

$$6C_1 + 2C_2 \geq P(S_a) \quad (10)$$

$$32C_1 + 64C_2 \geq M(S_a). \quad (11)$$

These constraint equations insure that the total number of processors in the configuration is no less than the total number of required processors and the total amount of memory in the configuration is no less than the total amount of memory required. In this framework, values for the parameters $C_1$ and $C_2$ must be optimized (in addition to the value of the parameter $S_a$). Although the parameter $S_a$ does not explicitly appear in the objective function that is to be minimized, i.e., $W$, its effect is implicit through the constraint equations.

To summarize, the proposed optimization problem is stated as follows: find nonnegative integer values for $C_1$, $C_2$, and $S_a$ such that $W$ is minimized and constraint Eqs. (10) and (11) are satisfied.

## 4 Solving the Optimal Configuration Problem

### 4.1 *Proposed Solution Technique*

As formulated, the proposed optimization problem can be classified as an integer programming problem. Solving such optimization problems can be computationally intensive (see [4] for a summary on integer programming techniques).

Instead of directly applying an integer programming technique, an alternative approach is proposed here for solving the formulated optimization problem. Notice that the objective and the constraint equations are *nearly* continuous functions of the optimization variables $C_1$, $C_2$, and $S_a$. If the objective and constraints were continuous, then nonlinear programming techniques (e.g., see [1]) could be applied. Such approaches often have fast convergence properties. The only discontinuous portion in the formulation is due to the definition of $F_a$, which is a discontinuous function of $S_a$. (Recall that $F_a$ is defined as the smallest integer power of two that is greater than $S_a + K_a$.) This discontinuous function prevents the direct application of nonlinear programming. However, by selecting $F_a$ as an integer power of two, and adding a constraint to ensure that $K_a + S_a$ is no greater than this selected value, the discontinuity can be removed. Thus, in addition to the constraints given by Eqs. (10) and (11), the following constraint equation is added

$$K_a + S_a \leq F_a, \tag{12}$$

where the value of $F_a = 2^k \geq K_a$ is fixed (the value of $K_a$ is known based on the values of the specified basic parameters). Thus, to ensure optimality, it may be necessary to solve several constrained optimizations based on different feasible values for $F_a$. In practice, however, only a few values for $F_a$ need to be tried: from the smallest feasible value up to the point at which the optimal value of $S_a$ is such that $K_a + S_a < F_a$ (i.e., the constraint becomes inactive).

### 4.2 *Numerical Studies*

The solution technique proposed in the previous subsection is applied to find optimal configurations based on four different sets of application-dependent parameters: (1) $\delta = 1, v = 300$; (2) $\delta = 1, v = 200$; (3) $\delta = 1.5$, $v = 300$; and (4) $\delta = 1.5$, $v = 200$ (the units for $\delta$ and $v$ are meters and meters/s, respectively). For all four cases considered, the radar-dependent parameters and processor-dependent parameters were fixed at the following values: $R = 10^5$, $R_s = 2 \times 10^4$, $\lambda = 0.03$, $\alpha_r = 0.3528$, $\alpha_a = 0.9068$, and $\gamma = 94$. These values are derived in [3] based on a Mercury Race System configured using SPARC processors.

Intuitively, case 1 represents the most computationally demanding scenario of the four cases considered – it has the largest platform velocity and the finest desired SAR resolution. Case 4, on the other hand, represents the other end of the spectrum – it is the scenario with the smallest velocity and coarsest resolution. Thus, it would be expected that case 1 have the highest power consumption requirement and case 4 the lowest – this intuition is confirmed in the numerical studies described next.

The formulated optimization problem was solved using a routine from the Optimization Toolbox of MATLAB called `constr`. This routine was executed interactively (in MATLAB's command line mode) on a Sun SparcStation, and the response time for solving each optimization was almost immediate (less that one second).

To illustrate the advantage associated with allowing configurations to have two types of cards (i.e., heterogeneous configurations), optimizations were also conducted in which only one card type is allowed (i.e., homogeneous configurations). Mathematically, finding an optimal homogeneous configuration corresponds to setting the value of either $C_1$ or $C_2$ to zero and solving the resulting optimization. Tables 1, 2, and 3 summarize the results of the numerical studies that were conducted. Table 1 shows the results for the optimal heterogeneous configurations, in which both types of cards are allowed (i.e., solving the optimization as described in the previous subsection). Tables 2 and 3 show the results of optimal homogeneous configurations, in which $C_2$ and $C_1$, respectively, were defined to be zero in the formulation. In all tables, the optimal values of $S_a$, $C_1$, $C_2$, and $F_a$ as well as the corresponding optimal value of the consumed power are tabulated for each of the four cases considered.

Notice that optimal values of $S_a$ and $F_a$ given in Table 2 are substantially less than those in Table 3. This is logical considering that the memory to processor ratio for the type 2 card is much higher than that for the type 1 card, and memory requirements grow linearly with the value of $S_a$ (refer to Eq. (7)). In reality, of course, a fractional number of cards cannot be installed in an actual configuration. Thus, the values for $C_1$ and $C_2$ would need to be rounded up to the nearest integers so that the processor and memory constraints are satisfied.

Table 1: Optimal Heterogeneous Configurations: Type 1 and 2 Cards

| case no. $(\delta : v)$ | $S_a$ | $C_1$ | $C_2$ | $F_a$ | Power (in watts) |
|---|---|---|---|---|---|
| 1 (1 : 300) | 548 | 4.8 | 4.2 | 2048 | 94.7 |
| 2 (1 : 200) | 548 | 2.1 | 5.3 | 2048 | 77.3 |
| 3 (1.5 : 300) | 357 | 1.5 | 1.4 | 1024 | 31.9 |
| 4 (1.5 : 200) | 357 | 0.7 | 1.8 | 1024 | 26.0 |

Table 2: Optimal Homogeneous Configurations: Type 1 Cards Only

| case no. $(\delta : v)$ | $S_a$ | $C_1$ | $C_2$ | $F_a$ | Power (in watts) |
|---|---|---|---|---|---|
| 1 (1 : 300) | 259 | 11.0 | 0 | 2048 | 134.4 |
| 2 (1 : 200) | 175 | 10.5 | 0 | 2048 | 127.9 |
| 3 (1.5 : 300) | 174 | 3.5 | 0 | 1024 | 42.8 |
| 4 (1.5 : 200) | 118 | 3.3 | 0 | 1024 | 39.9 |

Table 3: Optimal Homogeneous Configurations: Type 2 Cards Only

| case no. $(\delta : v)$ | $S_a$ | $C_1$ | $C_2$ | $F_a$ | Power (in watts) |
|---|---|---|---|---|---|
| 1 (1 : 300) | 2154 | 0 | 11.4 | 4096 | 109.7 |
| 2 (1 : 200) | 1559 | 0 | 9.6 | 4096 | 91.8 |
| 3 (1.5 : 300) | 1342 | 0 | 4.2 | 2048 | 40.2 |
| 4 (1.5 : 200) | 975 | 0 | 3.4 | 2048 | 32.9 |

## 5 Conclusions

A formal approach for optimally configuring an embedded computing platform for SAR processing was introduced. The formulation allows the platform to be configured using a combination of two types of cards. The variables that are optimized include the number of cards of each type and an FFT section size parameter. The advantage – in terms of minimizing consumed power – of optimally utilizing two card types (instead of restricting configurations to have only one card type) was illustrated through numerical studies. Also, an intuitive correspondence between optimal power consumption requirement and application-dependent parameters (i.e., SAR image resolution and platform velocity) was illustrated.

## Acknowledgments

The authors thank T. Einstein for his help in explaining some of the finer points of the computational framework assumed in this paper, which was originally described in [3]. We also thank A. G. Antonio for sharing with us his expertise in radar systems and SAR processing.

## References

[1] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley & Sons, New York, NY, 1993.

[2] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing*, John Wiley & Sons, New York, NY, 1991.

[3] T. Einstein, "Realtime Synthetic Aperture Radar Processing on the RACE Multicomputer," Application Note 203.0, Mercury Computing Systems, Inc., Chelmsford, MA, 1995.

[4] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, Sixth Edition, McGraw-Hill, New York, NY, 1995.

[5] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[6] "SHARC DSP Compute Nodes (3.3-Volt)," Mercury Computing Systems, Inc., Chelmsford, MA, Sept. 1995.

[7] J. S. Walker, *Fast Fourier Transforms*, Second Edition, CRC Press, New York, NY, 1996.

**Appendix B:** Jeffrey T. Muehring and John K. Antonio, "Optimal Configuration of Compute Nodes for Synthetic Aperture Radar Processing," *Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC '98)*, in *Lecture Notes in Computer Science 1388: Parallel and Distributed Processing,* edited by Jose Rolim, sponsor: IEEE Computer Society, Orlando, FL, USA, Apr. 1998, pp. 987-993.

# Optimal Configuration of Compute Nodes for Synthetic Aperture Radar Processing

Jeffrey T. Muehring and John K. Antonio

Deptartment of Computer Science, P.O. Box 43104, Texas Tech University, Lubbock, TX 79409-3104
{muehring, antonio}@ttu.edu

**Abstract.** Embedded systems often must adhere to strict size, weight, and power (SWAP) constraints and yet provide tremendous computational throughput. Increasing the difficulty of this challenge, there is a trend to utilize commercial-off-the-shelf (COTS) components in the design of such systems to reduce both total cost and time to market. Employment of COTS components also promotes standardization and permits a more generalized approach to system evaluation and design than do systems designed at the application-specific-integrated-circuit (ASIC) level. The computationally intensive application of synthetic aperture radar (SAR) is by nature a high-performance embedded application that lends itself to parallelization. A system performance model, in the context of SWAP, is developed based on mathematical programming. This work proposes an optimization technique using a combination of constrained nonlinear and integer programming.

## 1 Introduction

This work focuses on modeling and optimizing the processor-memory relationships of an embedded system for synthetic aperture radar (SAR) processing. The hardware computing platform of investigation is one constructed with commercial off-the-shelf (COTS) components that are based on daughtercards and the compute-node concept. A daughtercard consists of one or more compute nodes, where a compute node is defined as an entity consisting of one or more processors, a block of shared memory, and the requisite glue logic. Within the framework of the models developed, optimization is performed on parameters such as the convolution section size and the choice and number of daughtercards comprising the system.

Size, weight, and power (SWAP) constraints often motivate the maximization of performance density for a given SAR system, especially in the case of unmanned aerial vehicles (UAVs) or satellites, which often accommodate SAR systems. SAR in itself is an approach to densifying a radar system by substituting a large degree of data postprocessing for radar equipment with prohibitively high size, weight, and power characteristics. Minimizing the power consumption of the compute platform used for

SAR processing is the fundamental objective in this research (although with sufficient parameter guidelines, size and weight could also be minimized using the same approach).

## 2 Fundamentals of SAR Processing

The specific mode of SAR investigated in this research is known as *stripmapping*. In stripmapping, successive radar pulses are transmitted and returned in the range dimension, which is orthogonal to the line of flight. Each received series of pulses from an individual transmitted pulse is then convolved with a reference kernel to achieve range compression. The entire range dimension is processed at once in this way. Detailed coverage of SAR and SAR processing is available in such works as [1, 2].

To create a two-dimensional SAR image, processing in the azimuth dimension is also necessary. The azimuth dimension is parallel to the line of flight and is conceptually infinite in length. Thus, processing of the entire azimuth vector, created from stacked range-processed vectors, is infeasible. To counter this problem, sectioned convolution is employed.

Sectioned convolution extracts a piece (or section) of the azimuth vector, convolves it with a reference kernel as in the range dimension, and then discards a portion of the result equal to the length of the reference kernel. Successively processed azimuth sections are then overlapped (with overlaps equal to the discarded kernel length) to form continuous vectors in the azimuth dimension. As is intuitive, a large azimuth section length requires more memory than a small section. Correspondingly, small azimuth sections require more total processing than do large sections because the percentage of new data processed, which is not discarded, is low (the size of the reference kernel being fixed).

A key point in this work is the exploitation of the section size and the concomitant processor-memory tradeoff [3]. Different daughtercards are better suited for different scenarios depending on the memory per processor ratio associated with the daughtercard, which is largely dependent on the chosen section size. The combination of the choices for the section size and number and types of daughtercards employed greatly affects the overall performance and associated power consumption of the computational platform.

## 3 Optimization Models

Two models are presented in this work, which address the problem of determining the optimal parameter values for configuring the system. Both methods are based on mathematical programming, which provides a method of formulating an optimization problem given an objective and set of constraints [4, 5]. This work proposes optimization techniques using a combination of constrained nonlinear and integer programming.

The first model is based on the assumption of an ideal shared-memory system. It treats all the memory contributed by individual daughtercards as a conglomerate block, equally accessible by all processors located on all daughtercards. For a system that is tightly predicated on the compute node with relatively high penalties for inter-compute-node communication, this is an inaccurate oversimplification. However, it is useful to initially investigate the optimization of the SAR system based on such an assumption because it provides clear insight into the interrelationships between variables and the effects of perturbation of other external parameters. In addition, without constraints on the amount of local memory available to a processor, the ideal memory-per-processor ratio can be derived from the optimization solution.

The second model removes the assumption of global shared memory and purposes to address system configuration more realistically. With this goal comes an increase in the complexity of the optimization formulation. The constraint set is modified to ensure only local memory access by processors. To accomplish this optimization, a much higher degree of integer programming is required than in the first model, entailing greater computational intensity to perform the optimization. The benefits of this second model include solutions that consist of a complete specification of how system resources are to be utilized, whereas the first model only specifies which resources are to be employed.

Parallelization of SAR processing involves the allocation of system resources for either range or azimuth processing [6]. In the first model, range and azimuth processors and memory are treated as aggregate requirements that somehow must be met with an appropriate number of daughtercards of each type. The second model, however, specifies how many processors and how much memory on each compute node per daughtercard is allocated for each function to prevent remote memory access during computation. Note that a single compute node can perform both range and azimuth processing, although each processor within a compute node must be dedicated to a single task.

## 4 Numerical Studies

Test data is based on the availability of two different daughtercards. The first is comprised of two compute nodes. Each compute node on this daughtercard consists of three processors and a shared memory block of 16 MB. The second daughtercard consists of a single compute node with two processors and 64 MB of memory. The first daughtercard consumes 12.2 watts of power and the second 9.6 watts. Throughput data for the significant operations involved in SAR processing is based on SHARC processors [7].

MATLAB's **constr** function in the Optimization Toolbox was used to solve the nonlinear constrained programming problem presented by both models. The nonlinear nature of the problem results from the equations that express the required system memory and number of processors, which are derived in [8]. The **constr** implements a Sequential Quadratic Programming algorithm [9]. Integer programming, the need for which results from the inherently discrete number of processors per compute node and total compute nodes in a system, is implemented by multiple optimizations over the feasible discrete permutations.

Figs. 1 through 3 illustrate the result of solving the optimization problem of one of the models many times across a range of values for different platform velocities and desired resolutions. In each case, the platform velocity ranges from 50-400 m/s and the resolution from 0.5-2.0 m.

The utility of optimization of the section size is demonstrated by comparison of results produced by a heuristic used to determine section size, which defines the section size to be equal to the kernel size. This section size definition and resultant system configuration is designated as *nominal*. This work finds that the nominal section size, although relatively efficient in processing, is too large for most scenarios because of the excessive memory requirements involved. The optimizations performed show that forcing relatively inefficient processing with an associated reduction in memory requirements is optimal if power is to be minimized. Optimal section sizes thus often are found to be only a fraction of the kernel size, entailing the processing of more old data that is to be discarded than new data.

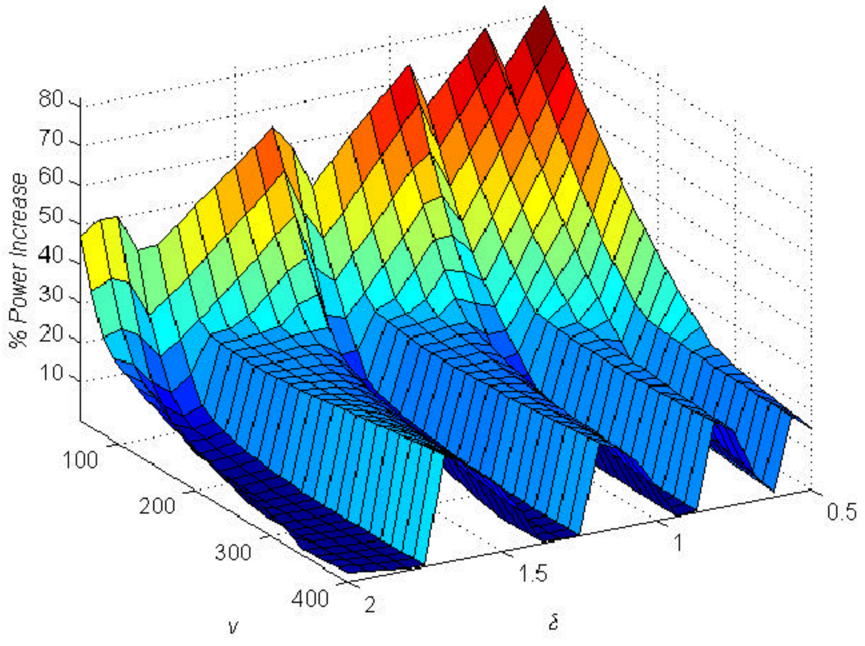


**Fig. 1.** Ratio of power consumption of the nominal section size to the optimal section size.

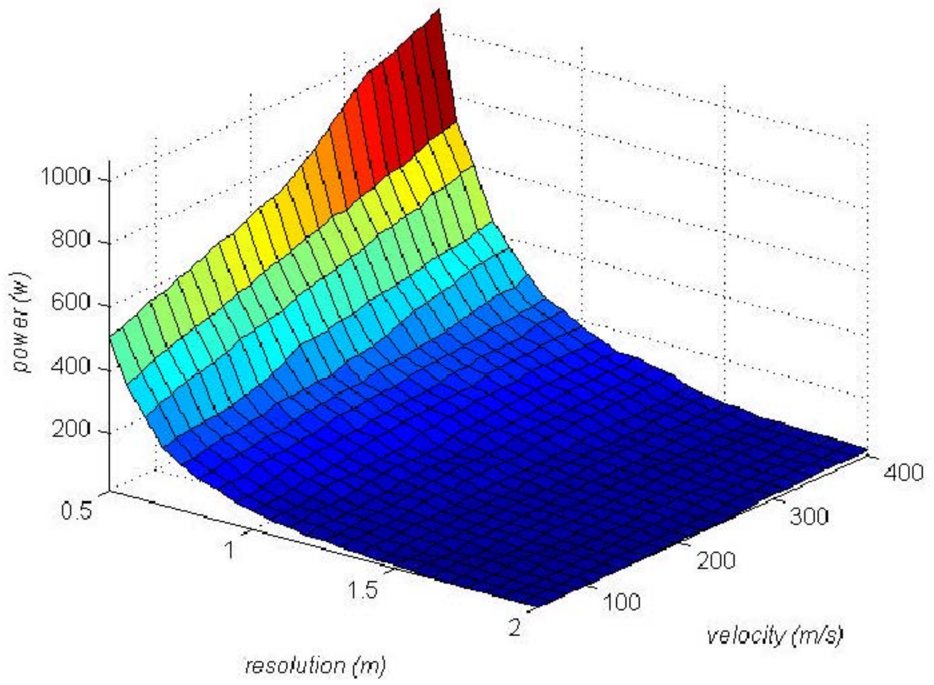


**Fig. 2.** Power consumption of the CN-constrained model.

Figure 1 shows the surface plot of the ratio of results obtained by employment of the nominal section size to the optimized section size of the first model. As would be expected, the optimized section size always results in equal or lower power consumption than does the nominal section size. The optimized section size adjusts to take advantage of unutilized processor and/or memory resources resulting from changes in system requirements produced by changes in the velocity (axis labeled *v*) and/or resolution (axis labeled *d*).

In both models, higher velocities and/or finer resolutions require more daughtercards and thus more power. All other radar parameters such as wavelength, range, range swath, and pulse width remain fixed at values representative of a real system [6]. These trends are illustrated in Fig. 2, which represents the optimal power consumption associated with the second model.

Fig. 3 displays the daughtercard configurations necessary for the optimal power values represented in Fig. 2. A configuration is defined as the processor and memory allocation (for range or azimuth processing) per compute node for a particular daughtercard type. An optimal system configuration consists of one or two daughtercard configurations. The two configurations are denoted as *X* and *Y*, with the subscipts *T*, *r*, and *a* designating the daughtercard type (*T*), number of range processors per compute node of that type daughtercard (*r*), and the number of azimuth processors (*a*).

## 5 Summary and Conclusions

Comparison of the two models shows the first model to be a good approximator to the second model. Both the simplicity of formulation and the speed of data collection lend the first model to be a useful method for obtaining a preliminary estimate for total required system power and number of daughtercards. Refer to [8] for a full comparison of the optimal power consumptions produced by the two models.

This work demonstrates the advantage of employing more than one type of daughtercard in a system. Different daughtercards are characterized by different power requirements and the processor-memory ratio of the compute nodes that they house. Optimization exploits these differences and determines the optimal system configurations.

Generalization of the models developed in this work is straightforward. Although data is collected based on sample daughtercards and compute nodes deemed to be representative of actual systems, the values that characterize the daughtercards are expressed as functions that can be immediately adapted to accommodate any number of additional components.

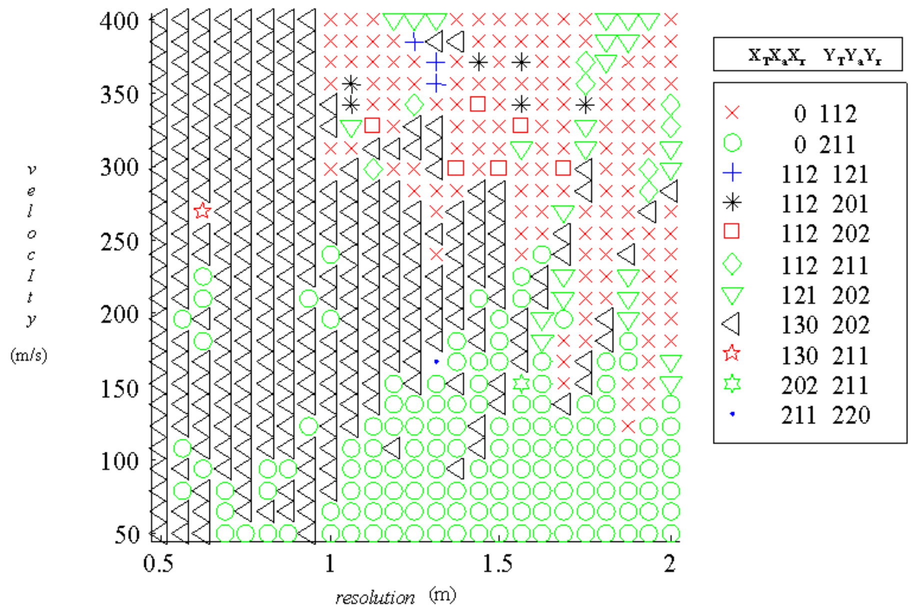


**Fig. 3.** Configurations of the CN-constrained model.

## Acknowledgements

This work was supported by Rome Laboratory under Grant No. F30602-96-1-0098 and Defense Advanced Research Projects Agency (DARPA) under Contract No. F30602-97-2-0297.

## References

1. J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing*, John Wiley & Sons, New York, NY, 1991.

2. W. G. Carrara, R. S. Goodman, and R. M. Majewski, *Spotlight Synthetic Aperture Radar: Signal Processing Algorithms*, Artech House, Boston, MA, 1995.

3. J. T. Muehring and J. K. Antonio, "Optimal Configuraion of Parallel Embedded Systems for Synthetic Aperture Radar," *Proceedings of the 7th International Conference on Signal Processing & Applied Technology*, October 1996, pp. 1189-1194.

4. F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, Sixth Edition, McGraw-Hill, New York, NY, 1995.

5. M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley & Sons, New York, NY, 1993.

6. T. Einstein, "Realtime Synthetic Aperture Radar Processing on the RACE Multicomputer," Application Note 203.0, Mercury Computing Systems, Inc., Chelmsford, MA, 1996.

7. "SHARC DSP Compute Nodes (3.3-Volt)," Mercury Computing Systems, Inc., Chelmsford, MA, Sept. 1995.

8. J. T. Muehring, *Optimal Configuration of a Parallel Embedded System for Synthetic Aperture Radar Processing*, M. S. Thesis, Texas Tech University, 1997 (http://hpcl.cs.ttu.edu/darpa/opt_config/thesis.pdf)

9. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

**Appendix C:** Jack M. West and John K. Antonio, “Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System,” *Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC ‘98)*, in *Lecture Notes in Computer Science 1388: Parallel and Distributed Processing,* edited by Jose Rolim, sponsor: IEEE Computer Society, Orlando, FL, USA, Apr. 1998, pp. 979-986.

# Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System

Jack M. West and John K. Antonio

Department of Computer Science, P.O. Box 43104, Texas Tech University, Lubbock, TX 79409-3104
{west, antonio}@ttu.edu

## Extended Abstract

The focus of this work involves the investigation of parallelization and performance improvement for a class of radar signal processing techniques known as space-time adaptive processing (STAP). STAP refers to an extension of adaptive antenna signal processing methods that operate on a set of radar returns gathered from multiple elements of an antenna array over a specified time interval. Because the signal returns are composed of range, pulse, and antenna-element samples, a three-dimensional (3-D) cube naturally represents STAP data. Typical STAP data cube processing requirements range from 10-100 giga floating point operations per second (Gflops). Imposed real-time deadlines for STAP applications restricts processing to parallel computers composed of numerous interconnected compute nodes (CNs). A CN has one or more processors connected to a block of shared memory.

Developing a solution to any problem on a parallel system is generally not a trivial task. The overall performance of many parallel systems is highly dependent upon network contention. In general, the mapping of data and the scheduling of communications impacts network contention of parallel architectures. The primary goals of many applications implemented on parallel architectures are to reduce latency and minimize interprocessor communication time (IPC) while maximizing throughput. It is indeed necessary to accomplish these objectives in STAP processing environments. In most STAP implementations, there are three phases of computations, one for each dimension of the data cube (i.e., range, pulse, and channel). To reduce computational latency, the processing at each phase must be distributed over multiple CNs using a single program multiple data (SPMD) approach. Additionally, prior to each processing phase, the data set must be partitioned in a fashion that attempts to equally distribute the computational load over the available CNs. Because each of the three phases process a different dimension of the data cube, the data must be redistributed to form contiguous vectors of the next dimension prior to the next processing phase. This redistribution of data or distributed "corner-turn" requires IPC. Minimizing the time required for interprocessor communication helps maximize STAP processing efficiency.

Driven by the need to solve complex real-time applications that require tremendous computational bandwidths such as STAP algorithms, commercial-off-the-shelf

(COTS) embedded high-performance computing systems that emphasize upward scalability have emerged in the parallel processing environment. In a message passing parallel system, CNs are connected with each other via a common data communication fabric or interconnection network. For the purposes of discussion and illustration, assume that a crossbar with six bidirectional channels is the building block for the interconnection network. Each of the six input/output channels is bidirectional, but may only be driven in one direction at a time. The versatility of the six-port crossbar allows for the interconnect to be configured into a number of different network topologies, including two-dimensional (2-D) and 3-D meshes, 2-D and 3-D rings, grids, and Clos networks. However, the most common configuration is a fat-tree, where the crossbars are connected in a parent-child fashion. In a fat-tree configuration, which is the configuration assumed in this paper, each crossbar has two parent ports and four child ports. The fat-tree architecture helps alleviate the problem of communication bottlenecks at high levels of the tree (present in conventional tree architectures) by increasing the number of effective parallel paths between CNs. Unfortunately, the addition of multiple paths between CNs increases the complexity of the communication pattern in applications such as STAP that involve data redistribution phases.

Additional complexity emerges when each CN is composed of more than one processor or compute element (CE) configured with the shared-memory address space of the CN. In a system with one CE per CN, the communication pattern during distributed corner-turn phases is very regular and well-understood (i.e., a matrix transpose operation implemented in parallel). However, the overall complexity of both the mapping and scheduling of communications increases in systems where the CNs contain more than one CE, for two reasons. First, the communication pattern can be less regular. Second, the message sizes are not uniform.

Two major challenges of implementing STAP algorithms on embedded high-performance systems are determining the best method for distributing the 3-D data set across CNs (i.e., the mapping strategy) and the scheduling of communication prior to each phase of computation. At each of the three phases of processing, data access is either vector-oriented along a data cube dimension or a plane-oriented combination of two data cube dimensions. During the processing at each phase, the contiguous vectors along the dimension of interest are distributed among the CNs for processing in parallel. Additionally, each CE may be responsible for processing one or more vectors of data during each phase. Before processing of the next phase can take place, the data must be redistributed among the available CNs to form contiguous vectors of the next dimension. Determining the optimal schedule of data transfers during phases of data repartitioning on a parallel system is a formidable task. The combination of these two factors, data mapping and communication scheduling, provides the key motivation for this work.

One approach to data set distribution in STAP applications is to partition the data cube into sub-cube bars (see Fig. 1). Each sub-cube bar is composed of partial data samples from two dimensions, while preserving one whole dimension of the data-cube. After performing the necessary computations on the current whole dimension, the data vectors must be redistributed to form contiguous sub-cube bars of the next dimension to be processed. By implementing a sub-cube bar partitioning scheme, IPC between processing stages is isolated to clusters of CNs and not the entire system (i.e., the required data exchanges occur only between CNs in the same logical row or column).

To illustrate the impact of mapping, consider the two examples shown in Fig. 2 and Fig. 3. For these two examples, assume that the parallel system is composed of four CNs, with each having three CEs, and connected via one six-port crossbar (see Fig 4). Additionally, the number on each sub-cube bar indicates the processor to which the sub-cube bar is initially distributed for processing. Fig. 2 illustrates a mapping scheme where the sub-cube bars are raster-numbered along the pulse dimension. In contrast, the sub-cube bars are raster-numbered along the channel dimension in Fig. 3. As illustrated in the two examples, the initial mapping of the data prior to pulse compression affects the number of required communications during the data redistribution phase prior to Doppler filtering. In the case where the data cube is raster-numbered along the pulse dimension, six messages, totaling 20 units in size, must be transferred through the interconnection network. By implementing the mapping scheme in Fig. 3, the number of required data transfers increases to twelve, while the total message size expands to 36 units. For this small example, the initial mapping of the sub-cube bars greatly affects the communication overhead that occurs during phases of data repartitioning.

To illustrate the impact of scheduling communications during data repartitioning phases, consider the problem depicted in Fig. 5, which is the same problem as shown in Fig. 3. The left-hand portion of the figure shows the current location of the STAP data cube on the given processors after pulse compression. The data cube on the right-hand side of the figure illustrates the sub-cube bars of the data cube after repartitioning. The coloring scheme indicates the destination CN of the data for the data prior to the next processing phase. If any part of the sub-cube bar is a different color than its current processor color in the left-hand data cube, the data must be transferred to the corresponding colored destination node. In this example, the repartitioning phase involves transferring six data sets through the interconnection network. If the six messages were sequentially communicated (i.e., no parallel communication) through the network, the completion time ($T_C$) would be the sum of the length of each message, which totals 20 network cycles. If two or more messages could be sent through the network concurrently, then the value of $T_C$ would be reduced (i.e., below 20).

Scheduling the communications for each of the six messages through the interconnection network greatly affects the overall performance (even for this small system consisting of only one crossbar). Fig. 6 shows the six messages, labeled A through F, in the outgoing first-in-first-out (FIFO) message queues of the source CNs. Each message's destination is indicated by its color code. The number in parenthesis by each message label represents the relative size of the message. The minimal achievable communication time is dependent upon the CN with the largest communication time of all outgoing and incoming messages. For this example, the minimum possible communication time is the sum of all outgoing and incoming messages on the CNs having two messages, which equals fourteen message units. The actual communication time, $T_C$, that would result from this example with the given message queue orderings (i.e., schedule) is 17 units. However, changing the ordering of the messages in the outgoing queues will yield an optimal schedule of messages. The message queues in Fig. 7 are identical to those in Fig. 6 except the positions of messages C and F have been swapped in the outgoing queue. Swapping the ordering of the messages on the green CN allows for an increase in the number of messages that can be communicated in parallel. For this new ordering of queued messages, the actual completion time achieves the optimal completion time of fourteen units. The purpose of this example is to illustrate that the order (i.e., the schedule) in which the messages are queued for transmission can impact how much (if any) concurrent communication can occur. The method used to decompose and map the data onto the CNs will also impact the potential for concurrent communication.

The current research involves the design and implementation of a network simulator that will model the effects of data mapping and communication scheduling on the performance of a STAP algorithm on an embedded high-performance computing platform. The purpose of the simulator is not to optimally *solve* the data mapping and scheduling problems, but to *simulate* the different data mappings and schedules and resultant performance. Thus, the simulator models the effects associated with how the data is mapped onto CNs, composed of more that one CE, of an embedded parallel system, and how the data transfers are scheduled.

The network simulator is designed in an object-oriented paradigm and implemented in Java using Borland's JBuilder Professional version 1.0. Java was chosen over other programming languages because of its added benefits. First, Java code is portable. This feature allows the simulator to run on various platforms regardless of the architecture and operating system. Additionally, Java can be used to create both applications (i.e., a program that executes on a local computer) and applets (i.e., an application that is designed to be transmitted over the Internet and executed by a Java-compatible web browser). Third, Java source code is written entirely in an object-oriented paradigm, which is well-suited for the simulator's design. Fourth, Java provides built-in support for multithreaded programming. Finally, Java development tools, like Borland's JBuilder, provide a set of tools in the Abstract Window Toolkit (AWT) for visually designing and creating graphical user interfaces (GUIs) for applications or applets.

The simulator's functionality is encompassed by a friendly GUI. The main user interface of the simulator provides a facility for the user to enter the corresponding values of the three dimensions of a given STAP data cube and the number of CNs to allocate to processing the STAP data cube using an element-space post-Doppler heuristic and a sub-cube bar partitioning scheme. After providing the problem definition information, the user selects an initial mapping that includes a set of predefined mappings (e.g., raster-numbering along the pulse dimension, raster-numbering along the channel, etc.), a random mapping, or a user-definable customized mapping. Furthermore, the user selects the ordering of the messages in the outgoing queues from a predefined set of scheduling algorithms (e.g., short messages first, longest messages first, random, custom, etc). After providing the necessary input, the network simulator simulates the defined problem and produces the timing results from both phases of data repartitioning. The level of detail that the simulator models could be defined as a medium- to fine-grained simulation of the interconnection network. The simulator assumes the network is circuit switched, and the contention resolution scheme is based on a port number tie-breaking mechanism to avoid deadlocks. In addition, the simulator incorporates a novel and efficient method of evaluating blocked messages within the interconnection network and queued messages waiting for transfer.

The simulator can be used as a tool for collecting and analyzing how the performance of a system is affected by changing the mapping and scheduling. If it is determined that mapping and/or scheduling choices have a significant impact on performance, then the simulator will serve as a basis for future research in determining the optimal mappings and communications.

## Acknowledgements

This work was supported by Rome Laboratory under Grant No. F30602-96-1-0098 and Defense Advanced Research Projects Agency (DARPA) under Contract No. F30602-97-2-0297.

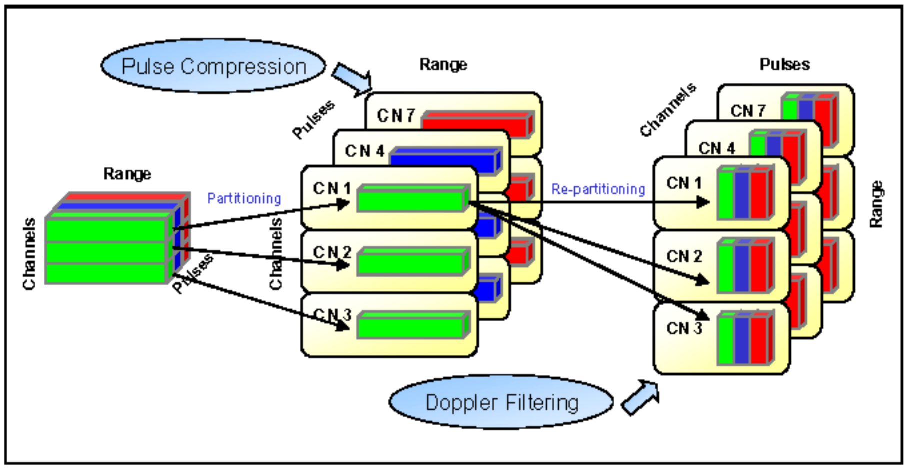


**Fig. 1.** STAP data cube partitioning by sub-cube bars
(This method of partitioning was first described in [1]).

[1] M. F. Skalabrin and T. H. Einstein, "STAP Processing on a Multicomputer: Distribution of 3-D Data Sets and Processor Allocation for Optimum Interprocessor Communication," *Proceedings of the Adaptive Sensor Array Processing (ASAP) Workshop*, March 1996.

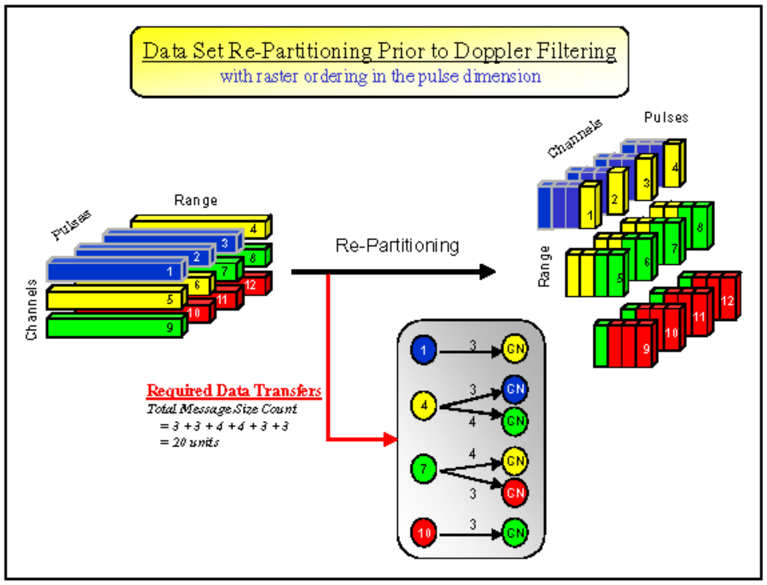


Fig. 2 Data set repartitioning with raster-numbering along the pulse dimension.

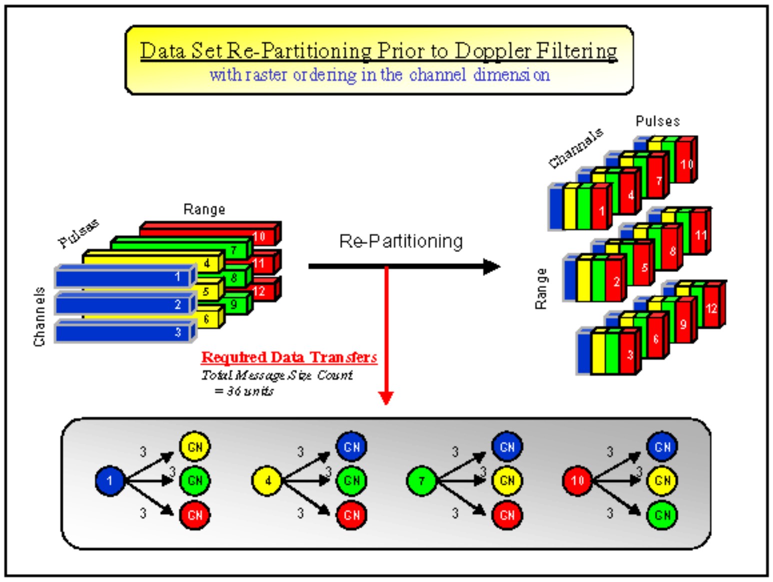


Fig. 3 Data set repartitioning with raster-numbering along the channel dimension.

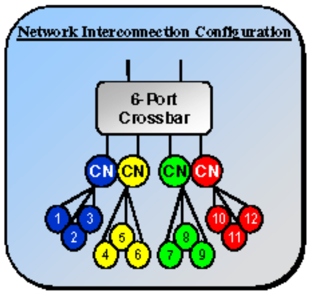


Fig. 4 An example configuration of a four CN (twelve CE) interconnection network.

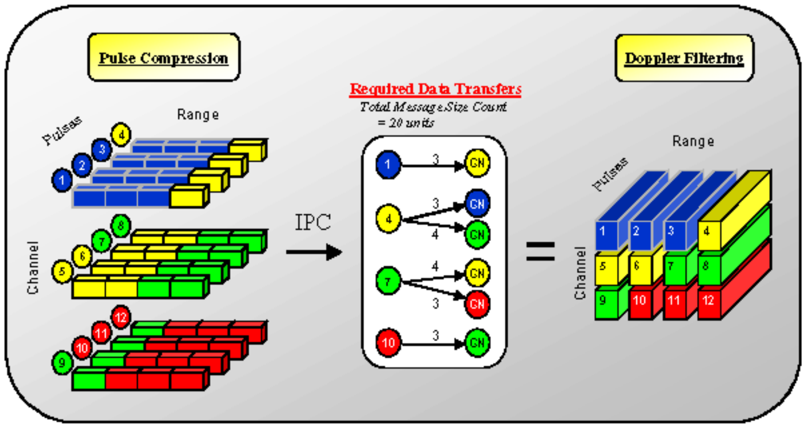


Fig. 5 An example of sub-cube bar repartitioning prior to Doppler filtering.

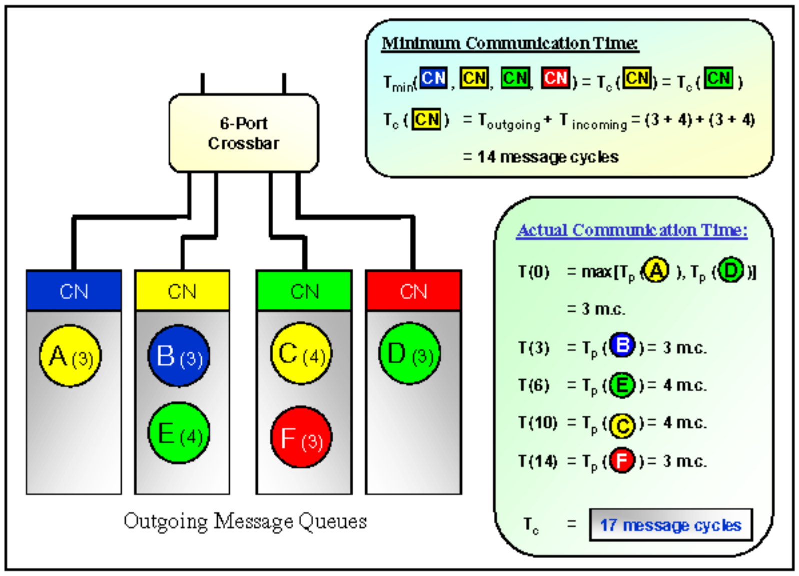

Minimum Communication Time:
$T_{min}$(CN, CN, CN, CN) = $T_c$(CN) = $T_c$(CN)
$T_c$(CN) = $T_{outgoing}$ + $T_{incoming}$ = (3 + 4) + (3 + 4)
= 14 message cycles
6-Port Crossbar
CN
CN
CN
CN
A (3)
B (3)
E (4)
C (4)
F (3)
D (3)
Outgoing Message Queues
Actual Communication Time:
T(0) = max[$T_p$(A), $T_p$(D)]
= 3 m.c.
T(3) = $T_p$(B) = 3 m.c.
T(6) = $T_p$(E) = 4 m.c.
T(10) = $T_p$(C) = 4 m.c.
T(14) = $T_p$(F) = 3 m.c.
$T_c$ = 17 message cycles

Fig. 6 A sub-optimal communication scheduling example.

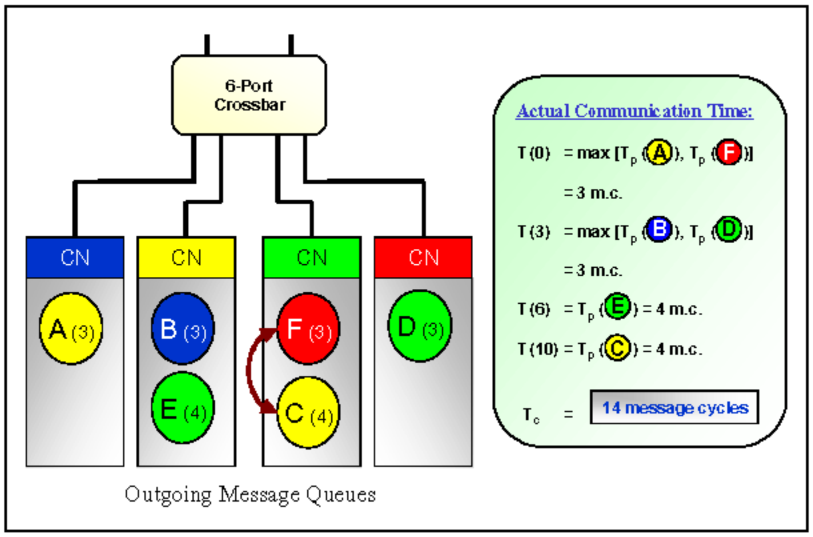

6-Port Crossbar
CN
CN
CN
CN
A (3)
B (3)
E (4)
F (3)
C (4)
D (3)
Outgoing Message Queues
Actual Communication Time:
T(0) = max [$T_p$(A), $T_p$(F)]
= 3 m.c.
T(3) = max [$T_p$(B), $T_p$(D)]
= 3 m.c.
T(6) = $T_p$(E) = 4 m.c.
T(10) = $T_p$(C) = 4 m.c.
$T_c$ = 14 message cycles

Fig. 7 An optimal communication scheduling example.

**Appendix D:** Jack M. West and John K. Antonio, "A Genetic Algorithm Approach to Scheduling Communications for a Class of Parallel Space-Time Adaptive Processing Algorithms," *Proceedings of the 5th International Workshop on Embedded/Distributed HPC Systems and Applications (EHPC 2000)*, in *Lecture Notes in Computer Science*, *IPDPS 2000 Workshops*, sponsor: IEEE Computer Society, Cancun, Mexico, May 2000, pp. 855-861.

# A Genetic Algorithm Approach to Scheduling Communications for a Class of Parallel Space-Time Adaptive Processing Algorithms

Jack M. West and John K. Antonio

School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019
Phone: (405) 325-4624
{west, antonio}@ou.edu

**Abstract.** An important consideration in the maximization of performance in parallel processing systems is scheduling the communication of messages during phases of data movement to reduce network contention and overall communication time. The work presented in this paper focuses on off-line optimization of message schedules for a class of radar signal processing techniques know as space-time adaptive processing on a parallel embedded system. In this work, a genetic-algorithm-based approach for optimizing the scheduling of messages is introduced. Preliminary results indicate that the proposed genetic approach to message scheduling can provide significant decreases in the communication time.

## 1 Introduction and Background

For an application on a parallel and embedded system to achieve required performance given tight system constraints, it is important to efficiently map the tasks and/or data of the application onto the processors to the reduce inter-processor communication traffic. In addition to mapping tasks efficiently, it is also important to schedule the communication of messages in a manner that minimizes network contention so as to achieve the smallest possible communication time.

Mapping and scheduling can both – either independently or in combination – be cast as optimization problems, and optimizing mapping and scheduling objectives can be critical to the performance of the overall system. For parallel and embedded systems, great significance is placed on minimizing execution time (which includes both computation and communication components) and/or maximizing throughput.

The work outlined in this paper involves optimizing the scheduling of messages for a class of radar signal processing techniques known as space-time adaptive processing (STAP) on a parallel and embedded system. A genetic algorithm (GA) based approach for solving the message-scheduling problem for the class of parallel STAP algorithms is proposed and preliminary results are provided. The GA-based optimization is performed off-line, and the results of this optimization are static schedules for each compute node in the parallel system. These static schedules are then used within the on-line parallel STAP implementation. The results of the study show that significant improvement in communication time performance are possible using the proposed approach for scheduling. Performance of the schedules were evaluated using a RACEway network simulator [6].

## 2 Overview of Parallel STAP

STAP is an adaptive signal processing method that simultaneously combines the signals received from multiple elements of an antenna array (the spatial domain) and from multiple pulses (the temporal domain) of a coherent processing interval [5]. The focus of this research assumes STAP is implemented using an element-space post-Doppler partially adaptive algorithm, refer to [5, 6] for details. Algorithms belonging to the class of element-space post-Doppler STAP perform filtering on the data along the pulse dimension, referred to as Doppler filtering, for each channel prior to adaptive filtering. After Doppler filtering, an adaptive weight problem is solved for each range and pulse data vector.

The parallel computer under investigation for this work is the Mercury RACE® multicomputer. The RACE® multicomputer consists of a scalable network of compute nodes (CNs), as well as various high-speed I/O devices, all interconnected by Mercury's RACEway interconnection fabric [4]. A high-level diagram of a 16-CN RACEway topology is illustrated in Figure 1. The interconnection fabric is configured in a fat-tree architecture and is a circuit switched network. The RACEway interconnection fabric is composed of a network of crossbar switches and provides high-speed data communication between different CNs. The Mercury multicomputer can support a heterogeneous collection of CNs (e.g., SHARC and PowerPCs), for more details refer to [6].

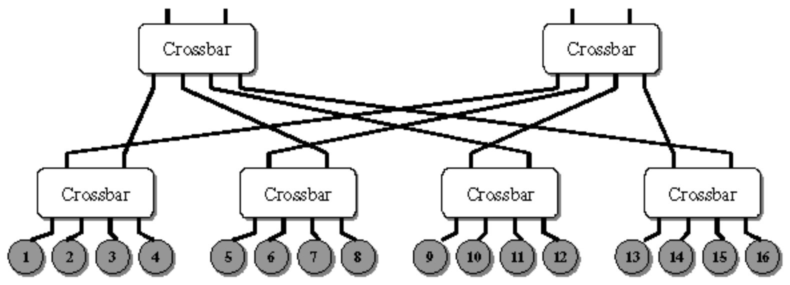


**Fig. 1.** Mercury RACE® Fat-Tree Architecture configured with 16 CNs.

Achieving real-time performance requirements for STAP algorithms on a parallel embedded system like the Mercury multicomputer largely depends on two major issues. First is determining the best method for distributing the 3-D STAP data cube across CNs of the multiprocessor system (i.e., the mapping strategy). Second is determining the scheduling of communications between phases of computation. In general, STAP algorithms contain three phases of processing, one for each dimension of the data cube (i.e., range, pulse, channel). During each phase of processing, the vectors along the dimension of interest are distributed as equally as possible among the processors for processing in parallel. An approach to data set partitioning in STAP applications is to partition the data cube into sub-cube bars. Each sub-cube bar is composed of partial data samples from two dimensions while preserving one whole dimension for processing. The work here assumes a sub-cube bar partitioning of the STAP data cube, for further details refer to [6]. Figure 2 shows an example of how sub-cube partitioning is applied to partition a 3-D data cube across 12 CNs.

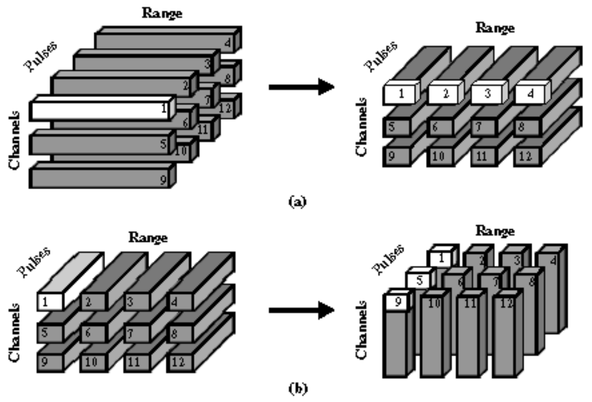


**Fig. 2.** Illustration of the sub-cube bar mapping technique for the case of 12 CNs. The mapping of the sub-cube bars to CNs defines the required data communications. (a) Example illustration of the communication requirements from CN 1 to the other CNs (2, 3, and 4) after completion of the range processing and prior to Doppler processing. (b) Example illustration of the communication requirements from CN 1 to other CNs (5 and 9) after the completion of Doppler processing and prior to adaptive weight processing.

During phases of data redistribution (i.e., communication) between computational phases, the number of required communications and the communication pattern among the CNs is dependant upon how the data cube is mapped onto the CNs. For example, in Figure 2(a) the mapping of sub-cube bars to CNs dictates that after range processing, CN 1 must transfer portions of it data sub-cube bar to CNs 2, 3, and 4. (Each of the other CNs, likewise, is required to send portions of their sub-cube bar to CNs on the same row.) The scheduling (i.e., ordering) of outgoing messages at each CN impacts the resulting communication time. For example, in Figure 2(a) note CN 1 could order its outgoing messages according to one of 3! = 6 permutations (i.e., [2,3,4], [3,2,4], etc.). Similarly, a scheduling of outgoing messages must be defined for each CN. Improper schedule selection can result in excessive network contention and thereby increase the time to perform all communications between processing phases. The focus in this paper is on optimization of message scheduling, for a fixed mapping, using a genetic algorithm methodology.

## 3 Genetic Algorithm Methodology

A GA is a population-based model that uses selection and recombination operators to generate new sample points in the solution space [3]. A GA encodes a potential solution to a specific problem on a chromosome-like data structure and applies recombination operators to these structures in a manner that preserves critical information. Reproduction opportunities are applied in such a way that those chromosomes representing a better solution to the target problem are given more chances to reproduce than chromosomes with poorer solutions. GAs are a promising heuristic approach to locating near-optimal solutions in large search spaces [3]. For a complete discussion of GAs, the reader is referred to [1, 3].

Typically, a GA is composed of two main components, which are problem dependent: the *encoding problem* and the *evaluation function.* The *encoding problem* involves generating an encoding scheme to represent the possible solutions to the optimization problem. In this research, a candidate solution (i.e., a chromosome) is encoded to represent valid message schedules for all of the CNs. The *evaluation function* measures the quality of a particular solution. Each chromosome is associated with a fitness value, which in this case is the completion time of the schedule represented by the given chromosome. For this research, the smallest fitness value represents the better solution. The "fitness" of a candidate is calculated here based on its simulated performance. In previous work [6, 7], a software simulator was developed to model the communication traffic for a set of messages on the Mercury RACEway network. The simulation tool is used here to measure the "fitness" (i.e., the completion time) of the schedule of messages represented by each chromosome.

Chromosomes evolve through successive iterations, called generations. To create the next generation, new chromosomes, called offspring, are formed by (a) merging two chromosomes from the current population together using a crossover operator or (b) modifying a chromosome using a mutation operator. Crossover, the main genetic operator, generates valid offspring by combining features of two parent chromosomes. Chromosomes are combined together at a defined crossover rate, which is defined as the ratio of the number of offspring produced in each generation to the population size. Mutation, a background operator, produces spontaneous random changes in various chromosomes. Mutation serves the critical role of either replacing the chromosomes lost from the population during the selection process or introducing new chromosomes that were not present in the initial population. The mutation rate controls the rate at which new chromosomes are introduced into the population. In this paper, results are based on the implementation of a position-based crossover operator and an insertion mutation operator, refer to [1] for details.

Selection is the process of keeping and eliminating chromosomes in the population based on their relative quality or fitness. In most practices, a roulette wheel approach, either rank-based or value-based, is adopted as the selection procedure. In a ranked-based selection scheme, the population is sorted according to the fitness values. Each chromosome is assigned a sector of the roulette wheel based on its ranked-value and not the actual fitness value. In contrast, a value-based selection scheme assigns roulette wheel sectors proportional to the fitness value of the chromosomes. In this paper, a ranked-based selection scheme is used. Advantages of rank-based fitness assignment is it provides uniform scaling across chromosomes in the population and is less sensitive to probability-based selections, refer to [3] for details.

## 4 Numerical Results

In the experiments reported in this section, it is assumed that the Mercury multicomputer is configured with 32 PowerPC compute nodes. For range processing, Doppler filtering, and adaptive weight computation, the 3-D STAP data cube is mapped onto the 32 processing elements based on an $8 \times 4$ process set (i.e., 8 rows and 4 columns), refer to [2, 6]. The strategy implemented for CN assignment in a process set is raster-order from left-to-right starting with row one and column one for all process sets. (The process sets not only define the allocation of the CNs to the data but also the required data transfers during phases of data redistribution.) The STAP data cube consists of 240 range bins, 32 pulses, and 16 antenna elements.

For each genetic-based scenario, 40 random schedules were generated for the initial population. The poorest 20 schedules were eliminated from the initial population, and the GA population size was kept a constant 20. The recombination operators included a position-based crossover algorithm and an insertion mutation algorithm. A ranked-based selection scheme was assumed with the angle ratio of sectors on the roulette wheel for two adjacently ranked chromosomes to be $1 + 1/P$, where $P$ is the population size. The stopping criteria were: (1) the number of generations reached 500; (2) the current population converged (i.e., all the chromosomes have the same fitness value); or (3) the current best solution had not improved in the last 150 generations.

Figure 3 shows the simulated completion time for three genetic-based message scheduling scenarios for the data transfers required between range and Doppler processing phases. Figure 4 illustrates the simulated completion time for the same three genetic-based message scheduling scenarios for the data transfers required between Doppler and adaptive weight processing phases. In the first genetic scenario (GA 1), the crossover rate ($P_{xover}$) is 20% and the mutation rate ($P_{mut}$) is 4%. For GA 2, $P_{xover}$ is 50% and $P_{mut}$ is 10%. For GA 3, $P_{xover}$ is 90% and $P_{mut}$ is 50%. Figures 3 and 4 provide preliminary indication that for a fixed mapping the genetic-algorithm-based heuristic is capable of improving the scheduling of messages, thus providing improved performance. All three genetic-based scenarios improve the completion time for both communication phases. In each phase, GA 2 records the best schedule of messages (i.e., the smallest completion time).

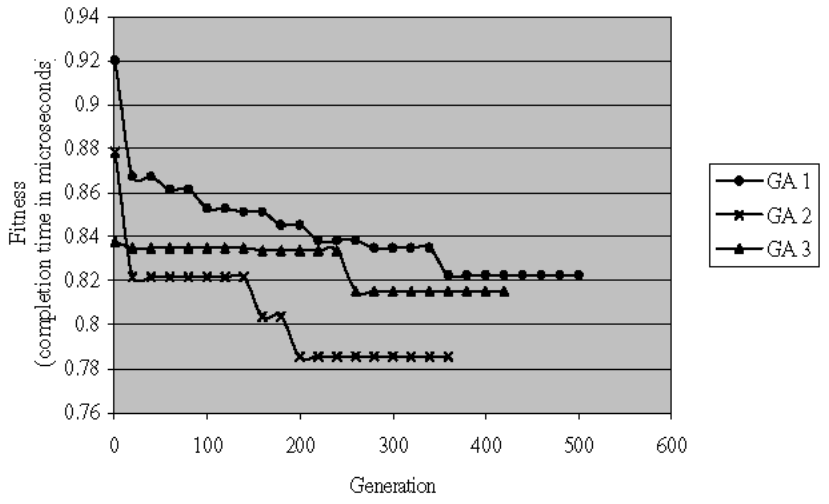


**Fig. 3.** Simulated completion time of the communication requirements for data redistribution after range processing and prior to Doppler processing for the parameters discussed in Section 4. For GA 1, the crossover rate ($P_{xover}$) = 20% and the mutation rate ($P_{mut}$) = 4%. For GA 2, $P_{xover}$ = 50% and $P_{mut}$ = 10%. For GA 3, $P_{xover}$ = 90% and $P_{mut}$ = 50%.

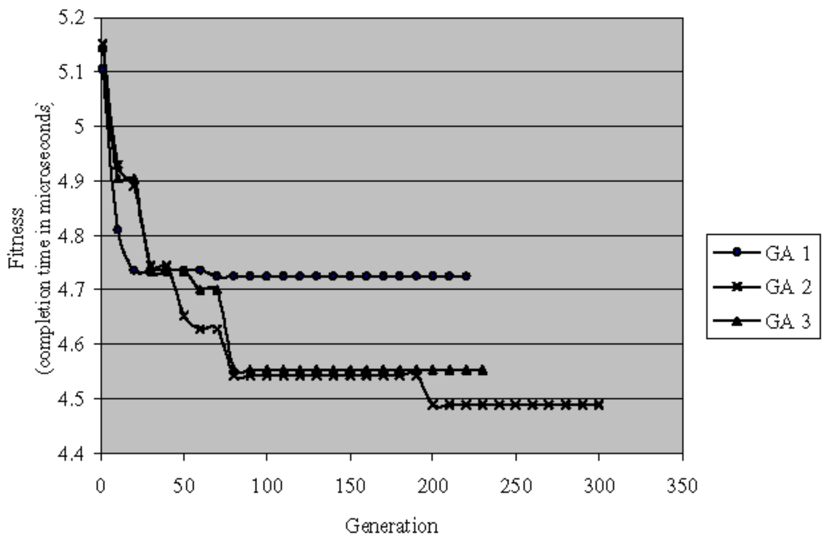


**Fig. 4.** Simulated completion time of the communication requirements for data redistribution after Doppler processing and prior to adaptive weight computation for the parameters stated in Section 4. For GA 1, the crossover rate ($P_{xover}$) = 20% and the mutation rate ($P_{mut}$) = 4%. For GA 2, $P_{xover}$ = 50% and $P_{mut}$ = 10%. For GA 3, $P_{xover}$ = 90% and $P_{mut}$ = 50%.

## 5. Conclusion

In conclusion, preliminary data demonstrates that off-line GA-based message scheduling optimization can provide improved performance in a parallel system. Future work will be conducted to more completely study the effect of changing parameters of the GA, including crossover and mutation rates as well as the methods used for crossover and mutation. Finally, future studies will be conducted to determine the performance improvement between a randomly selected scheduling solution and the one determined by the GA. In Figures 3 and 4, the improvements shown are conservative in the sense that the initial generations' performance on the plots represents the best of 40 randomly generated chromosomes (i.e., solutions). It will be interesting to determine improvements of the GA solutions with respect to the "average" and "worst" randomly generated solutions in the initial population.

## Acknowledgements

This work was supported by DARPA under contract no. F30602-97-2-0297.

## References

1. M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, Inc., New York, NY, 1997.
2. M. F. Skalabrin and T. H. Einstein, "STAP Processing on a Multicomputer: Distribution of 3-D Data Sets and Processor Allocation for Optimum Interprocessor Communication," *Proceedings of the Adaptive Sensor Array Processing (ASAP) Workshop*, March 1996.
3. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *Journal of Parallel and Distributed Computing*, Special Issue on Parallel Evolutionary Computing, Vol. 47, No 1, pp. 8-22, Nov. 25, 1997.
4. The RACE Multicomputer, Hardware Theory of Operation: Processors, I/O Interface, and RACEway Interconnect, Volume I, ver. 1.3.
5. J. Ward, Space-Time Adaptive Processing for Airborne Radar, Technical Report 1015, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA, 1994.
6. J. M. West, *Simulation of Communication Time for a Space-Time Adaptive Processing Algorithm Implemented on a Parallel Embedded System*, Master's Thesis, Computer Science, Texas Tech University, 1998.
7. J. M. West and J. K. Antonio, "Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System," *Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC '98)*, in *Lecture Notes in Computer Science 1388: Parallel and Distributed Processing*, edited by Jose Rolim, sponsor: IEEE Computer Society, Orlando, FL, USA, Apr. 1998, pp. 979-986.

**Appendix E:** Jack M. West and John K. Antonio, "A Genetic-Algorithm Approach to Scheduling Communications for Embedded Parallel Space-Time Adaptive Processing Algorithms," *Journal of Parallel and Distributed Computing*, Vol. 62, No. 9, Sept. 2002, pp. 1386-1406.

Journal of Parallel and Distributed Computing **62**, 1386–1406 (2002)
doi:10.1006/jpdc.2002.1852

# A Genetic-Algorithm Approach to Scheduling Communications for Embedded Parallel Space–Time Adaptive Processing Algorithms

Jack M. West and John K. Antonio[1]

*School of Computer Science, University of Oklahoma, 200 Felgar Street, Norman, Oklahoma 73019-6151*
E-mail: west@ou.edu; antonio@ou.edu

Received March 29, 2000; accepted January 29, 2002

Computational efficiency is of great significance for high-performance embedded applications. The work here develops and evaluates a genetic-algorithm-based (GA-based) optimization technique for the scheduling of messages for a class of parallel embedded signal processing techniques known as space–time adaptive processing (STAP). The GA-based optimization is performed off-line, resulting in static schedules for the compute nodes of the parallel system. These static schedules are utilized for the on-line implementation of the parallel STAP application. The primary motivation and justification for devoting significant off-line effort to solving the formulated scheduling problem is the resulting reduction of hardware resources required for the actual on-line implementation. Numerical studies illustrate that reductions in hardware requirements of around 50% can be achieved by employing the results of the proposed scheduling techniques. This reduction in hardware requirement is of critical importance for STAP, which is typically an airborne application in which the size, weight, and power consumption of the computational platform are severely constrained. © 2002 Elsevier Science (USA)

*Key Words:* embedded processing; genetic algorithms; hardware minimization; mapping; scheduling.

## 1. INTRODUCTION

For an application implemented on a parallel and embedded system to achieve required performance, it is important to effectively map the tasks of the application onto the processors in a way that reduces the volume of inter-processor communication traffic. It is also important to schedule the communication of

[1]To whom correspondence should be addressed.

messages in a manner that minimizes network contention so as to achieve the smallest possible communication times.

Mapping and scheduling can both—either independently or in combination—be cast as optimization problems, and optimizing mapping and scheduling objectives can be critical to the performance of the overall system. For embedded applications, great importance is often placed on determining minimal hardware requirements that can support a number of different application scenarios. This is because there are typically tight constraints on the amount of hardware that can be accommodated within the embedded platform. Using mappings and schedules that minimize the communication time of parallel and embedded applications can increase the overall efficiency of the parallel system, thus leading to reduced hardware requirements for a given set of application scenarios.

The work outlined in this paper focuses on using a genetic-algorithm-based (GA-based) approach to optimize the scheduling of messages for a class of parallel radar signal processing algorithms known as space–time adaptive processing (STAP). STAP is an adaptive signal processing method that simultaneously combines the signals received from multiple elements of an antenna array (the spatial domain) and from multiple pulses (the temporal domain) of a coherent processing interval [6]. The focus of this research assumes that STAP is implemented using an element-space post-Doppler partially adaptive algorithm, refer to Appendix A and [6, 7] for details.

STAP involves signal processing methods that operate on data collected from a set of spatially distributed sensors over a given time interval. Signal returns are composed of range, pulse, and antenna-element digital samples; consequently, a three-dimensional (3-D) data cube naturally represents the STAP data. A distributed memory multiprocessor machine is assumed here for the parallel STAP implementation. The core processing requirement proceeds in three distinct phases of computation, one associated with each dimension of the STAP data cube. After each phase of processing, the data must be re-distributed across the processors of the machine, which represents the communication requirements of this parallel application. Thus, there are two primary phases of inter-processor data communication required: one between the first and second phases of processing and the other between the second and third phases of processing. After all three phases of processing are complete for a given STAP data cube, a new data cube is input into the parallel machine for processing.

A proposed GA-based approach is used to solve the message-scheduling problem associated with each of the two phases of inter-processor data communication. This GA-based optimization is performed off-line, and the results of this optimization are static schedules for the compute nodes of the parallel system. These static schedules are used within the on-line parallel STAP implementation. The results of the study show that significant improvements in communication time performance are possible using the proposed approach for scheduling. It is then shown that these improvements in communication time translate to reductions in required hardware for a class of scenarios. Performance of the mappings and schedules are evaluated based on a Mercury RACE$^{R}$ network simulator developed in [7] and described in Appendix C.

The rest of the paper is organized as follows. Section 2 investigates the issue of defining suitable mappings of the STAP data cube onto the multiprocessor system. The GA-based approach for scheduling messages associated with the two phases on inter-processor communication is given in Section 3. The benefits of using the GA-based approach are illustrated through numerical studies in Section 4, followed by conclusions in Section 5.

## 2. DATA MAPPING FRAMEWORK

For this work, the STAP data cube is partitioned into sub-cube bars of vectors where each bar is mapped onto a given compute node (CN), refer to Appendix B for more details. A two-dimensional process set, as described in [8], defines the mapping of data onto CNs for each computational phase. Additionally, the process set defines the communication pattern for the required "distributed corner turns" of the STAP data cube [3].

Figure 1 illustrates the application of a two-dimensional process set to a STAP data cube prior to processing contiguous data in the range dimension. The STAP data is distributed to the processors based on the process set definition. Defining a process set requires two important steps. First, the two dimensions of the process set should be specified such that the product of the two dimensions is not greater than the number of available processors. Second, each CN number should be assigned a location (row and column) in the process set. In this example, the STAP data cube, which contains $L$ range samples, $M$ pulse samples, and $N$ channel elements, is partitioned by a $3 \times 4$ process set (i.e., three rows and four columns for a total of 12 CNs). The $3 \times 4$ process set defines the partitioning of the data cube prior to range processing. The CNs are assigned in a raster ordering from left to right. Each of the 12 CNs is assigned a sub-cube of contiguous data vectors of size $L \times \frac{M}{4} \times \frac{N}{3}$ based on their respective location in the process set.

Recall that STAP requires three phases of processing, one associated with each dimension of the data cube. Consequently, a process set must be defined for each

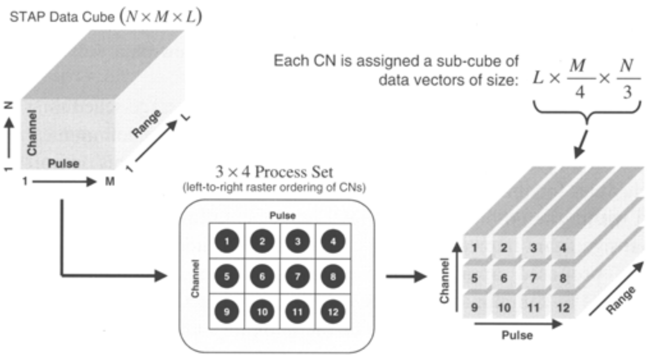

STAP Data Cube $(N \times M \times L)$
Channel
Pulse
Range
N
1
M
L
Each CN is assigned a sub-cube of data vectors of size: $L \times \frac{M}{4} \times \frac{N}{3}$
$3 \times 4$ Process Set
(left-to-right raster ordering of CNs)
Pulse
Channel
1 2 3 4
5 6 7 8
9 10 11 12
Range

FIG. 1. Process set partitioning of a STAP data cube for range processing.

phase of processing. The process sets not only define the allocation of CNs to data but also the required data transfers during phases of data redistribution. To illustrate, let $T_1$ represent the process set for range processing and $T_2$ define the process set for processing in the pulse dimension. The process sets $T_1$ and $T_2$ define the required message traffic to form contiguous vectors in the pulse dimension after range processing is complete. The row and column dimensions of $T_1$ and $T_2$ affect the communication pattern that is induced for the first communication phase. Similarly, the row and column dimensions of $T_2$ and $T_3$ affect the volume and pattern of the second communication phase. Refer to Appendix B for a more detailed explanation of how mapping choices impact communication requirements.

The possible values for the row and column dimensions of a given process set, denoted by $(R, C)$, is defined by the following:

$$(R, C) \in \{(i, j) \mid i\,j = p\}, \tag{1}$$

where $p$ is the number of processors (i.e., the number of CNs). A complete mapping is defined by specifying the dimensions of all three process sets; thus, the number of complete data cube mappings is given by

$$|\{(i, j) \mid i\,j = p\}|^3. \tag{2}$$

To illustrate, for $p = 12$ there are six possible process sets: $\{(1,12),(2,6),(3,4),(4,3),(6,2),(12,1)\}$. Because a process set must be applied to each of the three dimensions of the data cube, there are a total of $6^3 = 216$ possible mapping alternatives. It is noted that the number of possible *schedules* associated with a single mapping is generally much larger than the number of mappings. In Section 3, a GA-approach to optimal scheduling for a given mapping is developed.

Based on the class of mappings defined above, an objective function is developed next for defining the merit of individual mappings. The mapping objective function quantifies the quality of the mapping associated with a collection of three process sets. The message size and the distance each message must travel (i.e., the number of crossbar connections required for transmission) are key parameters of the objective function. The process sets $T_1$ and $T_2$ induce message traffic requirements as do the process sets $T_2$ and $T_3$. The induced message traffic produced by process sets $T_1$ and $T_2$ is quantified using the following expression:

$$\sum_{(i,\, j) \in S_1} |m_{ij}| d_{ij}, \tag{3}$$

where $S_1$ represents the set of all messages induced by process sets $T_1$ and $T_2$, $m_{ij}$ defines a message from CN $i$ to CN $j$, $|m_{ij}|$ is the message size, $d_{ij}$ is the distance the message traverses from source to destination. By combing the above expression with a similar expression for the message traffic between process sets $T_2$ and $T_3$, an objective measure of overall mapping quality is defined as

$$\sum_{(i,\, j) \in S_1} |m_{ij}| d_{ij} + \sum_{(i,\, j) \in S_2} |m_{ij}| d_{ij}. \tag{4}$$

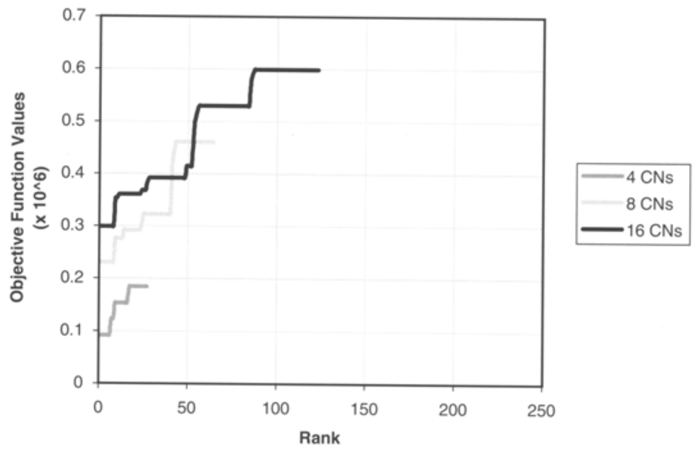


**FIG. 2.** Illustration of the mapping objective function values associated with mapping a 240 × 32 × 16 STAP data cube mapped onto a 4, 8, and 16 CN system.

Figure 2 compares mapping objective function values associated with mapping a 240 × 32 × 16 STAP data cube onto a 4, 8 and 16 CN system. The results shown in the figure illustrate that as the number of CNs is increased, the (ranked) objective function values also increase. It should be noted that the addition of more CNs decreases the underlying parallel computation time because the data cube size is fixed and thus each CN has less work to accomplish. However, the addition of more processors provides a greater dissection of the data; thus, repartitioning the data among computational stages requires a greater number of messages between a greater number of processors. More messages and more processors generally require greater communication time and more network resources. The overall goal, of course, is to produce the minimal overall execution time (which includes both computation and communication phases). This goal will be considered later in Section 4.

In general, the results in Fig. 2 (and many more studies conducted in [8]) illustrate a wide variation of objective function values. In addition to this variation, it was discovered that multiple mappings produce the same mapping objective function values. The computed ratio in objective function values between good and poor mappings range from one-third to one-half, refer to [8] for more detailed studies.

It is noted that there is no guarantee that a mapping having a minimal objective function value, once implemented, is necessarily the best overall choice. This is because the scheduling of messages also has a significant impact on performance. How to optimally schedule messages for a given mapping using a GA-based approach is the topic of the next section.

## 3. GENETIC-ALGORITHM APPROACH TO MESSAGE SCHEDULING

A GA is a population-based model that uses selection and recombination operators to generate new sample points in a solution space [5]. A GA encodes a potential solution to a specific problem on a chromosome-like data structure and applies recombination operators to these structures in a manner that preserves critical information. Reproduction opportunities are applied in such a way that those chromosomes representing a better solution to the target problem are given more chances to reproduce than chromosomes with poorer solutions. GAs are a promising heuristic approach to locating near-optimal solutions in large search spaces [5]. For a complete discussion of GAs, the reader is referred to [2, 5, 8].

Typically, a GA is composed of two main components, which are problem dependent: the *encoding problem* and the *evaluation function*. The *encoding problem* involves generating an encoding scheme to represent the possible solutions to the optimization problem. In this research, a candidate solution (i.e., a chromosome) is encoded to represent valid message schedules for all of the CNs. The *evaluation function* measures the quality of a particular solution. Each chromosome is associated with a fitness value, which in this case is the simulated completion time of the schedule represented by the given chromosome. For this research, smaller completion times indicate better fitness. The network simulator described in Appendix C is used to determine the communication time of the schedule encoded by each chromosome.

Chromosomes evolve through successive iterations, called generations. A new generation is created when new chromosomes, called offspring, are formed by (a) merging two chromosomes from the current population together using a crossover operator or (b) modifying a chromosome using a mutation operator. Crossover, the main genetic operator, generates valid offspring by combining features of two parent chromosomes. Chromosomes are combined together at a defined crossover rate, which is defined as the ratio of the number of offspring produced in each generation to the population size. Mutation, a background operator, produces spontaneous random changes in various chromosomes. Mutation serves the critical role of either replacing the chromosomes lost from the population during the selection process or introducing new chromosomes that were not present in the initial population. The mutation rate controls the rate at which new chromosomes are introduced into the population. In this paper, results are based on the implementation of a position-based crossover operator and an insertion mutation operator, refer to [2] for details.

Selection is the process of ordering (i.e., ranking) chromosomes in the population by their fitness values from the best to worst. There are two fundamental paradigms for implementing the selection process: (1) value-based roulette wheel selection scheme and (2) rank-based roulette wheel selection scheme. In a value-based scheme, the probability of a chromosome being selected for reproduction is proportional to its fitness value. Each chromosome is allocated a sector on a roulette wheel proportional to its fitness value. To better illustrate the value-based approach to selection, let $P$ denote the population size and $A_i$ denote the angle allocated to the $i$th chromosome. In addition, let $f_i$ represent the fitness of the $i$th chromosome, and let the average fitness of the population be $f_{\text{avg}}$. In this selection scheme, the $i$th chromosome is allocated a sector of the roulette wheel with area proportional to $f_{avg}/f_i$ [5]. This proportionality assumes the best chromosome has the smaller fitness value; therefore, it is allocated a larger slice of the roulette wheel.

In a value-based scheme, chromosomes with the same fitness values have the same probability of being selected. In contrast, chromosomes in a rank-based scheme that have the same fitness value are arbitrarily ranked among themselves. The 0th ranked chromosome is the fittest and has the sector with the largest angle $A_0$; the $(P-1)$th ranked chromosome is the least fit and has the smallest angle $A_{p-1}$ [5]. The ratio between two adjacent chromosomes is a constant $R = A_i/A_{i+1}$. If the 360° of the roulette wheel are normalized to one, then

$$A_i = R^{P-i-1} \times \frac{(1-R)}{(1-R^P)}, \tag{5}$$

where $R > 1$, $0 \leqslant i < P$, and $0 < A_i < 1$ [5].

The selection step involves the generation of $P$ uniformly distributed random numbers ranging from zero to one. Each number maps to a location on the roulette wheel, thereby selecting the chromosome allocating that sector of the wheel. Because better solutions occupy larger portions of the wheel than poorer solutions, the better candidates have a higher probability of selection. This selection process produces $P$ candidates for recombination and mutation operations, where multiple copies of the same candidate are permissible. For this research, the size of the next generation is always kept a constant $P$, and a rank-based selection scheme is used. Advantages of rank-based fitness assignment is, it provides uniform scaling across chromosomes in the population and is less sensitive to probability-based selections, refer to [5] for details.

As successive generations emerge in the GA heuristic, it is important to compare the best solution found thus far to the best solution in the current population. The best solution is updated whenever the fitness value (i.e., the completion time) of a particular candidate is smaller than the current best solution. After evaluating and possibly updating the best solution, the stopping criteria are evaluated. The algorithm terminates if one of the stopping criteria are true, otherwise the algorithm continues by performing the states of selection, crossover, and mutation.

The optimization of schedules during phases of data redistribution between CNs on the parallel system can be viewed as a problem with discrete objects (i.e., the source and destination locations of the messages are fundamental to the encoding of the chromosomes). Optimization problems involving discrete data sets are called combinatorial optimization problems. In traditional genetic-based algorithms, chromosomes are represented as binary strings. However, this representation is not well suited for all combinatorial problems. The most natural representation, and the one implemented in this research, is a permutation representation. In this approach, messages are listed in the order in which they appear in each CN queue by a decimal number representing the destination node of the message. This representation (see Fig. 3) is called path representation.

The illustrative example in Fig. 3 shows four CNs with associated message queues. The boxes represent a message, and the number in the box indicates the destination

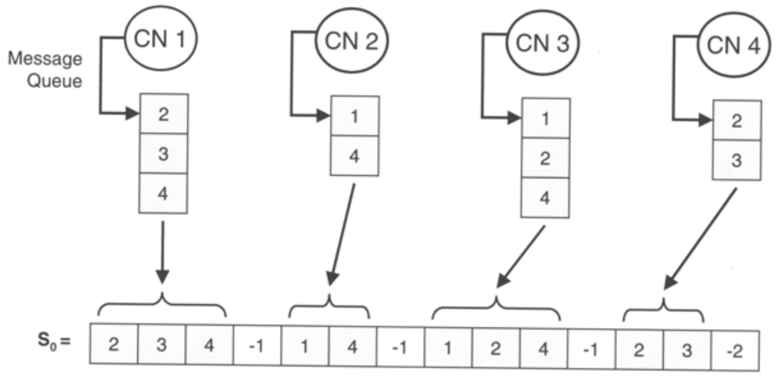


**FIG. 3.** Illustration of the encoding of a chromosome.

of the message. For example, CN 1 needs to send a message to CN 2, 3, and 4. A chromosome for a schedule is composed of a particular ordering of all the messages for all four CNs. One permutation of the messages for each CN is combined together to form a chromosome, in this case $S_0$. The message identifiers for the CNs are appended in sequence starting with the first CN. For the work in this paper, special separation tags (i.e., $-1$ values) are inserted between the orderings of successive CNs. Crossover operations are applied to sections of the chromosome bounded by the separation tags, thus eliminating the prospect of creating invalid chromosomes. In addition to separation tags, a termination tag (i.e., $-2$) is appended to the end of the chromosome to signal the end of the chromosome.

## 4. NUMERICAL RESULTS

### *4.1. GA-Based Message Scheduling*

The GA-based approach is applied to the two phases of communications induced by a given mapping for a given data cube. Forty random schedules were initially generated; then the poorest 20 schedules were eliminated from this population and the GA population size was kept a constant 20 after each iteration (i.e., generation). The recombination operators included a position-based crossover algorithm and an insertion mutation algorithm. A rank-based selection scheme was employed with the angle ratio of sectors on the roulette wheel for two adjacently ranked chromosomes to be $1 + 1/P$, where $P$ is the population size. The stopping criteria were: (1) the number of generations reached 500; (2) the current population converged (i.e., all the chromosomes have the same fitness value); or (3) the current best solution had not improved in the last 150 generations.

Figures 4 and 5 show the evolution of fitness values—which are simulated communication times—for the GA-based message scheduling approach applied to the two phases of communication induced by the $[8 \times 4, 8 \times 4, 8 \times 4]$ mapping (32 CNs) for a STAP data cube assumed to have 240 range bins, 32 pulses, and 16

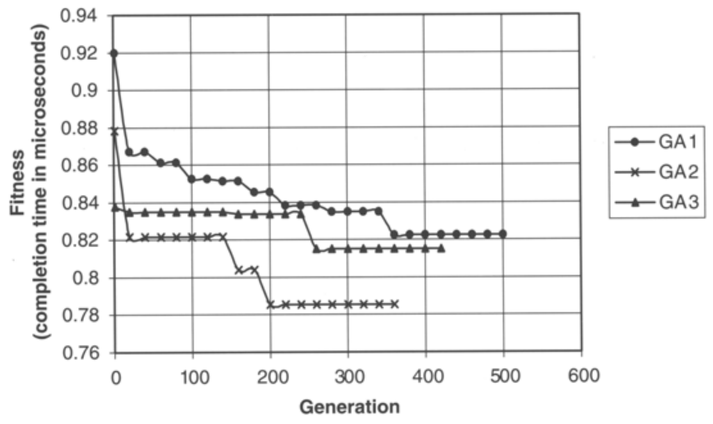


**FIG. 4.** Simulated communication completion time for the communication requirements for data redistribution after range processing and prior to processing in the pulse dimension. For GA 1, the crossover rate ($P_{xover}$) = 20% and the mutation rate ($P_{mut}$) = 4%. For GA 2, $P_{xover}$ = 50% and $P_{mut}$ = 10%. For GA 3, $P_{xover}$ = 90% and $P_{mut}$ = 50%.

antenna elements. Figure 4 is for the first communication phase (between processing in the range and pulse dimensions) and Fig. 5 is for the second communication phase (between processing in the pulse and element dimensions). The three curves in each

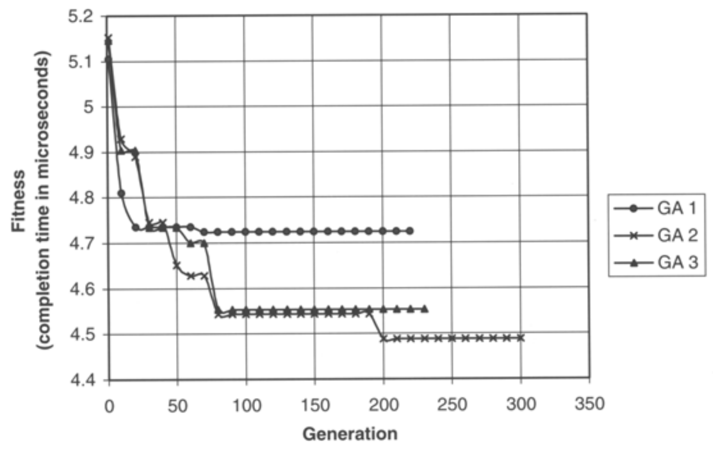


**FIG. 5.** Simulated communication completion time for the communication requirements for data redistribution after processing in the pulse dimension and prior to the final phase of processing. For GA 1, the crossover rate ($P_{xover}$) = 20% and the mutation rate ($P_{mut}$) = 4%. For GA 2, $P_{xover}$ = 50% and $P_{mut}$ = 10%. For GA 3, $P_{xover}$ = 90% and $P_{mut}$ = 50%.

figure correspond to different values for the crossover and mutation probability parameter settings. In the first GA scenario (GA 1), the crossover rate ($P_{\text{xover}}$) is 20% and the mutation rate ($P_{\text{mut}}$) is 4%. For GA 2, $P_{\text{xover}}$ is 50% and $P_{\text{mut}}$ is 10%. For GA 3, $P_{\text{xover}}$ is 90% and $P_{\text{mut}}$ is 50%.

Figures 4 and 5 illustrate that for a fixed mapping, the GA-based approach is capable of improving the scheduling of messages, thus providing improved overall performance for a given mapping. All three genetic-based scenarios improve the completion time for both communication phases. In each phase, GA 2 records the best schedule of messages (i.e., the smallest completion time). Extensive numerical studies involving different mappings, data cube sizes, and numbers of CNs have been conducted, but are not included here due to space limitations. A summary of some of these results will be given at the end of this section; for more details and analysis, the reader is referred to [8].

To better understand the interplay between mapping and scheduling, the following four process set mappings were considered for a 32 CN multicomputer: $[32 \times 1, 32 \times 1, 32 \times 1]$, $[16 \times 2, 16 \times 2, 16 \times 2]$, $[8 \times 4, 8 \times 4, 8 \times 4]$, and $[8 \times 4, 2 \times 16, 16 \times 2]$. Based on a data cube of size $32 \times 32 \times 32$, the best mapping objective function value (from all possible mappings) is the $[32 \times 1, 32 \times 1, 32 \times 1]$ mapping. The ranking of the $[16 \times 2, 16 \times 2, 16 \times 2]$ mapping was 13th, the $[8 \times 4, 8 \times 4, 8 \times 4]$ mapping ranked 31st, and the $[8 \times 4, 2 \times 16, 16 \times 2]$ mapping ranked 111th. For this example, the number of possible mappings was $6^3 = 216$.

The results illustrated in Fig. 6 compare the communication times associated with the best GA-schedules for each of the four mappings defined above (when applied to a $32 \times 32 \times 32$ data cube). The required communications between the range and pulse processing is denoted by Phase 1 label; Phase 2 label refers to the

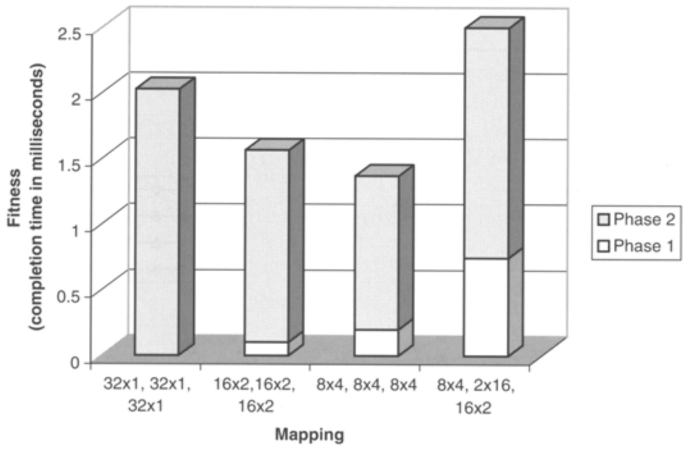


**FIG. 6.** Illustration comparing communication times associated with four different mappings schemes using the best GA-based schedules for each mapping. The STAP data cube is of size $32 \times 32 \times 32$ and there are 32 CNs in the multiprocessor system.

communications between processing in the pulse and element dimensions. Based on the mapping objective function, the $[32 \times 1, 32 \times 1, 32 \times 1]$ mapping receives the highest ranking. With this mapping, there is no required communication in the first phase of communication. Thus, the communications for this mapping is entirely in the second data transfer phase. And although the mapping objective function is minimal, the network is flooded with messages during the second communication phase resulting in high levels of network contention. In two of the other mappings, the data transfers are dispersed between two phases of traffic, thereby resulting in a smaller overall communication time. The poorest ranked mapping considered (i.e., $[8 \times 4, 2 \times 16, 16 \times 2]$), is indeed associated with the poorest overall communication time. Unlike the case presented here, other sets of mappings and data cube sizes are considered in [8] in which the rankings of the mappings remain consistent with the resulting rankings of the overall communication times associated with using the best GA-based schedules.

### *4.2. SWAP Analysis*

For many airborne radar systems, the goal of minimizing the total size, weight, and power (SWAP) of the computing platform can be critical. Until recently, very little research has focused on the effect data mapping and message scheduling have on overall SWAP requirements for a parallel computing platform.

Figure 7 shows the total computation time and three candidate communication completion times (total, for both communication phases) for a set of processors ranging from 4 to 32. For this figure, the STAP data cube consisted of 480 range bins, 64 pulses, and 16 channel elements. The computation time component was measured from an actual Mercury system executing the Real-Time STAP

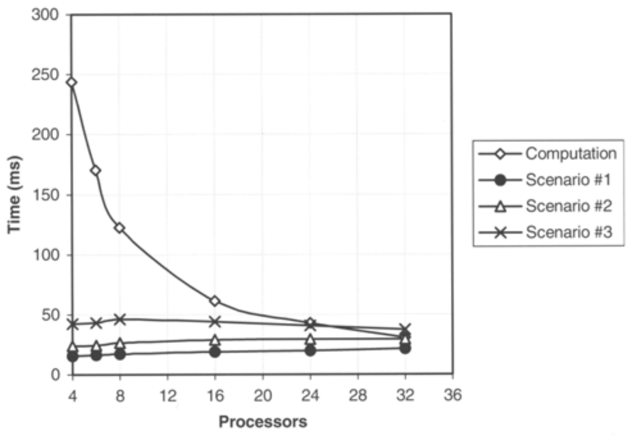


**FIG. 7.** Comparison of the computation time and communication times for three mapping/scheduling scenarios for a STAP data cube composed of 480 range bins, 64 pulses, and 16 channel elements.

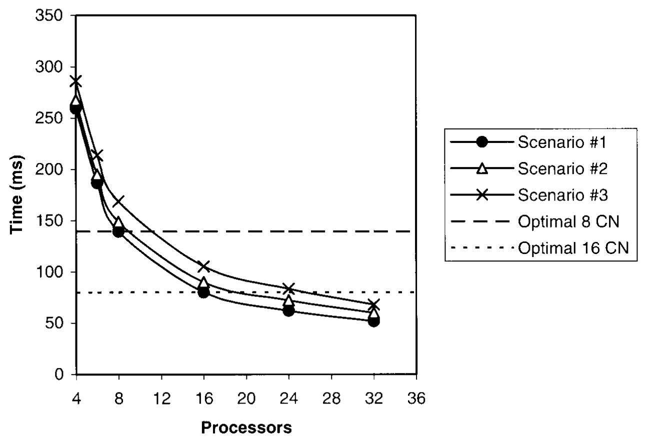


**FIG. 8.** Comparison of the overall completion times, including both computation and communication, for the three mapping/scheduling scenarios of Fig. 7. The optimal mapping/scheduling for 8-processor and 16-processor systems are indicated with dashed lines.

benchmark provided by Mitre [1]. Scenario 1 communication time corresponds to the best time reported by the GA optimization utilizing the best mapping for the given number of assigned processors. The communication completion times for a baseline scheduling of transfers given a typical mapping is illustrated by scenario 2, and communication scenario 3 consists of a typical mapping and a poor schedule. The illustration shows a distinctive variation in the communication scenarios' completion times. Additionally, note that as the number of processors increases, the computation time decreases.

To better visualize the affect data mapping and scheduling have on hardware requirements, the computation time can be added to each of the three communication scenarios shown in Fig. 7; the resulting completion times are depicted in Fig. 8. The intersection of optimal 8 processor dashed line and scenario 1 line represent the optimal mapping and scheduling for an 8 processor system. In this case, the completion time is around 140 ms; however, if scenario 3 was used the completion time would be closer to 170 ms per data cube. Obviously with the optimal mapping and scheduling (scenario 1), more data cubes per unit time can be processed; thus, in a unit of time more data cubes can be processed than with scenario 2 or 3. Note also from the figure that if a poor mapping/scheduling strategy (scenario 3) were utilized, then 11 or 12 processors would be required in order to match the performance of the optimally mapped (scenario 1) 8 processor system. This represents a potential reduction in hardware requirements of around 50% by utilizing the overall optimal mapping and scheduling scheme.

An optimal 16 processor system, which includes optimal data mapping and scheduling, can achieve the same performance as a 24 processor system with a poor mapping and scheduling. As a result, if a poor mapping and scheduling was selected for a 24 processor system, the same performance could be realized with an optimal 16 processor configuration. The overall SWAP requirements for a 16 processor system would be less than a 24 processor system. Therefore, by optimizing the mapping of data and the scheduling of messages the SWAP requirements can be reduced.

This example illustrates that by utilizing the optimal mapping and scheduling methodologies of Sections 2 and 3, hardware savings of 50% and more can be realized when compared to sub-optimal solutions to the mapping and scheduling problems. Because of limitations on the size of problems that could be executed/simulated, systems up to a size of only 32 processors were investigated. However, from the trends observed in overall completion times, it appears that even more significant savings in hardware/power requirements are realizable for STAP applications that require substantially larger systems having hundreds or even thousands of processors.

### *4.3. Summary of Numerical Studies*

The results recorded here for message scheduling demonstrate that off-line GA-based message scheduling can significantly improve the communication performance in a parallel system. In most cases, a moderate level of crossover (50%) and mutation rates (10%) yielded the best schedules. Although not included here, when compared to baseline and randomly generated schedules, the GA-based schedules are significantly superior—typically reducing communication times by between 20% and 50%, see [8] for details.

Interestingly, it is shown here that the best mapping—defined as a mapping that minimizes a mapping objective function—is not always the best choice in terms of minimizing overall communication time. In particular, as the number of CNs is increased, optimal mappings that require only one phase of communication generally report higher overall communication times than those good, but not optimal mappings that require two non-trivial phases of communication.

## 5. CONCLUSION

The optimization of mapping and scheduling, either independently or in combination, is critical to the performance of the STAP application for embedded parallel systems. For such systems, great significance is placed on minimizing overall execution time, which includes both computation and communication components. Such reductions in execution time also translate into improved hardware efficiency and thus reduced hardware requirements, which is often critical. The focus of this research is off-line optimization of data mapping and message schedules for a class of STAP algorithms to be implemented on a parallel embedded system.

An objective function is proposed and developed to measure the merit of a class of mappings for STAP for implementation on the Mercury multicomputer. The objective-function-based ranking provides a measure of the communication costs associated with a given mapping. A combination of the message size and required network resources for each message are key attributes used by the objective function.

A GA-based approach is proposed and developed for solving the message-scheduling problem for a given mapping. A GA is a population-based optimization model that uses selection and recombination operators to generate new sample points in the solution space. Reproduction opportunities are applied in such a way that those chromosomes representing a better solution to the targeted problem are given more opportunities to reproduce than poorer chromosomes. Each chromosome is associated with a fitness value, which in this case is the communication completion time of a schedule. The fitness of a candidate solution is calculated based on its simulated performance. The GA-based optimization is performed off-line, and the results of this optimization are static schedules for each CN in the parallel system. These static schedules can then be used within the online parallel STAP implementation. Through extensive numerical studies, it is shown that the off-line optimization approaches can yield mappings and schedules that greatly improve the on-line performance and reduce the hardware requirements of the parallel embedded system.

## APPENDIX A: OVERVIEW OF STAP

STAP algorithms have been developed to extract desired signals from potential target returns comprised of Doppler shifts associated with radar platform motion, clutter returns, and interference including jamming. In order to solve complex, large-scale, and real-time problems such as STAP, parallel processing has emerged as a key hardware technology. This appendix provides a brief overview of STAP methods; for a thorough theoretical treatment of STAP, the reader is referred to [6].

Current and future airborne radars must detect smaller targets in the presence of increasing interference such as clutter, jamming, noise, and platform motion. If the interference is localized in frequency and comes from a limited number of sources, targets can be detected by using adaptive spatial weighting of the data from each element of an antenna array [6]. By applying computed weights (determined in real time) to the data, the effects of interference can be reduced.

For an airborne radar platform that is in motion, the Doppler spread of the clutter returns is significant and the clutter characteristics are highly variable due to the changing ground terrain. In this type of an environment the weights must be adapted from the data in both the time and space dimensions. This general type of signal processing method, which is referred to as STAP, is an adaptive processing technique that simultaneously combines signals received from multiple elements of an antenna array (the spatial domain) and from multiple pulses (the temporal domain). The paragraphs to follow provide a general description of the computational complexity involved in implementing STAP algorithms. For a detailed theoretical foundation and analysis of these and other STAP algorithms, the reader is referred to [6].

Consider an $N$ element airborne radar array that transmits a coherent burst of $M$ pulses at a constant pulse repetition frequency (PRF) $f_r = 1/T_r$, where $T_r$ is the pulse repetition interval (PRI). The time interval over which the echo returns are collected is referred to as the coherent processing interval (CPI), and the resultant length of

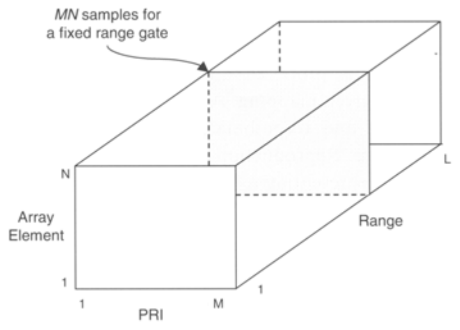


**FIG. A1.** The STAP CPI three-dimensional data cube (derived from [6]).

one CPI is $MT_r$. For each of the $M$ pulses, $L$ range samples are collected by each array element. With $M$ pulses and $N$ array channels, the return signal for one CPI is composed of $LMN$ complex signal samples. Because the signal returns are composed of $L$ range gates, $M$ pulses, and $N$ antenna array samples, the data may be represented by the 3D data set shown in Fig. A1. This $L \times M \times N$ data set will be referred to as a CPI data cube (or simply a data cube) [6].

Let $x_{nml}$ represent the $n$th array element and the $m$th pulse at the $l$th range sample time [6]. Next, define $x_{m,l}$ to represent an $N \times 1$ column vector, or a spatial snapshot, composed of the complex return signals from each array element for the $m$th pulse and the $l$th range. By combining all of the spatial snapshots at a given range of interest, an $N \times M$ matrix $X_l$ can be formed, where $X_l = (x_{1,l}, x_{2,l}, x_{3,l}, \ldots, x_{M,l})$. The shaded plane in Fig. A1, referred to as a range gate, represents the $X_l$ spatial snapshot at the $l$th range. To detect the presence of a target within a range gate, a space–time processor combines the data samples from the range gate to produce a scalar output, which is then passed through a threshold process for target detection.

The major components of a generic space–time processor are illustrated in Fig. A2. First, a set of rules called the training strategy is applied to the data to estimate the interference. The objective of the training strategy is to provide a good estimate of the interference at a given range gate. Because the interference is unknown, the training data is estimated data-adaptively from the STAP data cube.

The training data is used as input to the next component where the adaptive weight vector is calculated. The weight computation component is the most computationally intense portion of the space–time processor (and this component is the focus of attention in this paper). Weight computation itself is typically performed with three phases of processing: the first two phases involving linear filtering and the final phase requiring the solution of a set of linear equations [6]. After completing each phase of processing associated with weight computation, the data must be redistributed across the compute nodes of the machine, which represents the communication requirements of STAP. Thus, there are two primary phases of inter-processor data communication required: one between the first and second phases of processing and one between the second and third phases of processing.

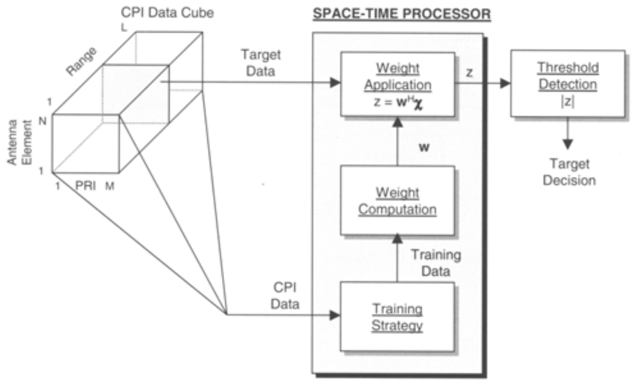


**FIG. A2.** Generic space–time adaptive processor (derived from [6]).

After all three phases of processing for weight computation are complete for a given STAP data cube, a new data cube is input into the parallel machine for processing.

The space–time processor produces a scalar output by computing the inner product of the weight vector and range gate of interest. The scalar output is compared to a threshold value to determine if a target is present at a specified angle and Doppler [6]. Because a potential target's angle and velocity are unknown, the space–time processor computes multiple weight vectors to cover a range of possible target angles, ranges, and velocities at which target detection is to be queried.

## APPENDIX B: PARALLEL STAP ON THE MERCURY SYSTEM

The weight computation component of any STAP algorithm is the most computationally intensive of the three components illustrated in the generic space–time processor of Fig. A2; it is the component that typically requires significant parallel computing resources in order to perform the required computations in a real-time setting. For this reason, our focus in this paper is on mapping and scheduling strategies for the weight computation component of processing. Furthermore, the terms "STAP" and "STAP computation" are understood to refer to the weight computation component unless noted otherwise.

Typical processing requirements of STAP range from 10 to 1000 giga-floating-point operations (Gflops), which can be met by multiprocessor systems composed of numerous interconnected compute elements (CEs) [3]. A CE contains a processor, local memory, and a connection to the network interconnecting the CEs. In the parallel STAP implementation assumed here, the network supports the three phases of inter-processor communication in which data must be exchanged among CEs.

The parallel computing system targeted for this work is the Mercury RACE® multicomputer. The RACE® multicomputer consists of a collection of compute

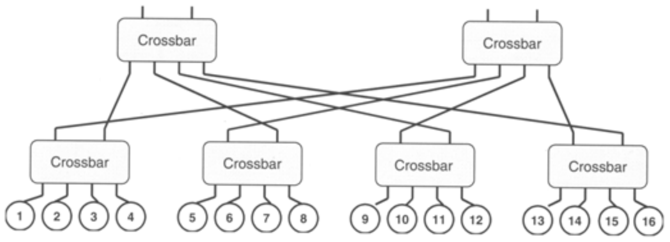


**FIG. B1.** Mercury RACE® fat-tree architecture configured with 16 CNs.

nodes (CNs), as well as various high-speed I/O devices, all interconnected by Mercury's RACEway® interconnection network [4]. A CN is a collection of one or more CEs, where the CEs within a CN are interconnected locally by a shared-memory. A high-level diagram of a 16-CN RACEway® topology is illustrated in Fig. B1. The interconnection network is configured in a fat-tree topology and is a circuit switched network. The RACEway® interconnection network is composed of a network of crossbar switches and provides high-speed data communication among the CNs. The Mercury multicomputer can support a heterogeneous collection of CN types (e.g., SHARC and PowerPCs processors), for more details refer to [7].

Achieving desired performance requirements for STAP implemented on a parallel embedded system like the Mercury multicomputer largely depends on two major issues. First is determining the best method for distributing the 3D STAP data cube across CNs of the multiprocessor system (i.e., the mapping strategy). Second is determining the scheduling of communications between phases of computation.

STAP computations contain three phases of processing, one for each dimension of the data cube (i.e., range, pulse, channel). During each phase of processing, the vectors along the dimension of interest are mapped as equally as possible among the CNs for processing in parallel. The framework assumed here for mapping is to partition the data cube into sub-cube bars. Each sub-cube bar is composed of partial data samples from two dimensions while preserving one whole dimension for processing. Fig. B2 shows an example of how sub-cube partitioning is applied to map a 3D data cube across 12 CNs. The sub-cube bar mapping approach was first described in [3].

During phases of data redistribution (i.e., communication) between computational phases, the number of required communications and the communication pattern among the CNs is dependant upon how the data cube is mapped to the CNs for each computational phase. For example, in Fig. B2(a) the mapping of sub-cube bars to CNs dictates that after range processing, CN 1 must transfer portions of it data sub-cube bar to CNs 2, 3, and 4. (Each of the other CNs, likewise, is required to send portions of their sub-cube bar to CNs on the same row.) The scheduling (i.e., ordering) of outgoing messages at each CN impacts the resulting communication time. For example, in Fig. B2(a) note CN 1 could order its outgoing messages according to one of $3! = 6$ permutations, i.e., $(2, 3, 4)$, $(3, 2, 4)$, etc. Similarly, a scheduling of outgoing messages must be defined for each CN. Improper schedule

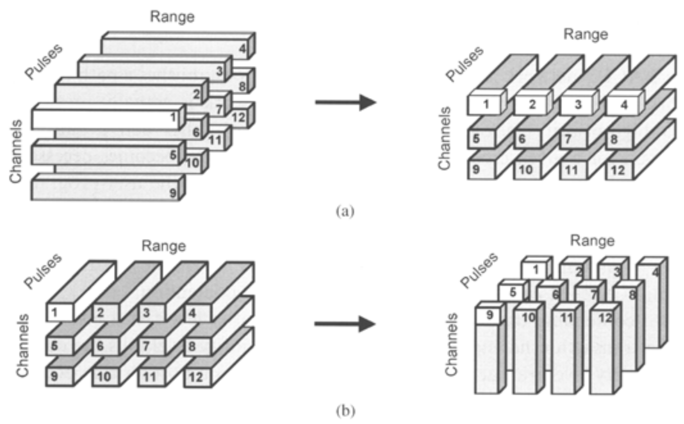


**FIG. B2.** Illustration of the sub-cube bar mapping technique for the case of 12 CNs. The mapping of the sub-cube bars to CNs defines the required data communications. (a) Example illustration of the communication requirements from CN 1 to the other CNs (2, 3, and 4) after completion of the range-dimensions processing and prior to Doppler (i.e., pulse-dimension) processing. (b) Example illustration of the communication requirements from CN 1 to other CNs (5 and 9) after the completion of Doppler processing and prior to the final phase of processing.

selection can result in excessive network contention and thereby increase the time to perform all communications between processing phases. Likewise, different mappings can be defined by considering all the combinations of process set dimensions whose product equals the number of processors. The example in Fig. B2 illustrates a $3 \times 4$ process set, but other dimensions are possible, i.e., $4 \times 3, 2 \times 6, 14 \times 12$, etc.

Once the mapping and scheduling is defined for each of the STAP computation and communication phases, respectively, the communication time for both of the communication phases can be evaluated. In this paper, evaluation of communication performance is made using a network simulator developed in [8].

## APPENDIX C: RACEway® NETWORK SIMULATOR

Each CN in the multicomputer interfaces the network through the RACE® network chip. The network chip is a crossbar with six bi-directional channels consisting of 32 parallel data lines and eight control leads [4]. Each crossbar transfers data synchronously at a clock rate of 40-MHz. Each channel is bi-directional but is only driven in one direction at a time at a rate of 160 MB/s [4]. Among the six ports comprising a RACE® crossbar, each switch can either interconnect any three port pairs, providing an aggregate bandwidth of 480 MB/s, or can cause data to be broadcast to all or a subset of the remaining five ports [4]. These crossbars are interconnected in a parent–child fashion to form a fat tree topology as shown in Fig. B1.

The RACE® network is circuit-switched, thus a CN establishes a path through the network prior to data transfer. The RACEway network is actually preemptive in that a high-priority message can suspend (preempt) other active paths. When arbitration for a given crossbar port, or sequence of ports, becomes necessary, the arbitration is carried out on the basis of a combination of the user-programmable packet priority and a fixed hardware priority at each crossbar based on the entry and exit ports at the given crossbar [4]. For this work, the user-programmable packet priority is assumed equivalent for all data packets, thus, the hardware priority arbitration rules at each crossbar are used to resolve contention.

If two contending transactions have different priority levels at a given crossbar, then the transaction having the highest hardware priority level kills the contending lower-priority level transaction. If a transaction requires a port already occupied by a lower-priority transaction, then the transmission of the lower-priority message is suspended (i.e., preempted) and the released port is then taken by the higher-priority transaction. The unsent data associated with the suspended transaction is re-packaged as a new message at the originating CN and begins the process of establishing a new path through the network. If two or more contending transactions have the same priority level, the first one started holds off any other contending transactions at the same level until the transmission of its data is completed.

The functionality of the RACEway® network has been encoded as a network simulator for use in this research. The details of the implementation and operation of the simulator are not given here, but can be found in [7, 8, 9]. Provided here is an overview of experimental studies performed that illustrate the accuracy of the simulator when compared with measured communication times taken from an actual Mercury multicomputer.

Two classes of communication patterns were employed to evaluate the accuracy of the simulator: simple test patterns and complex test patterns. Simple test patterns included the following three test categories: (I) single-source message tests; (II) two-source message tests (non-contending and contending paths); and (III) 3..$N$-source message tests (non-contending and contending paths). Complex communication patterns included the following categories: (IV) all-to-all personalized test and (V) randomized message queue communication test.

For the all-to-all personalized test, the outgoing message queues on each CN contained one message to each of the other CNs in the network. For the randomized message queue communication test (which closely resembles the communication pattern required by STAP) a random number of messages are sent from each of the CNs to randomly selected destinations. The outgoing message queues at each CN were randomly scheduled (i.e., ordered). For all test cases, identical communication patterns were executed on the actual Mercury computer and the network simulator.

A small subset of the tests performed are presented here. For each test, 50 independent trials were performed and averages computed for both the actual system and the software simulator. (Note that both the actual system and the simulator are non-deterministic.) The CNs are numbered left-to-right starting with 1 and incrementing by 1 for each successive CN. For instance, the first crossbar located

**TABLE C1**

**Comparison of Measured and Simulated Communication Times for Different Communication Patterns for Messages of Size 64 kB**

| Category | Description | Measured time (ms) | Simulated time (ms) | Percent error (%) |
|---|---|---|---|---|
| II (non-contending) | $2\to6$, $3\to7$ | 0.41119 | 0.40013 | 2.69 |
| II (contending) | $2\to4$, $3\to4$ | 0.84948 | 0.79608 | 6.29 |
| III (contending) | $2\to3$, $3\to4$ $4\to2$ | 1.19329 | 1.19097 | 0.19 |
| III (contending) | $2\to6$, $3\to6$ $6\to4$ | 1.28852 | 1.21279 | 5.88 |
| IV | $5\to\{6,7,8\}$ $6\to\{5,7,8\}$ $7\to\{5,6,8\}$ $8\to\{5,6,7\}$ | 3.67124 | 3.40914 | 7.14 |
| IV | All-to-all personalized communication involving CNs 2 through 8 | 9.52672 | 10.12816 | −6.31 |
| V | $2\to\{4,6,8\}$ $3\to\{5,7\}$ $4\to\{2,6,8\}$ $5\to\{7,3\}$ $6\to\{8,4,2\}$ $7\to\{5,3\}$ $8\to\{6,4,2\}$ | 3.85421 | 3.45185 | 5.89 |

at the bottom left of the fat-tree contains the first four CNs, numbered 1, 2, 3, and 4. The next four CNs (i.e., 5, 6, 7, and 8) are connected to the second (lowest-level) crossbar from the left, and so forth. Provided in Table C1 are representative results of the tests conducted. For all cases shown in the table, all transmitted messages were of size 64 kB. This study demonstrates the accuracy of the simulator, in that it typically has errors of around 5% or less. For a detailed discussion of these and other tests, the reader is referred to [8].

## ACKNOWLEDGMENT

This work was supported by DARPA under Contract F30602-97-2-0297.

## REFERENCES

1. K. C. Cain, J. A. Torres, and R. T. Williams, "Real-Time Space–Time Adaptive Processing Benchmark," Mitre Technical Report: MTR 96B0000021, Mitre, Center for Air Force C3 Systems, Bedford, MA, February 1997.
2. M. Gen and R. Cheng, "Genetic Algorithms and Engineering Design," Wiley, New York, 1997.
3. M. F. Skalabrin and T. H. Einstein, STAP processing on a multicomputer: distribution of 3-D data sets and processor allocation for optimum interprocessor communication, *in* "Proceedings of the Adaptive Sensor Array Processing (ASAP) Workshop," March 1996.
4. The RACE Multicomputer, "Hardware Theory of Operation: Processors, I/O Interface, and RACEway Interconnect," Vol. I, version 1.3.
5. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, *J. Parallel Distrib. Comput.* (Special Issue on Parallel Evolutionary Computing) **47** (November 1997), 8–22.
6. J. Ward, "Space–Time Adaptive Processing for Airborne Radar," Technical Report 1015, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA, 1994.
7. J. M. West, "Simulation of Communication Time for a Space–Time Adaptive Processing Algorithm Implemented on a Parallel Embedded System," Master's thesis, Computer Science, Texas Tech University, 1998.
8. J. M. West, "Processor Allocation, Message Scheduling, and Algorithm Selection for Parallel Space–Time Adaptive Processing," Dissertation, Computer Science, Texas Tech University, 2000.
9. J. M. West and J. K. Antonio, Simulation of the communication time for a space–time adaptive processing algorithm on a parallel embedded system, *in* "Proceedings of the International Workshop on Embedded HPC Systems and Applications (EHPC '98)" (J. Rolim, Ed.), Lecture Notes in Computer Science, Vol. 1388: Parallel and Distributed Processing, pp. 979–986, IEEE Computer Society, Orlando, FL, April 1998.

JACK M. WEST received the B.S., M.S., and Ph.D. in computer science from the Texas Tech University, Lubbock, Texas, in 1995, 1998, and 2000, respectively. After graduation, he was involved in post-doctoral work at the University of Oklahoma in the area of embedded high-performance systems. He is currently a software developer with RiskMetrics Group.

JOHN K. ANTONIO received the B.S., M.S., and Ph.D. from the Texas A&M University, College Station, Texas, in 1984, 1986, and 1989, respectively. He is currently professor and director of the School of Computer Science at the University of Oklahoma. Before joining the University of Oklahoma, he was with the Department of Computer Science at Texas Tech University and the School of Electrical and Computer Engineering at Purdue University. He is a member of the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honorary societies and is a senior member of the IEEE Computer Society. Dr. Antonio's current research interests include embedded high performance computing, reconfigurable computing, parallel and distributed computing, and cluster computing.

**Appendix F:** Timothy Osmulski, Jeffrey T. Muehring, Brian Veale, Jack M. West, Hongping Li, Sirirut Vanichayobon, Seok-Hyun Ko, John K. Antonio, and Sudarshan K. Dhall, "A Probabilistic Power Prediction Tool for the Xilinx 4000-Series FPGA," *Proceedings of the 5th International Workshop on Embedded/Distributed HPC Systems and Applications (EHPC 2000)*, in *Lecture Notes in Computer Science*, *IPDPS 2000 Workshops,* sponsor: IEEE Computer Society, Cancun, Mexico, May 2000, pp. 776-783.

# A Probabilistic Power Prediction Tool for the Xilinx 4000-Series FPGA

Timothy Osmulski, Jeffrey T. Muehring, Brian Veale, Jack M. West, Hongping Li, Sirirut Vanichayobon, Seok-Hyun Ko, John K. Antonio, and Sudarshan K. Dhall

School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019
Phone: (405) 325-7859
antonio@ou.edu

**Abstract.** The work described here introduces a practical and accurate tool for predicting power consumption for FPGA circuits. The utility of the tool is that it enables FPGA circuit designers to evaluate the power consumption of their designs without resorting to the laborious and expensive empirical approach of instrumenting an FPGA board/chip and taking actual power consumption measurements. Preliminary results of the tool presented here indicate that an error of less than 5% is usually achieved when compared with actual physical measurements of power consumption.

## 1 Introduction and Background

Reconfigurable computing devices, such as field programmable gate arrays (FPGAs), have become a popular choice for the implementation of custom computing systems. For special purpose computing environments, reconfigurable devices can offer a cost-effective and more flexible alternative than the use of application specific integrated circuits (ASICs). They are especially cost-effective compared to ASICs when only a few copies of the chip(s) are needed [1]. Another major advantage of FPGAs over ASICs is that they can be reconfigured to change their functionality while still resident in the system, which allows hardware designs to be changed as easily as software and dynamically reconfigured to perform different functions at different times [6].

Often a device's performance (i.e., speed) is a main design consideration; however, power consumption is of growing concern as the logic density and speed of ICs increase. Some research has been undertaken in the area of power consumption in CMOS (complimentary metal-oxide semiconductor) devices, e.g., see [4, 5]. However, most of this past work assumes design and implementation based on the use of standard (basic cell) VLSI techniques, which is typically not a valid assumption for application circuits designed for implementation on an FPGA.

## 2 Overview of the Tool

A probabilistic power prediction tool for the Xilinx 4000-series FPGA is overviewed in this section. The tool, which is implemented in Java, takes as input two files: (1) a *configuration file* associated with an FPGA design and (2) a *pin file* that characterizes the signal activities of the input data pins to the FPGA. The configuration file defines how each CLB (configurable logic block) is programmed and defines signal connections among the programmed CLBs. The configuration file is an ASCII file that is generated using a Xilinx M1 Foundation Series utility called *ncdread*. The pin file is also an ASCII file, but is generated by the user. It contains a listing of pins that are associated with the input data for the configured FPGA circuit. For each pin number listed, probabilistic parameters are provided which characterize the signal activity for that pin.

Based on the two input files, the tool propagates the probabilistic information associated with the pins through a model of the FPGA configuration and calculates the activity of every internal signal associated with the configuration [1]. The activity of an internal signal $s$, denoted $a_s$, is a value between zero and one and represents the signal's relative frequency with respect to the frequency of the system clock, $f$. Thus, the average frequency of signal $s$ is given by $a_s f$.

Computing the activities of the internal signals represents the bulk of computations performed by the tool [1]. Given the probabilistic parameters for all input signals of a configured CLB, the probabilistic parameters of that CLB's output signals are determined using a well-defined mathematical transformation [2]. Thus, the probabilistic information for the pin signals is transformed as it passes through the configured logic defined by the configuration file. However, the probabilistic parameters of some CLB inputs may not be initially known because they are not directly connected to pin signals, but instead are connected to the output of another CLB for which the output probabilistic parameters have not yet been computed (i.e., there is a feedback loop). For this reason, the tool applies an iterative approach to update the values for unknown signal parameters. The iteration process continues until convergence is reached, which means that the determined signal parameters are consistent based on the mathematical transformation that relates input and output signal parameter values, for every CLB.

The average power dissipation due to a signal $s$ is modeled by ½ $C_{d(s)}V^2 a_s f$, where $d(s)$ is the Manhattan distance the signal $s$ spans across the array of CLBs, $C_{d(s)}$ is the equivalent capacitance seen by the signal $s$, and $V$ is the voltage level of the FPGA device. The overall power consumption of the configured device is the sum of the power dissipated by all signals. For an $N$ x $N$ array of CLBs, Manhattan signal distances can range from 0 to $2N$. Therefore, the values of $2N$ + 1 equivalent capacitance values must be known, in general, to calculate the overall power consumption. Letting $S$ denote the set of all internal signals for a given configuration, the overall power consumption of the FPGA is given by:

$$\begin{aligned} P_{\text{avg}} &= \sum_{s\in S} \frac{1}{2} C_{d(s)} V^2 a_s f \\ &= \frac{1}{2} V^2 f \sum_{s\in S} C_{d(s)} a_s . \end{aligned} \tag{1}$$

The values of the activities (i.e., the $a_s$'s) are dependent upon the parameter values of the pin signals defined in the pin file. Thus, although a given configuration file defines the set $S$ of internal signals present, the parameter values in the pin file impact the activity values of these internal signals.

## 3 Calibration of the Tool

Let $S_i$ denote the set of signals of length $i$, i.e., $S_i = \{s \in S \mid d(s) = i\}$. So, the set of signals $S$ can be partitioned into $2N + 1$ subsets based on the length associated with each signal. Using this partitioning, Eq. 1 can be expressed as follows:

$$P_{avg} = \frac{1}{2}V^2 f\left( C_0 \sum_{s \in S_0} a_s + C_1 \sum_{s \in S_1} a_s + \cdots + C_{2N} \sum_{s \in S_{2N}} a_s \right) \tag{2}$$

To determine the values of the tool's capacitance parameters, actual power consumption measurements are taken from an instrumented FPGA using different configuration files and pin input parameters. Specifically, $2N + 1$ distinct measurements are made and equated to the above equation using the activity values (i.e., the $a_s$'s) computed by the tool. For the $j$-th design/data set combination, let $P_j$ denote the measured power and let $A_{j,k}$ denote the aggregate activity of all signals of length $k$. The resulting set of equations is then solved to determine the $2N + 1$ unknown capacitance parameter values:

$$\frac{1}{2}V^2 f \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,2N} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,2N} \\ \vdots & & \ddots & \vdots \\ A_{2N,0} & A_{2N,1} & & A_{2N,2N} \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{2N} \end{pmatrix} = \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{2N} \end{pmatrix} \tag{3}$$

Solving the above equation for the vector of unknown capacitance values is how the tool is calibrated.

## 4 Power Measurements

For this study, a total of 70 power measurements were made using 5 different configuration files and 14 different data sets. Descriptions of these configuration files and data sets are given in Tables 1 and 2, respectively. All of the configuration files listed in Table 1 each take a total of 32-bits of data as input. The first three configurations (fp_mult, fp_add, int_mult) each take two 16-bit operands on each clock cycle, and the last two (serial_fir and parallel_fir) each take one 32-bit complex operand on each clock cycle. The 32 bits of input data are numbered as 0 through 31 in Table 2, and two key parameters are used to characterize these bits: an *activity factor, a* and a *probability factor, p*. The activity factor of an input bit is a value between zero and one and represents the signal's relative frequency with respect to the frequency of the system clock, *f*. The probability factor of a bit represents the fraction of time that the bit has a value of one.

Fig. 1 shows plots of the measured power for all combinations of the configuration files and data sets described in Tables 1 and 2. For all cases, the clock was run at $f =$ 30 MHz. With the exception of the fp_mult configuration file, the most active data set file (number 6) is associated with the highest power consumption. Also, the least active data set file (number 5) is associated with the lowest power consumption across all configuration files. There is somewhat of a correlation between the number of components utilized by each configuration and the power consumption; however, note that even though the serial_fir implementation is slightly larger than parallel_fir, it consumes less power. This is likely due to the fact that the parallel_fir design requires a high fan-out (and thus high routing capacitance) to drive the parallel multipliers.

**Table 1.** Characteristics of the configuration files.

| **Configuration File Name** | **Description** | **Component Utilization of Xilinx 4036xla** |
|---|---|---|
| fp_mult | Custom 16-bit floating point multiplier with 11-bit mantissa, 4-bit exponent, and a sign bit [3]. | 368 |
| fp_add | Custom 16-bit floating point adder with 11-bit mantissa, 4-bit exponent, and a sign bit [3]. | 339 |
| int_mult | 16-bit integer array multiplier; produces 32-bit product [3]. | 509 |
| serial_fir | FIR filter implementation using a serial-multiply with a parallel reduction add tree. Input data is 32-bit integer complex. Constant coefficient multipliers and adders from core generator. | 1060 |
| parallel_fir | FIR filter implementation using a parallel-multiply with a series of delayed adders. Input data is 32-bit integer complex. Constant coefficient multipliers and adders from core generator. | 1055 |

**Table 2.** Characteristics of the data sets.

| **Data Set Number** | **Description** |
|---|---|
| 1 | Pins 0 through 15 $\Rightarrow p = 0.0$ and $\alpha = 0.0$.<br>Pins 16 through 31 $\Rightarrow p = 0.5$ and $\alpha = 1.0$ |
| 2 | Pins 0 through 15 $\Rightarrow p = 0.0$ and $\alpha = 0.0$<br>Pins 16 through 31 $\Rightarrow p = 0.75$ and $\alpha = 0.4$ |
| 3 | Pins 0 through 15 $\Rightarrow p = 0.25$ and $\alpha = 0.45$<br>Pins 16 through 31 $\Rightarrow p = 0.0$ and $\alpha = 0.0$ |
| 4 | Pins 0 through 15 $\Rightarrow p = 0.\ 5$ and $\alpha = 1.0$<br>Pins 16 through 31 $\Rightarrow p = 0.0$ and $\alpha = 0.0$ |
| 5 | Pins 0 through 31 $\Rightarrow p = 0.0$ and $\alpha = 0.0$ |
| 6 | Pins 0 through 31 $\Rightarrow p = 0.5$ and $\alpha = 1.0$ |
| 7 | Even numbered pins $\Rightarrow p = 0.0$ and $\alpha = 0.0$<br>Odd numbered pins $\Rightarrow p = 0.5$ and $\alpha = 1.0$ |
| 8 | Even numbered pins $\Rightarrow p = 0.3$ and $\alpha = 0.5$<br>Odd numbered pins $\Rightarrow p = 0.7$ and $\alpha = 0.5$ |
| 9 | Even numbered pins $\Rightarrow p = 0.5$ and $\alpha = 1.0$<br>Odd numbered pins $\Rightarrow p = 0.0$ and $\alpha = 0.0$ |
| 10 | Even numbered pins $\Rightarrow p = 0.8$ and $\alpha = 0.1$<br>Odd numbered pins $\Rightarrow p = 0.2$ and $\alpha = 0.15$ |
| 11 | For all pins, $p$ and $\alpha$ selected at random (different from data set 12). |
| 12 | For all pins, $p$ and $\alpha$ selected at random (different from data set 11). |
| 13 | Pins 0 through 2, $p = 0.1$ and $\alpha = 0.1$<br>Pins 3 through 5, $p = 0.2$ and $\alpha = 0.2$, etc.,<br>$p$'s continue to increase in steps of 0.1 and $\alpha$'s increase to 0.5 in steps of 0.1 and then decrease back down to 0.0. |
| 14 | Pin 0, $p = 0.1$ and $\alpha = 0.2$<br>Pin 1, $p = 0.2$ and $\alpha = 0.4$<br>Pin 2, $p = 0.3$ and $\alpha = 0.6$, etc.,<br>$p$'s continue to increase to 1.0 in steps of 0.1 (and then decrease) and $\alpha$'s increase to 1.0 in steps of 0.2 (and then decrease). |

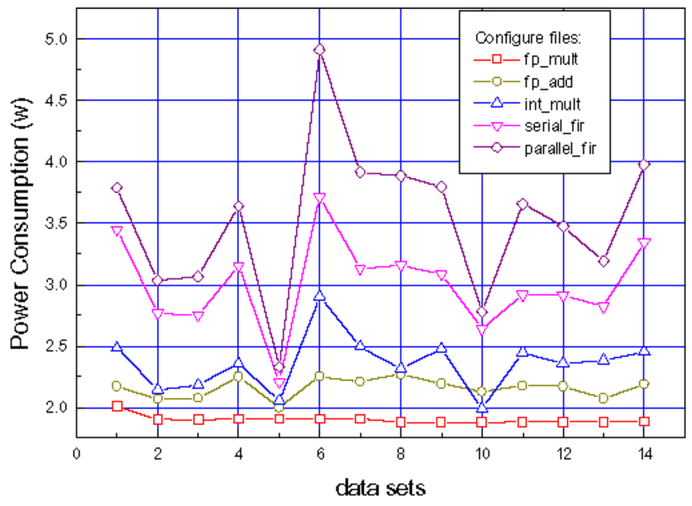


**Fig. 1.** Measured power consumption for the configuration files and data sets described in Tables 1 and 2.

## 5 Experimental Evaluation of the Tool

Because 73 values are used to model all of the internal capacitances of the device used in this study, at least three more measurement scenarios are required to calibrate all capacitance values (by solving the complete set of linear equations defined by Eq. 3). Fortunately, however, we were able to calibrate a subset of capacitance values by considering the power consumption of the two FIR filters (serial_fir and parallel_fir). This was because there turned out to be a total of only 28 non-zero entries for the rows of the matrix of Eq. 3, corresponding to aggregate activities for the two FIR filter designs.

Fig. 2 shows the measured power consumption curve along with 29 different prediction curves generated by the tool for the serial FIR filter design. One of the prediction curves corresponds to predicted values based on using all 28 measured values to calibrate the tool's capacitance values (this curve is labeled "all" in the legend of the figure). This curve naturally has excellent accuracy; predicted power consumption values match measured values nearly perfectly.[1] The remaining 28 prediction curves are associated with capacitance values determined by using all but one of the measured data values to calibrate the tool (the data set not used is indicated in the legend of the figure). For each of these curves, the data set not used in the

[1] The reason the predicted values do not match measured values exactly is because the equations used to determine capacitance values did not have full rank, and thus a least-squares solution was determined.

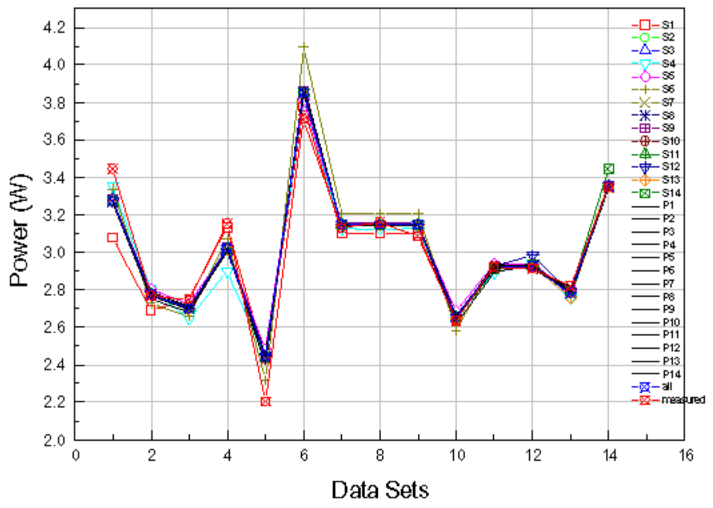


**Fig. 2.** Measured and predicted power consumption curves using various calibration scenarios for the serial FIR filter implementation.

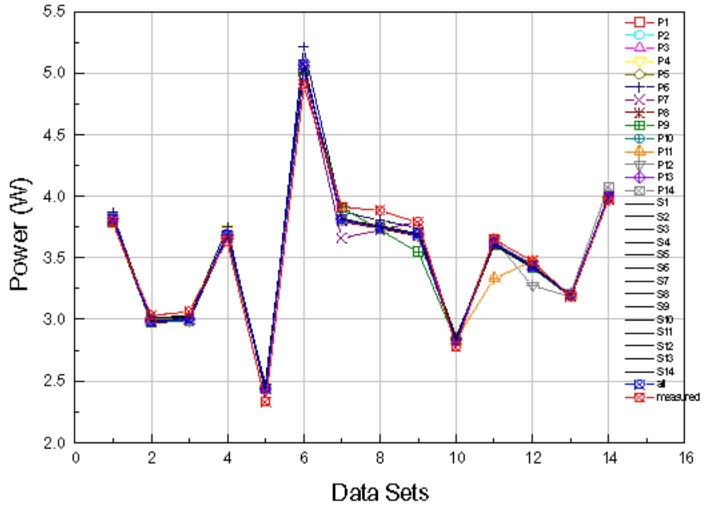


**Fig. 3.** Measured and predicted power consumption curves using various calibration scenarios for the parallel FIR filter implementation.

calibration of the tool's capacitance values generally associates with the highest error in the predicted value for that data point. For example, note that when data set number six for the serial FIR (labeled S6 in the figure's legend) was not used in the calibration process, the resulting prediction for that value was highest (around 10% error). When data sets associated with the parallel FIR design were not included, the prediction curves did not change, thus those curves are all drawn as solid lines with no symbols. Fig. 3 shows the same type of results as Fig. 2, except for the parallel FIR instead of the serial FIR.

## 6 Summary

To summarize the results for both filter designs, when all 28 sets of measurements are used to calibrate the tool, the maximum error in predicted versus measured power is typically less than about 5%. With one data set removed, the maximum error increases to about 10%, and the predicted value with this highest error is typically associated with the data set not used in calibrating the tool. This level of error is acceptable for most design environments, and represents a considerable accomplishment in the area of power prediction for FPGA circuits. Thus, these preliminary results indicate that the tool is able to adequately predict power consumption (i.e., for data sets not used in calibrating the tool). By using more data sets to calibrate the tool in the future, it is expected that even greater prediction accuracy and robustness will be achieved.

## Acknowledgements

This work was supported by DARPA under contract no. F30602-97-2-0297. Special thanks go to Annapolis Micro Systems, Inc. for their support and for providing the instrumented FPGA board that was used to take power measurements.

## References

1. T. Osmulski, *Implementation and Evaluation of a Power Prediction Model for Field Programmable Gate Array*, Master's Thesis, Computer Science, Texas Tech University, 1998.
2. K. P. Parker and E. J. McClusky, "Probabilistic Treatment of General Combinatorial Networks," *IEEE Trans. Computers*, vol. C-24, pp. 668-670, June 1975.
3. B. Veale, *Study of Power Consumption for High-Performance Reconfigurable Computing Architectures*, Master's Thesis, Computer Science, Texas Tech University, 1999.
4. T. L. Chou, K. Roy, and S. Prasad, "Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching," *Proc. IEEE Int'l Conf. Comput. Aided Design*, pp. 300-303, Nov. 1994.
5. A. Nannarelli and T. Yang, "Low-Power Divider," *IEEE Trans. Computers*, Vol. 48, No. 1, Jan. 1999, pp. 2-14.
6. *Xilinx XC4000E and XC4000X Series Field Programmable Gate arrays, Product Specification*, Xilinx Inc., v1.5, http://www.xilinx.com/partinfo/databook.htm#xc4000, 1999.

**Appendix G:** Hongping Li, John K. Antonio, and Sudarshan K. Dhall, “Fast and Precise Power Prediction for Combinational Circuits,” *Proceedings of the IEEE Symposium on VLSI*, sponsor: IEEE, Tampa, FL, Feb 2003, pp. 254-259.

# Fast and Precise Power Prediction for Combinational Circuits

Hongping Li, John K. Antonio, and Sudarshan K. Dhall
*School of Computer Science, University of Oklahoma*
*200 Felgar Street, Norman OK 73019-6151*
*hongping@ou.edu, antonio@ou.edu, sdhall@ou.edu*

## Abstract

*The power consumed by a combinational circuit is dictated by the switching activities of all signals associated with the circuit. An analytical approach is proposed for calculating signal activities for combinational circuits. The approach is based on a Markov chain signal model, and directly accounts for correlations present among the signals. The accuracy of the approach is verified by comparing signal activity values calculated using the proposed approach with corresponding values produced through simulation studies. It is also demonstrated that the proposed approach is computationally efficient.*

## 1. Introduction

Power consumption of integrated circuits (ICs) is of growing concern as more electronic devices are being deployed in mobile and portable applications, e.g., PDAs, mobile telephones, and other battery-powered electronic devices. As the functionality of such devices increases, so does the complexity and sophistication of the underlying circuits. More complexity and faster clock rates generally translate into higher power consumption for a given hardware implementation technology. Because battery technology has not improved at the same rate as IC technology, there is strong motivation to design circuits that are as power efficient as possible to extend battery life for portable devices.

The focus of this paper is the development of an analytical tool for predicting power consumption of combinational circuits. This tool, which is implemented in software, can be utilized during the design phase to give the designer quick and accurate predictions of power consumption for a given circuit design.

Several similar and related approaches to this problem have been proposed in the past, including simulation-based [1] and analytical approaches [2, 3, 4]. A good survey of past approaches can be found in [5]. Generally, simulation-based approaches achieve high accuracy but require long execution times; in contrast, the analytical approaches are faster but are generally less accurate. In this paper a new analytical approach is proposed that achieves fast execution time and accuracy that is comparable with simulation-based methods. As explained below, the particular focus is on power consumption of circuits implemented in CMOS, but the proposed approach may be applicable for other technologies as well.

Power consumption in a CMOS circuit is primarily due to three types of current flow: leakage current, switching transient current, and load capacitance charging current [9]. The last is the dominant component of power consumption in CMOS devices, and is strongly dependent on signal switching activity.

Let $S$ denote the set of all signals associated with a circuit. For each $s \in S$, let $C(s)$ denote the capacitive load associated with signal $s$. Also, let $\alpha(s)$ denote the activity of signal $s$, which has a value between zero and one, and represents the signal's normalized average frequency relative to the frequency of a system clock, $f$. Thus, $f\alpha(s)$ gives the average frequency of signal $s$. Based on these assumptions and notation, the average power for a CMOS circuit operating at a voltage level of $V$ can be expressed as [4, 5]:

$$\text{Power}_{\text{avg}} = \frac{1}{2}V^2 f \sum_{s \in S} C(s)\alpha(s)\,. \qquad (1)$$

The problem addressed in this paper is to determine the activity of all signals of a combinational circuit given an appropriate probabilistic model for the primary input signals that drive the circuit. The signal model proposed in this paper is based on a Markov chain. The signal activity is easily computed from the parameters associated with the proposed signal model. In the proposed approach, signals with known Markov chain representations are propagated through the circuit to produce Markov chain representations for the outputs of all gates in the circuit. Accuracy of the approach is verified by comparing signal activities produced by the proposed method with corresponding activities produced through simulation studies. When compared with other related approaches, a key aspect of the proposed approach is that correlations present among the signals due to re-convergent fan-out are accounted for directly.

## 2. Previous Related Approaches

### 2.1 Signal Probability Calculation

In [2], probabilistic signal modeling for combinational circuits was first introduced. Each signal is modeled with a single probabilistic parameter that defines the probability of a signal having a logical value of one. For signal $x$, the probability that $x$ has logic value 1 is defined by $P(x) = P(x = 1)$. Two algorithms for calculating signal probabilities are introduced in [2]. These approaches require that a Boolean function expression associated with each signal be derived in terms of the primary inputs. Because the number of terms in these expressions can grow exponentially with the number of inputs, the complexity of these approaches can be prohibitive for practical circuits.

A computationally efficient algorithm for calculating signal probabilities is introduced in [7], named "Algorithm 1," which operates by propagating probability values through the gates of circuit, thereby drastically reducing the size of the Boolean functions that must be evaluated. This algorithm is simple and fast – it has a linear complexity in the number of gates – but is not accurate for all classes of circuits.

Another algorithm is proposed in [7] called the Weighted Averaging Algorithm (WAA), which generally achieves better accuracy than Algorithm 1 and has a comparable time complexity. However, the WAA still does not always produce correct values.

A method for accounting for signal probability correlations was developed in [6] named the correlation coefficient method (CCM). By using this approach, the probability of the output of a two-input gate can be more accurately calculated, given the probabilities of the two inputs and an associated correlation factor associated with the two signals. In this algorithm, the correlation factor can also be calculated analytically by means of a set of basic propagation rules.

### 2.2. Signal Activity Calculation

The above-described approaches of [2], [6], and [7] are concerned with determining the probabilities of signal values, not the probabilities of signal transitions, i.e., activities, which are necessary for estimating power consumption, refer to Eq. 1. An early approach for estimating signal activities was developed in [3], in which signals of a circuit are modeled to be mutually independent strict-sense-stationary (SSS) mean-ergodic 0-1 processes. Under these assumptions, the activity of a signal $y$ from a circuit with $n$-primary inputs can be expressed as

$$\alpha(y) = \sum_{i=1}^{n} P\left(\frac{\partial y}{\partial x_i}\right)\alpha(x_i) \qquad (2)$$

where $\partial y / \partial x_i$ is the Boolean difference of function $y$ with respect to $x_i$ and is defined by

$$\frac{\partial y}{\partial x_i} = y|_{x_i=1} \oplus y|_{x_i=0} = y(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \oplus y(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n). \qquad (3)$$

Intuitively, the Boolean difference $\partial y / \partial x_i$ defines whether a transition of signal $x_i$ will cause a transition in output signal $y$. Specifically, if the Boolean difference function evaluates to one, then a transition of signal $x_i$ causes a transition in $y$. So, the probability of the Boolean difference function, $P\left(\frac{\partial y}{\partial x_i}\right)$, defines the probability that a change in $y$ will occur given that there is a change in $x_i$.

The calculation of the probability of the Boolean difference terms, i.e., $P\left(\frac{\partial y}{\partial x_i}\right)$, this calculation can be complicated for large and complex circuits. In [3], the calculation of these terms is accomplished by first representing the nodes of the circuit with a binary decision diagram (BDD) [3, 5]. In practice, the BDD approach often achieves linear or near linear time complexity; however, in the worst case the complexity can grow exponentially with the number of gates.

It is noted in [4] that Eq. 2, i.e., the approach described in [3], fails to consider the effect of simultaneous switching of gate inputs. Each Boolean difference term associated with Eq. 2 describes an input-switching event in which exactly one of the inputs makes a transition. Thus, Eq. 2 does not account for events involving simultaneous switching of two or more of the input signals. The concept of the generalized Boolean difference was introduced in [4] to account for simultaneous switching, and is denoted as follows:

$$\frac{\partial y^k \mid b_{i_1}, b_{i_2}, \ldots, b_{i_k}}{\partial x_{i_1} \partial x_{i_2} \ldots \partial x_{i_k}} = (y \mid x_{i_1} = b_{i_1}, x_{i_2} = b_{i_2}, \ldots x_{i_k} = b_{i_k}) \oplus (y \mid x_{i_1} = \overline{b_{i_1}}, x_{i_2} = \overline{b_{i_2}}, \ldots, x_{i_k} = \overline{b_{i_k}}), \qquad (4)$$

where $k$ is a positive integer, $x_{i_j}$, $j = 1, 2, \ldots, k$, are distinct mutually independent primary inputs of $y$, and

$b_{i_j}$ are binary values of 0 or 1. Note that if the generalized Boolean difference evaluates to one, then the simultaneous transitions of signals $(x_{i_1}, x_{i_2}, ..., x_{i_k})$ from $(b_{i_1}, b_{i_2}, ..., b_{i_k})$ to $(\overline{b_{i_1}}, \overline{b_{i_2}}, ..., \overline{b_{i_k}})$ or from $(\overline{b_{i_1}}, \overline{b_{i_2}}, ..., \overline{b_{i_k}})$ to $(b_{i_1}, b_{i_2}, ..., b_{i_k})$ will cause a transition at $y$.

Eq. 2 is adapted in [4] using the generalized Boolean difference concept to account for simultaneous switching, resulting in:

$$\alpha(y) = \sum_{i=1}^{n} Pc\left(\frac{\partial y}{\partial x_i}\right)\left(\alpha(x_i)\prod_{\substack{j\neq i\\ 1\le j\le n}}[1-\alpha(x_j)]\right) \tag{5}$$
$$+\frac{1}{2}\left\{\sum\left[Pc\left(\frac{\partial^2 y|_{00}}{\partial x_i \partial x_j}\right)+Pc\left(\frac{\partial^2 y|_{01}}{\partial x_i \partial x_j}\right)\right]\left(\alpha(x_i)\alpha(x_j)\prod_{l\in\{1,2,...,n\}-\{i,j\}}[1-\alpha(x_l)]\right)\right\}+...$$
$$+\frac{1}{2^{n-1}}\left[Pc\left(\frac{\partial^n y|_{00..0}}{\partial x_1 \partial x_2 ... \partial x_n}\right)+Pc\left(\frac{\partial^n y|_{00..1}}{\partial x_1 \partial x_2 ... \partial x_n}\right)+...+Pc\left(\frac{\partial^n y|_{01..1}}{\partial x_1 \partial x_2 ... \partial x_n}\right)\right]\left(\prod_{l=1}^{n}\alpha(x_l)\right)$$

where $Pc\left(\frac{\partial y}{\partial x_i}\right)$, $Pc\left(\frac{\partial^2 y|_{00}}{\partial x_i \partial x_j}\right)$, ... , $Pc\left(\frac{\partial^n y|_{01...1}}{\partial x_1 \partial x_2 \cdots \partial x_n}\right)$ are conditional probabilities of the generalized Boolean differences under the condition that only the indicated inputs simultaneously switch, and the rest do not. Details on how to calculate these conditional probabilities can be found in [4].

The approaches of [3] and [4] can have high computational complexities because the number of terms in the underlying equations/transformations can grow exponentially with the number of primary inputs to the circuit. Trade-offs between computational complexity and accuracy are possible relative to the evaluation of Eq. 2 and Eq. 5 (associated with [3] and [4], respectively). Instead of deriving a signal's logic function in terms of the circuit's primary inputs, the parameters to the immediate inputs of the signal's logic gate can be used. This type of "gate-by-gate" technique will generally introduce error because it does not account for correlations present among the internal signals that drive the gates within the circuit.

## 3. Markov Chain Signal Model

### 3.1. Preliminaries

In this section we introduce a signal model that is based on a Markov chain having three event parameters. It is shown that the proposed Markov chain model is equivalent to the two-parameter probability/activity signal model of [3] and [4]. The advantage of modeling signals with Markov chains is that it makes it possible to compute correlations between signals related to both probability and activity.

The approach derived here can be viewed as a generalization of the approach in [6]. Instead of tracking a correlation factor for the single probability parameter model, transformations for correlation factors associated with the three parameters of the Markov model are derived. This ultimately leads to a fast and accurate "gate-by-gate" algorithm for calculating signal probabilities and activities.

As illustrated in Figure 1, the proposed Markov chain signal model has three event parameters for signal $A$. The event denoted by $A$ represents the signal being in state 1, and $A_1$ and $A_2$ represent the events that there is a transition from state 0 to 1 and from state 1 to 0, respectively. Note that the probability of event $A$ is denoted by $P(A)$, and is equivalent to the signal probability defined in the previous section.

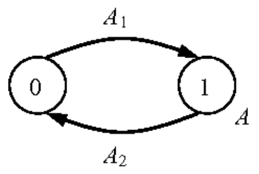


**Figure 1. Proposed Markov chain signal model.**

For notational convenience and clarity, we will denote the value of $P(A)$ as $p_A$ (for the value of the probability of signal $A$) and the value of the activity $\alpha(A)$ as $\alpha_A$ (for the value of the activity of signal $A$) throughout the rest of the paper. Using these notations and applying basic properties of Markov chains along with the definition of signal activity, the following expressions can be derived for $P(A)$, $P(A_1)$ and $P(A_2)$:

$$P(A) = p_A, \quad P(A_1) = \frac{\alpha_A}{2(1-p_A)}, \quad P(A_2) = \frac{\alpha_A}{2p_A}. \tag{6}$$

Thus, if the values of both the probability and activity parameters of a signal are known (i.e., $p_A$ and $\alpha_A$), then the probabilities of the three events associated with the proposed Markov model for the signal are completely determined. Likewise, knowing the probability values of the three parameters of the Markov model fully determines the probability and activity parameters of the signal.

In order to define correlations between two signals modeled with Markov chains, some basic definitions are needed. Let $A$ and $B$ denote two events and let $P(AB)$ denote the probability of both $A$ and $B$ occurring. From basic probability theory [8], $P(AB) = P(A/B)P(B)$, where $P(A/B)$ represents the probability of $A$ given $B$. Also, the correlation coefficient of two events $A$ and $B$ is defined as

$$\rho_{AB} = \frac{\sigma_{AB}}{\sigma_A \sigma_B} \quad (7)$$

where $\sigma_{AB}$ is the covariance and $\sigma_A$ and $\sigma_B$ are the positive square roots of the variances of $A$ and $B$. It can be shown that

$$\rho_{AB} = \frac{P(AB) - P(A)P(B)}{\sqrt{P(A)(1-P(A))}\sqrt{P(B)(1-P(B))}}. \quad (8)$$

In order to simplify later derivations, it is convenient to define the correlation factor $C_{AB}$ of two events $A$ and $B$ as

$$C_{AB} = \frac{P(AB)}{P(A)P(B)} = \frac{P(A/B)}{P(A)} = \frac{P(B/A)}{P(B)}. \quad (9)$$

By applying Eq. 8 to Eq. 9, the following relationship can be derived:

$$\rho_{AB} = \frac{P(A)}{\sqrt{P(A)(1-P(A))}} \frac{P(B)}{\sqrt{P(B)(1-P(B))}} (C_{AB} - 1). \quad (10)$$

Thus, $C_{AB}$ is related to $\rho_{AB}$ through scaling and shifting. The value of $\rho_{AB}$, by definition [8], is a real number in the interval [-1, 1]; therefore, according to Eq. 10, $C_{AB}$ takes on real non-negative values. Also, $\rho_{AB} = 0$ corresponds to $C_{AB} = 1$, and indicates that the events $A$ and $B$ are mutually independent. Similarly, $\rho_{AB} < 0$ (i.e., $A$ and $B$ are negatively correlated) corresponds to $0 \le C_{AB} < 1$, and $\rho_{AB} > 0$ (i.e., $A$ and $B$ are positively correlated) corresponds to $C_{AB} > 1$.

## 3.2. Markov Chain Model for Basic Logic Gates

The focus in this subsection is on deriving the Markov chain model for the output of a basic logic gate in which the Markov chain models of the input signals are known. The simple case of a NOT gate is considered first followed by the analysis of two-input basic logic gates.

For a NOT gate with input $A$, the Boolean output function is given by $Y = \overline{A}$. From Figure 1, it is clear that the Markov model for $Y$ is given by

$$P(Y) = 1 - P(A), \quad P(Y_1) = P(A_2), \; P(Y_2) = P(A_1). \quad (11)$$

Consider now the case of a two-input basic logic gate. Assuming the Markov chain models of inputs $A$ and $B$ are known, the objective is to derive the Markov chain model for output signal $Y$. A key to deriving the Markov chain model for signal $Y$ is to represent the state transition diagram associated with the gate's two inputs, as shown in Figure 2. The four states in the figure correspond to the four input combinations for the two inputs. The first digit of each state label corresponds to the value of $A$, and the second to the value of $B$, e.g., the state labeled "01" corresponds to $A = 0$ and $B = 1$. Although not labeled on the figure, the directed edges represent transition events. To illustrate the notation to label transition events, "00→10" will be used to represent the event that input signal $A$ transitions from 0 to 1 and signal $B$ stays in state 0.

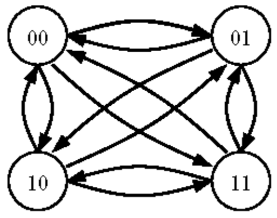


**Figure 2. State transition diagram for a two-input gate.**

The known parameters of the Markov chain models for signals $A$ and $B$ are given by $P(A)$, $P(A_1)$, $P(A_2)$, $P(B)$, $P(B_1)$, and $P(B_2)$. Also assumed to be known are the correlation factors for pairs of events associated with the Markov chain models for the inputs. From Eq. 9 note that $P(AB) = P(A)P(B)C_{AB}$, where $C_{AB}$ is the correlation factor associated with events $A$ and $B$. Similarly, the correlation factor $C_{A_1B_2}$ enables the calculation of $P(A_1B_2)$ using the fact that $P(A_1B_2) = P(A_1)P(B_2)C_{A_1B_2}$. Recall from Eq. 10 that independent events correspond to a correlation factor of unity. Given the Markov chain models for signals $A$ and $B$ (and the corresponding correlation factors) it is possible to derive the probability associated with every event shown in the state transition diagram of Figure 2. A complete tabulation of these expressions can be found in [11].

Deriving a Markov chain model for the output ($Y$) of a two-input gate depends on the particular function of the gate. To illustrate, consider the specific example of an AND gate, i.e., $Y = AB$. For an AND gate, the output takes on logic value 1 if and only if both inputs are 1. Thus,

$$P(Y) = P(11) = p_A p_B C_{AB}. \quad (12)$$

The event $Y_1$ is associated with three events from Figure 2, namely: 00→11, 01→11, and 10→11. Thus, equality can be established as follows:

$$P(\overline{Y})P(Y_1) = P(00)P(00 \to 11) + P(01)P(00 \to 11) + P(01)P(00 \to 11). \quad (13)$$

Solving Eq. 13 for $P(Y_1)$ and using Eqs. 6 results in the following expression:

$$P(Y_1) = (\frac{1}{2}\lambda_A p_B \alpha_A + \frac{1}{2}\lambda_B p_A \alpha_B)/(1 - p_A p_B C_{AB}) - \left[\frac{1}{4}\left(\lambda_A C_{A_1B_2} + \lambda_B C_{A_2B_1} - \lambda C_{A_1B_1}\right)\alpha_A \alpha_B /(1 - p_A p_B C_{AB})\right] \quad (14)$$

The parameters $\lambda$, $\lambda_A$, and $\lambda_B$ are simply functions of probabilities and correlations factors and are used for notational convenience; expressions for these parameters can be found in [11]. Derivation for $P(Y_2)$ follows in a similar fashion and can be expressed as

$$P(Y_2) = \frac{\alpha_1}{2p_1} + \frac{\alpha_2}{2p_2} - \frac{\alpha_1}{2p_1}\frac{\alpha_2}{2p_2}C_{A_2B_2}. \quad (15)$$

Derivations of $P(Y)$, $P(Y_1)$, and $P(Y_2)$ for two-input OR and XOR gates are included in [11]. Methods for calculating/propagating correlation factors through basic elements of a circuit are also included in [11].

## 4. Markov Chain Propagation Algorithm

This section describes a proposed Markov Chain Propagation (MCP) algorithm for determining the Markov chain models for all signals of a given combinational circuit. The Markov chain signal model of Section 3 is employed, and it is assumed that the parameters of the model are known for the circuit's primary inputs. The overall approach is to propagate signal information associated with the Markov chain model through the circuit in a "gate-by-gate" fashion. Recall that once the Markov chain model is determined for all signals, the signal activities and circuit power estimate are determined using Eq. 6 and Eq. 1, respectively. It is assumed that the given circuit is specified at the level of basic logic gates.

**MCP Algorithm**

**Step 1: Represent the given combinational circuit as a directed acyclic graph (DAG).**
*Vertices of the DAG correspond to basic gates and edges represent signals. Two extra vertices (a source and a sink) are included in the DAG to accommodate the primary inputs and outputs of the circuit. An example of how to represent a circuit with the DAG model is illustrated by Figures 3(a) and 3(b).*

**Step 2: Perform a topological sort [10] on the DAG to obtain an ordering of the gates.**
*See Figure 3(c).*

**Step 3: Transform to two-input basic logic gates.**
*As shown in Figure 3(d), replace all basic gates having more than two inputs with an equivalent sequence of two-input basic gates.*

**Step 4 Partition the circuit into levels.**
*As shown in Figure 3(e), levels are defined at the input and output of each basic gate. Note that there is at most one gate between any two consecutive levels.*

**Step 5: Successively apply propagation rules at each level.**
*Apply the propagation rules from [11] for calculating the parameters of the Markov model for the basic gate outputs and the associated correlation factors.*

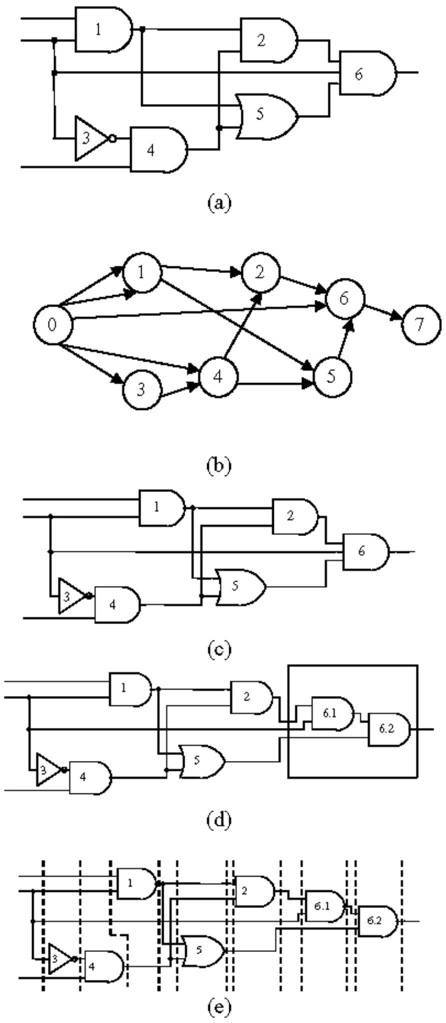


**Figure 3. Illustration of the MCP Algorithm.**

For a circuit with $M$ signals and $N$ gates, the time complexity of the MCP Algorithm can be shown to be $\Theta(M + N)$. Due to space limitations, a detailed derivation of the time complexity of the MCP Algorithm is not included here, but can be found in [11].

## 5. Experimental Results

The MCP Algorithm has been implemented and evaluated using several test circuits. To verify the accuracy of the results produced by the MCP algorithm, PSpice® circuit simulations were performed on the same test circuits. In the simulation studies, time-series realizations from the assumed Markov chain model for each primary input were used to drive the circuit simulation. Estimates of signal probabilities were derived from the simulations by counting the fraction of time each signal took on a value of unity. Estimates of signal activities were derived from the simulations by counting signal transitions.

The MCP Algorithm was also evaluated using a circuit named C432 from the ISCAS-85 Benchmark Set. For this circuit there are a total of 145 distinct signals, not including the primary inputs. (Note that there are a total of 432 physical signals, which includes fan-out signals.) Table 1 shows the distribution of absolute differences and relative percentage errors between activity values computed by the MCP Algorithm and those derived through simulation. Other circuits were also tested and these results also indicate the accuracy of the MCP Algorithm.

**Table 1. Accuracy for the MCP Algorithm.**

| Absolute Diff. Range | Number of Signals |
|---|---|
| [0, 0.01] | 70 |
| (0.01, 0.02] | 35 |
| (0.02, 0.03] | 19 |
| (0.03, 0.04] | 10 |
| (0.04, 0.05] | 10 |
| (0.05, 0.06] | 1 |
| (0.06, 1] | 0 |

| Relative Error Range (%) | Number of Signals |
|---|---|
| [0, 1] | 43 |
| (1, 2] | 41 |
| (2, 5] | 31 |
| (5, 10] | 25 |
| (10, 20] | 3 |
| (20, 50] | 2 |
| >50 | 0 |

## 6. Summary and Future Work

The problem of determining the activities of all signals of a combinational circuit is addressed in this paper. A new signal model is proposed based on a Markov chain. Signal activity is easily computed from the parameters associated with the proposed signal model. In the proposed approach, signals with known Markov chain representations are propagated through the circuit to produce a Markov chain representation for the output of each gate in the circuit. Accuracy of the approach is verified by comparing signal activities produced by the proposed method with corresponding activities produced through simulation studies. These initial testing results will be extended in future work by testing more and larger circuits.

## Acknowledgments

This research was supported by DARPA under Contract F30602-97-2-0297. The authors would like to thank Dr. S. Lakshmivarahan for his contributions to this work.

## References

[1] R. Burch, F. N. Najm, P. Yang, and T. Trick, "A Monte Carlo Approach for Power Estimation", *IEEE Trans. VLSI Systems*, Vol. 1, No. 1, Mar. 1993, pp. 63-71.

[2] K. P. Parker and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks," *IEEE Trans. Computers*, Vol. C-24, No. 6, June 1975, pp. 668-670.

[3] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 2, Feb. 1993, pp. 310-323.

[4] T.-L. Chou and K. Roy, "Estimation of Activity for Static and Domino CMOS Circuits Considering Signal Correlations and Simultaneous Switching," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 10, Oct. 1996, pp 1257-1265.

[5] F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits," *IEEE Trans. on VLSI Systems*, Vol. 2, No. 4, Dec. 1994, pp. 446-455.

[6] S. Ercolani, M. Favalli, M. Damiani, P. Olovo, and B. Ricco, "Estimate of Signal Probability in Combinational Logic Networks," *Proc. IEEE European Test Conference*, April 1989, pp. 132-138.

[7] B. Krishnamurthy and I. G. Tollis, "Improved Techniques for Estimating Signal Probabilities," *IEEE Trans. Computers*, Vol. 38, No. 7, July 1989, pp. 1041-1045.

[8] J. B. Thomas, *An Introduction to Applied Probability and Random Processes*, Krieger Publishing, Huntington, NY, 1981.

[9] M. J. M. Smith, *Application-Specific Integrated Circuits*, Addison Wesley, Reading, MA, 1997.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill New York, NY, 2001.

[11] H. Li, J. K. Antonio, and S. K. Dhall, "Fast and Precise Power Prediction for Combinational Circuits," Technical Report No. CS-TR-02-001, School of Computer Science, University of Oklahoma, Nov. 2002.

**Appendix H:** Nikhil D. Gupta, John K. Antonio, and Jack M. West, “Reconfigurable Computing for Space-Time Adaptive Processing” *Proceedings of the Sixth Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 1998, pp. 335-336.

# Reconfigurable Computing for Space-Time Adaptive Processing

*Nikhil D. Gupta*, *John K. Antonio*, and *Jack M. West*
Department of Computer Science
Box 43104
Texas Tech University
Lubbock, TX 79409-3104 USA
Tel: 806-742-1659
{gupta, antonio, west}@ttu.edu

## 1. Introduction

Space-time adaptive processing (STAP) refers to a class of signal processing techniques used to process returns of an antenna array radar system [4]. STAP algorithms are designed to extract desired target signals from returns comprised of Doppler shifts, ground clutter, and jamming interference. STAP simultaneously and adaptively combines the signals received on multiple elements of an antenna array – the spatial domain – and from multiple pulse repetition periods – the temporal domain.

The output of STAP is a weighted sum of multiple returns, where the weights for each return in the sum are calculated adaptively and in real-time. The most computationally intensive portion of most STAP approaches is the calculation of the adaptive weight values. Calculation of the weights involves solving a set of linear equations based on an estimate of the covariance matrix associated with the radar return data.

Existing approaches for STAP typically rely on the use of multiple digital signal processors (DSPs) or general-purpose processors (GPPs) to calculate the adaptive weights. These approaches are often based on solving multiple sets of linear equations and require the calculation of numerous vector inner products. This paper proposes the use of FPGAs as vector co-processors capable of performing inner product calculation.

Two different "inner-product co-processor" designs are introduced for use with the host DSP or GPP. The first has a multiply-and-accumulate structure, and the second uses a reduction-style tree structure having two multipliers and an adder.

## 2. STAP Weight Calculation

### 2.1 Basic Formulation

The STAP algorithm assumed here is known as $K^{\text{th}}$-order Doppler factored STAP, which is classified as a partially adaptive technique. Due to the space limitation, it will not be possible to fully explain this algorithm. Instead, the focus here will be on the necessary notation and core calculations required to determine the values of the adaptive weights. For more information on STAP, the reader is referred to [1, 4].

Determining the values for the $n$-vector of adaptive weights, denoted by $\vec{w}$, involves solving a system of linear equations of the form:

$$\Psi \vec{w} = \vec{s}\ , \qquad (1)$$

where $\vec{s}$ is a known $n$-vector called the steering vector and $\Psi$ is an estimate of the covariance matrix, which is determined based on the sampled radar returns. $\Psi$ is derived based on space-time data matrix $X$, which is an $n \times N$ matrix defined by: $X = [\vec{x}_1\ \vec{x}_2\ \ldots \vec{x}_N]$. Based on this the definition, $\Psi$ is given by:

$$\Psi = \frac{1}{N} X X^H \qquad (2)$$

### 2.2 QR-Decomposition and Conjugate Gradient

The QR-decomposition approach is a direct approach for solving a system of linear equations. The QR approach always gives an exact solution and the complexity of the algorithm is fixed. It involves performing a QR-decomposition on the matrix $X^T$, the result of which is an $N \times N$ orthogonal matrix $Q$ and an $n \times N$ upper triangular matrix $R$ such that $X = QR$. The final result is obtained by forward and backward substitution. For more details the reader is referred to [1].

The conjugate gradient approach is an iterative method that provides a general means for solving a system of linear equations [2]. For the system of equations given in Eq. (1), it is based on the idea of minimizing the following function:

$$f(\vec{w}) = \frac{1}{2}\vec{w}^T \Psi \vec{w} - s\vec{w}\ . \qquad (3)$$

The function $f$ is minimized when its gradient is zero, i.e., $\nabla f = \Psi\vec{w} - \vec{s} = 0$, which corresponds to the solution to the original system of linear equations. The very repetitive and regular numerical structure of the conjugate gradient update equations makes it a prime candidate for implementation on an FPGA system.

Numerical studies were conducted using Matlab implementations of the QR-decomposition and CG methods on actual STAP data collected by the Multi-Channel Airborne Radar Measurement (MCARM) system of Rome Lab [3]. Further details of this study can be found in [6].

## 3. Inner-Product FPGA Co-Processor

Each of the two methods outlined above requires calculating a number of inner products. Given enough resources, all the inner products could be done in parallel on FPGAs. Because the available system has only two FPGAs [5], the computations was divided among the host processor and the FPGA board. The two schemes that were implemented are outlined below. For both schemes, the data vectors are assumed to be in block-floating-point format [9]. Additionally, the

This work was supported by DARPA under contract number F30602-97-2-0297

multiplier implementation is based on discussion in [7] and the adder unit uses 4-bit carry-look-ahead adders [8] in each stage of the adder pipe.

### 3.1 Multiply-and-Accumulate Implementation

In the first implementation shown in Figure 1, the FPGA is configured to perform the multiply-and-accumulate operations on the input vectors. The implementation consists of a multiply unit and an accumulator, which is composed of a normalization unit and an adder. The normalization unit shifts the binary point of the

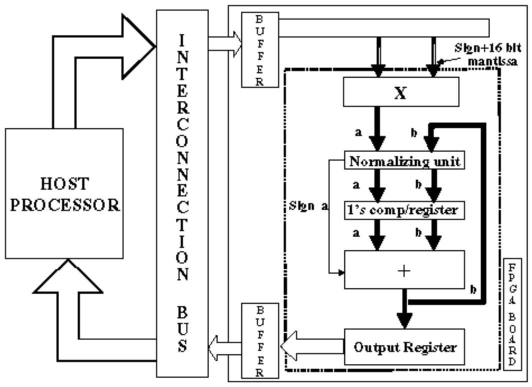


Figure 1: Block diagram implementation of the multiply-and-accumulate unit on WildOne FPGA board.

mantissa and makes a compensating adjustment to the exponent prior to the addition. The output of the adder is fed back and accumulated with the next product term.

The single cycle multiply-and-accumulate is achieved by pipelining each unit of the implementation. This unit reads in two operands and performs two operations per cycle. Thus, the unit reduces two *N*-vectors to a constant number of partial sums equal to the number of stages in the accumulator pipe. The implementation allocates approximately 88% of the configurable logic blocks (CLBs) on the Xilinx 4028EX FPGA. The implementation can be clocked at 40MHz, thus giving a throughput of 80 million block-floating-point operations per second.

### 3.2 Multiply-and-Add Implementation

Figure 2 illustrates the second implementation that performs an inner product, i.e., a multiply-and-add operation on the two input vectors. The design incorporates two 16-bit multiply units and an adder. By using this approach, two multiplies can be performed in parallel, and afterwards, the adder computes the sum of the two products.

A challenge associated with this implementation is that four 16-bit input operands, i.e., 64 bits, are required per computation cycle. Unfortunately, the data-path to the FPGA board is only 36-bits wide. The solution to this problem involves clocking the input state machine at twice the frequency of the multiply-and-add state machine, and registering the first two operands for one input state machine clock cycle.

The multiply-and-add unit reads in four operands and performs three block-floating-point operations per cycle. Thus,

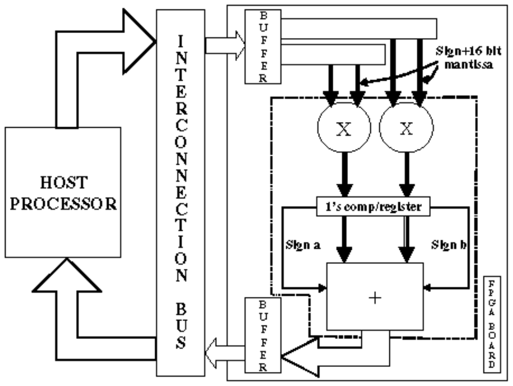


Figure 2: Block diagram implementation of the multiply-and-add unit on WildOne FPGA board.

the two input *N*-vectors are reduced to an *N*/2-vector of partial sums. This implementation, however, involves an additional *N*/2 addition operations to obtain the inner product result. For this implementation, approximately 99% of the available CLBs on the Xilinx 4028EX FPGA are required. In summary, for a fixed clock rate, the second design can provide a higher throughput, but requires more computation from the host (to perform the final summation of the partial sums).

## 4. References

[1] K. C. Cain, J. A. Torres, and R. T. Williams, "*Real-Time Space-Time Adaptive Processing Benchmark*", Mitre TR: MTR 96B0000021, Mitre, Bedford, MA, February 1997.

[2] D. G. Luenberger, *Linear and Nonlinear Programming*, Second Edition, Addison-Wesley, Reading, MA, 1984.

[3] Real-Time MCARM Data Sets, http://sunrise.oc.rl.af.mil.

[4] J. Ward, *Space-Time Adaptive Processing for Airborne Radar*, Technical Report 1015, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA, 1994.

[5] *Wild-One Hardware Reference Manual 11927-0000 Revision 0.1*, Annapolis Micro Systems Inc., MD, 1997.

[6] Nikhil D. Gupta, *Reconfigurable Computing for Space-Time Adaptive Processing*, MS Thesis Proposal, TTU, http://hpcl.cs.ttu.edu/darpa/reconfigurable/, 1997.

[7] T.T. Do, H.Kropp, P. Pirsch, "Implementation of Pipelined Multipliers on Xilinx FPGAs," *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, Springer Verlag, September1997

[8] M. Morris Mano, *Digital Logic and Computer Design*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1992

[9] W. W. Smith, J. M. Smith, *Handbook of Real-Time Fast Fourier Transforms*, IEEE Press, New York, NY, 1995

**Appendix I:** Jeffrey T. Muehring and John K. Antonio, "Minimizing Power Consumption using Signal Activity Transformations for Very Deep FPGA Pipelines," *Proceedings of the Military and Aerospace Applications for Programmable Devices and Technologies Conference (MAPLD 2000),* sponsors: NASA and Johns Hopkins University/Applied Physics Laboratory, Laurel, MD, Sep. 2000.

# Minimizing Power Consumption using Signal Activity Transformations for Very Deep FPGA Pipelines

Jeffrey T. Muehring and John K. Antonio

School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019-6151
Phone: 405-325-7859
Fax: 405-325-4044
antonio@ou.edu

*Submitted to:*

*2000 MAPLD Conference*

*The 3rd Annual Military and Aerospace Applications of Programmable Devices and Technologies International Conference*

June 2000

## Abstract

The density of commercially available FPGAs (field programmable gate arrays) has increased dramatically in the past several years. Because of this trend, it has become possible to move more and more computations associated with high-performance embedded applications from DSPs (digital signal processors) onto FPGA devices. The potential advantages of utilizing FPGAs instead of DSPs, in this context, include reductions in the utilization of size, weight, and power (SWAP) required to implement the computational platform. These types of reductions are of particular interest for military applications such as synthetic aperture radar (SAR) processing, which is often performed on a small unmanned aerial vehicle (UAV) having limited available power for the on-board computational platform.

Two major contributions are presented in this paper. First, it is shown that the core computations from the SAR application, including both the range compression and azimuth processing phases, can be structured as a single deep computational pipeline that can be implemented directly onto an array of FPGAs. Past results for high-throughput SAR processing typically assume the computations are to be mapped onto a distributed memory multiprocessor system in which a subset of the available processing elements (PEs) are assigned to perform range compression and the remaining processors perform azimuth processing. In this type of traditional approach, a number of compressed (processed) range vectors are sent from the range PEs to the azimuth PEs where they are buffered in memory. After a prescribed number of compressed range vectors are present in the memory space of the azimuth processors, azimuth processing commences. Because of the significant intermediate buffer storage required by this approach, and the

associated placing and fetching of data in this memory space by the range and azimuth PEs, respectively, this type of SAR implementation is generally not thought to be a "purely streaming" application. However, as is presented in this paper, these computations (both phases) can in fact be structured as a single computational pipeline, which can be directly mapped onto an array of FPGAs. In the proposed approach, no intermediate memory buffer is required between the two phases of computation. Instead, within the structure of the computational pipeline are long segments of delay elements that effectively provide the intermediate storage associated with the more traditional approach. One potential advantage of the proposed approach is that data need not be continuously stored and then fetched from a separate memory module by PE (which, incidentally, can require significant power consumption). Instead, the data streams continuously through a long computational pipeline. Within this pipeline are the taps of the FIR (finite impulse response) implementations of both the range and azimuth processing, interspersed with long segments of delay elements. Although the resulting pipeline may be thousands of stages long for practical values of SAR parameters, it is a viable approach because end-to-end latencies on the order of 1 millisecond are typically acceptable, provided that the required throughput is achieved.

The second contribution presented in this paper demonstrates how signal activity parameters of incoming data can be transformed, before the data are processed by a computational pipeline, as a means of reducing overall power consumption. The key to understanding this approach is the realization that the activity levels of the input signals to the computational pipeline dictate its level of power consumption. The activity of a given input signal (i.e., bit) is defined as the fraction of times that the signal transitions relative to the system clock. It has been demonstrated that increasing the signal activities of input data to a pipelined circuit implemented on an FPGA also increases the power consumption of the circuit [1]. In the present paper we illustrate how the activities of the input data can be transformed (pre-processed) so that the resulting (transformed) signals that are input into the computational pipeline have activity values that are well-matched with the pipelined circuit in terms of minimizing consumed power. At the end of the computational pipeline, an inverse transformation is applied to the output values to convert them back to their proper (and meaningful) representation. This approach is based on two fundamental assumptions: (1) that the power consumption of the computational pipeline is significantly higher than that of the computational structures implemented to perform the transform and inverse transformation of the data and (2) that the computations performed within the computational pipeline are linear and time invariant.

The final version of this paper will contain further details related to the two contributions outlined here. Details on the structure and depth of the computational pipeline associated with the proposed SAR processing approach will be provided. This approach, in terms of estimated power consumption, will be compared with more traditional approaches that make use of a multicomputer architecture. Also presented will be measurements and estimates of overall power savings possible by using the proposed signal activity transformation approach.

## Reference

[1] Timothy Osmulski, Jeffrey T. Muehring, Brian Veale, Jack M. West, Hongping Li, Sirirut Vanichayobon, Seok-Hyun Ko, John K. Antonio, and Sudarshan K. Dhall, "A Probabilistic Power Prediction Tool for the Xilinx 4000-Series FPGA," *Proceedings of the 5th International Workshop on Embedded/Distributed HPC Systems and Applications (EHPC 2000)*, in *Lecture Notes in Computer Science*, sponsor: IEEE Computer Society, Cancun, Mexico, May 2000, pp. 776-783.

**Appendix J:** S. Vanichayobon, Sudarshan K. Dhall, S. Lakshmivarahan, and John K. Antonio, "Power-speed Trade-off in Parallel Prefix Circuits," *Proceedings of ITComm 2002, High-Performance Pervasive Computing Conference*, sponsor: SPIE, Boston, MA, July/Aug. 2002, pp. 109-120.

# Power-speed Trade-off in Parallel Prefix Circuits

S. Vanichayobon, Sudarshan K. Dhall, S. Lakshmivarahan, J. K. Antonio
School of Computer Science, University of Oklahoma, Norman, OK 73019.

## ABSTRACT

Optimizing area and speed in parallel prefix circuits have been considered important for long time. The issue of power consumption in these circuits, however, has not been addressed. The paper presents a comparative study of different parallel prefix circuits from the point of view of power-speed trade-off. The power consumption and the power-delay product of seven parallel prefix circuits were compared. A linear output capacitance assumption, combined with PSpice simulations, is used to investigate the power consumption in the parallel prefix circuits. The degrees of freedom studied include different parallel prefix algorithms and voltage scaling. The results show that the use of the linear output capacitance assumption provides results that are consistent with those obtained using PSpice simulations. The study can help identify parallel prefix algorithms with the desirable power consumption with a given throughput.

Keywords: Parallel prefix circuits, power, power-speed trade-off.

## 1. INTRODUCTION

The three most widely accepted metrics for measuring the quality of a circuit are its area, speed, and power consumption. Optimizing area and speed have been considered important for long time, but minimizing power consumption has been gaining prominence only recently [1, 5, 10]. One important reason for minimizing power consumption of a circuit is the proliferation of portable electronic systems, such as laptops, mobile phones and wireless devices, where maximizing battery life is important. Since it is desirable to minimize the size and weight of batteries in such devices, while increasing the time between battery recharges, finding methods of reducing power consumption has assumed considerable importance.

In this paper, we study power-speed trade-off for prefix circuits. The prefix circuits play an important role in many applications. It appears in a number of areas such as the carry-look-ahead adder, ranking, packing, radix sort, etc. [8]. Many new approaches for prefix circuits with the goal of optimizing depth (i.e., speed) and size (i.e., area) have been proposed [2, 6, 8, 9, 12]. As a result, performance in terms of the speed and area has improved. The issue of power consumption in these circuits, however, has not been addressed. Therefore, our goal is to make a comparative study of different prefix circuits from the point of view of power-speed trade-off in order to facilitate the design choices, specifications, and resource limitations. In this study, we use the power-delay product as a quality measure for the prefix circuits. The power-delay product is the product of the circuit's power consumption and propagation delay, which represents the energy consumed by the circuit per operation.

In this paper, we first propose power modeling of prefix circuits. Then, the analysis, combined with PSpice simulations [3], is used to investigate the power consumption in the prefix circuits considered. The simulations were carried out on voltage scaling. It is found that the divide-and-conquer prefix circuit, which is the fastest circuit, consumes the most power. Also according to PSpice simulations, the power-delay product of the LYD prefix circuit seems to be the best amongst the circuits considered while the power-delay product of the divide-and-conquer is the highest.

The rest of this paper is divided into five sections. Section 2 provides an overview of prefix circuits. Section 3 reviews the sources of power consumption in a CMOS circuit and presents strategies to estimate power consumption of the circuit. Section 4 focuses on modeling the power consumption of the prefix circuits studied here. Section 5 describes the analysis of power-speed trade-off of prefix circuits considered. Finally, Section 6 concludes the results of the paper.

## 2. PREFIX CIRCUITS – AN OVERVIEW

A ***prefix computation*** is the process of taking $N$ input values $x_1, x_2, \ldots, x_{N-1}, x_N$ and producing $N$ output values $y_1, y_2, \ldots, y_{N-1}, y_N$ such that $y_1 = x_1$, and

$$y_i = y_{i-1} \bullet x_i = x_1 \bullet x_2 \bullet \ldots \bullet x_{i-1} \bullet x_i, \qquad \text{for } 2 \le i \le N$$

where $\bullet$ is an associative binary operation. A prefix circuit with $N$ inputs can also be viewed as a layered directed acyclic graph with $N$ input nodes, $N$ output nodes, and at least $N$-1 operation nodes. An operation node is neither an input nor an output node. Figure 1 illustrates the layout and the components of a prefix circuit. The numbers along the left-hand side of the layout give the depth (level) of the operation nodes on the right.

The traditional metrics for measuring the performance of a prefix circuit include its size, depth, fan-in, and fan-out. The ***size*** of a prefix circuit, *size(N),* is the total number of operation nodes in the circuit. The ***depth*** of a prefix circuit, *depth(N),* is the length of the longest path measured in terms of the number of operations along the path in the circuit from its input nodes to its output nodes. The circuit depth is related to its computation time. In VLSI implementation, a circuit with smaller depth is generally faster than one with greater depth when the fan-out of most nodes in the two circuits is similar [14]. A prefix circuit is ***depth-optimal*** if the circuit has the smallest depth among all possible circuits. An $N$-input prefix circuit is ***(size, depth)-optimal*** if $size + depth = 2N - 2$ [12]. Every prefix circuits have size-depth trade-off property [6] – a reduction of the circuit depth is achieved at the cost of an increase in circuit size. The ***fan-in*** of a prefix circuit is the maximum fan-in of all nodes in the circuit. The ***fan-out*** of a prefix circuit is the maximum fan-out of all nodes in the circuit. In this study, we are interested in prefix circuits with a fan-in of two and we assume that the fan-out of the prefix circuit is a function of $N$. In the rest of this section, we give a brief review of the design of some prefix circuits. For full description of these circuits, refer to [8] and [13].

## 2.1 The Serial Prefix Circuit

The layout of the serial circuit for $N$ inputs, denoted $S(N)$, is illustrated in Figure 2. Clearly, both size and depth of this circuit is $N$-1. The serial prefix circuit has the smallest size amongst all prefix circuits. Moreover, the circuit is (size, depth)-optimal since the sum of its size and depth is $2N - 2$.

## 2.2 Parallel Prefix Circuits

Figures 3 to 9 give illustrations of divide-and-conquer, Ladner-Fischer ($LF_0$), Ladner-Fischer ($LF_k$), Brent-Kung, Snir, Shih-Lin, and LYD prefix circuits, respectively. Information about their size, depth, and fan-out is given in Table 1. For complete details, refer to [8]. All these circuits have a depth $O(\lg N)$. Snir circuits are a family of circuits whose depth lies in the range [max ($\lg N$, $2\lg N - 2$), $N - 1$]. The divide-and-conquer circuit and $LF_0$ have fan-out $O(N)$, whereas all the other circuits have a fan-out of $O(\lg N)$. Ladner and Fischer [6] were the first to discuss the size-depth trade-off in prefix circuits. They introduced a family of circuits, $LF_k(N)$, where $k\,(0 \le k \le \lceil \lg N \rceil)$ refers to the *extra* depth (above $\lceil \lg N \rceil$) used to bring about the reduction in size. The circuit size and depth depend on the value of $k$. Snir [12] showed that the sum of depth and size of any prefix circuit with $N$ inputs is bounded below by $2N - 2$. He also introduced an algorithm to construct the (size, depth)-optimal prefix circuit for any $N$ with the depth in the range $\lceil \lg N \rceil \le depth(N) \le \max(\lceil \lg N \rceil, 2\lceil \lg N \rceil - 3)$ may not exist. Lakshmivarahan, Yang, and Dhall [7] were the first to introduce an algorithm for a (size, depth)-optimal parallel prefix circuit with the depth in the above range. Their design provides (size, depth)-optimal circuits with a smaller depth than hitherto known. Furthermore, for $N$ = 9 to 12, 17 to 20, and 33, the LYD circuits are not only (size, depth)-optimal, but are also depth-optimal.

## 2.3 Comparison

Table 1 provides a comparison of the prefix circuits illustrated in the previous subsection. While the parallel prefix circuits have desirable depths, which are $O(\lg N)$, they differ widely in the number of operations performed. Only four prefix circuits (i.e., serial, Snir, Shih-Lin, and LYD prefix circuits) are (size, depth)-optimal. The divide-and-conquer circuit and the $LF_0$ prefix circuit have the shortest depth and the serial circuit has the smallest size.

The size-depth trade-off does apply to any prefix circuit. For example, the serial prefix circuit performs fewest operations (i.e., smallest size) compared to the others, but has the longest depth while the divide-and-conquer prefix circuit has the largest size, but has the smallest depth. Although the Shih-Lin prefix circuit and the Snir prefix circuit have similar circuit layouts, Shih-Lin's circuit has a smaller depth than Snir's circuit. All circuits have unbounded fan-out except the serial circuit that has a constant fan-out of two. The divide-and-conquer prefix circuit and the $LF_0$ prefix circuit have the largest fan-out ($(N/2)+1$). Brent-Kung's circuit, Shih-Lin's circuit and Snir's circuit have the same fan-out ($\lceil \lg N \rceil + 1$), which is smaller than that of the LYD circuit ($2\lceil \lg N \rceil - 2$).

## 3. POWER CONSUMPTION IN CIRCUITS

In the previous section we examined size and depth trade-offs of different prefix circuit designs. We want to examine the power consumption characteristics of these circuits. In this section, the sources of power consumption in circuits are reviewed and the strategies to estimate the power consumption of the prefix circuits are presented.

### 3.1 Sources of Power Consumptions

Presently, CMOS (*Complementary-symmetry Metal-Oxide Semiconductor*) technology is the most popular technology used by the digital IC (Integrated Circuit) industry because of its low power consumption, its good scalability and its speed [5, 10, 14]. In CMOS circuits, power consumption is due to the following three types of current flow [14] (a) static power consumption due to leakage currents (b) dynamic power consumption due to short-circuit currents, and (c) dynamic power consumption due to switching currents from repetitively charging and discharging the parasitic capacitances at the transistors' gates (Figure 10). In properly designed CMOS circuits, the major portion of the power consumption is from dynamic switching [5, 10, 14]. As a result, in this study, we focus on the dynamic component due to the repetitive charging and discharging of the capacitive loads.

The average power consumption in a CMOS gate or module (e.g., an adder) due to switching can be written as [5, 14]:

$$P_{switching} = C_{eff} V_{DD}^2 f \,, \qquad (3.1)$$

where $C_{eff}$ is the effective capacitance switched, $V_{DD}$ is the supply voltage, and $f$ is the clock frequency. $C_{eff}$ has two components, the switching activity (signal transition activity) per clock cycle, $p_f$, and the load capacitance, $C_L$. Thus, for a given circuit running at a given speed (i.e., $C_L$ and $f$ constant), power consumption is a function of the supply voltage and switching activity. Therefore, power reduction can be achieved by either operating the circuit at a lower voltage or by choosing an architecture that reduces the switching activity of the circuit's signals.

**Effect of Voltage Scaling**

Due to the quadratic relationship between the supply voltage and the power consumption, lowering supply voltage can be an effective way to achieve dramatic power savings. However, as the supply voltage is decreased, the circuit delay generally increases relatively independent of the logic function and style(Figure 11). Thus, reducing supply voltage unfortunately reduces the system throughput. This loss in throughput can be recovered in some cases by applying architectural techniques to compensate for the additional delay (e.g., using parallelism and pipeline). Reference [5] shows that by changing circuit architecture it is possible to gain significant speed improvements with only a slight increase in power, hence enabling some voltage down-scaling while maintaining the throughput.

**Effect of Switching Activity**

The power in CMOS circuits is dissipated when the signals in the circuit switch (i.e., change values). As a result, the amount of switching activity is an indicator of the power consumption. The manner in which the nodes in a circuit are interconnected can have a strong influence on the overall switching activity [5]. Some architectures induce extra transition activity at the operation nodes called glitching transitions or dynamic hazards, which consume extra power. Glitching is a major problem that increases the effective switching activity, causing a circuit node to undergo several rapid transitions in a single clock cycle [5, 10].

Figure 12 illustrates an example of the glitching behavior for a chain of eight NAND gates [10] by using a PSpice® simulation [3]. In the simulation, all bits of the first input were set to logic 'one' and all bits of second input transition from logic 'zero' to 'one'. For an ideal circuit without propagation delays, the resultant outputs VOUT2, 4, 6 and 8 would stay logic 'one' all the time. However, due to the presence of delays, these outputs switch to low temporarily. This glitching causes extra power to be consumed. Outputs VOUT1, 3, 5 and 7 do not glitch; they just have some propagation delay. It is noted that the degree of glitching depends on the switching pattern of the input signals [10].

To reduce glitching activity, the depth of the signal paths in the circuit should be balanced. Figure 13 gives an illustration of two different circuit architectures of a 4-input adder. We assume that all primary inputs (A, B, C, and D) arrive at the time $t_0$ and the implementation is non-pipelined. While the adder in Figure 13a makes one transition by computing A+B, the second adder also makes one transition based on C and the previous (initial) value of A+B. After the correct value of A+B has propagated through the first adder at time say $t_0 + t_p$, the second adder re-evaluates (A+B)+C, which is complete at time $t_0 + 2t_p$. Thus, there is a second transition at the second adder. Similarly, there will be three transitions at the third adder. With a path-balancing approach of Figure 13(b), while the first and second adders make one transition the third adder will make only two transitions to produce the same output as in Figure 13(a). In [5], the "total switched capacitance" of the circuit layout in Figures 13(a) and 13(b) has been simulated by using a switch-level simulator over random input patterns. The results show that the switched capacitance of the circuit layout in Figure 13(a) is larger than that of the layout in Figure 13(b) by a factor of 1.5 for a four input addition, and 2.5 for an eight input addition. Hence, increasing circuit depth generally increases the total switched capacitance due to glitching and thus increases power consumption [5]. As a consequence, the amount of transition activity (switching activity) for a layered and non-pipelined circuit can be a function of depth $d$ and the number of nodes at each level $i$, $w_i$, as [5]

$$\sum_{i=1}^{d} i w_i \,. \qquad (3.2)$$

From this, it follows that in the worst case estimate for the switching activity of such a circuit can grow according to $O(d^2)$, assuming a constant number of nodes at each level.

From the previous discussion and the example of Figure 13, we have seen that different circuit architectures for performing the same function can consume different amounts of power. Therefore, the implementation of the various prefix circuits in an application will have different power consumption as well. However, in the prefix circuits, we cannot say with certainty that the circuit with the longer depth will consume more power than one with shorter depth. The reason is that both depth and the number of operation nodes among the candidate prefix circuits differ. In prefix circuits, when the depth decreases, the number of operation nodes (i.e., size) generally increases and vice versa. This is known as the size-depth trade-off [6, 8]. As a result, the switching activity in a prefix circuit not only depends on its logic depth but also on the number of operation nodes at each level. The circuit with shorter depth and more nodes might have more switching activity than the one with longer depth and fewer nodes.

### 3.2 Power Consumption and Fan-out

Besides the switching activity at an operation node, the node's fan-out also has an effect on power consumption in a circuit design in VLSI [4, 14]: the larger the fan-out, the more power the circuit consumes because there are more signals. For example, by using the PSpice over random input patterns, the power consumed by a 2-input XOR gate is dependent on the fan-out and the relationship is linear (Figure 14). Hence, fan-out should be taken into account when a power consumption estimate is made for the prefix circuit.

## 4. POWER MODELING OF PREFIX CIRCUITS

In this section, we will analyze switching activity and fan-out for each prefix circuit considered. We then use this to further estimate and investigate the power-speed trade-off between various types of prefix circuits.

Having seen the various sources of power consumption in general circuits we now focus on analytical model under linear output capacitance assumption for predicting the average power consumption of a prefix circuit. As mentioned previously, the signal switching activity has a major influence on the power consumption. Therefore, the switching activity will be used as a basis to determine power consumption of prefix circuits. Further, as mentioned in Section 3.2, the power consumption of an operation node is a linear function of fan-out [4]. Therefore, to take into account the effect of fan-out on the output load capacitance of an operation node, we assume that the load capacitance of a node with fan-out $k$ is equal to $C_0 + C'(k-1)$, where $C_0$ is the load capacitance of a node with fan-out 1, and $C'$ is the load capacitance for each additional fan-out (Figure 15).

The ***effective circuit capacitance*** of a prefix circuit, $cap_{eff}(N)$, is the effective load capacitance of all nodes in the circuit. As defined here, the effective circuit capacitance depends on input signal patterns and the effects of signal glitching. Thus if a node output experiences two transitions due to glitching, its effective capacitance is twice that of the physical capacitance. Because the degree of glitching depends on input signal patterns, we consider derivations of the worst case scenario in which glitching at the nodes are assumed to be the maximum possible. By scaling the effective circuit capacitance by the circuit clock frequency and $V_{DD}^2$, we arrive at our power estimate

$$P = cap_{eff}(N)V_{DD}^2 f \; . \qquad (4.1)$$

The capacitance evaluation for various circuits according to our model is made in two steps. As a first step, in Section 4.1, we assume that load capacitance for each operation node is independent of the fan-out, i.e., the load capacitance is constant $C_0$. In the second step we first compute the residual circuit by deleting one output of each operation node with fan-out ≥ 1. We then compute the load capacitance of the residual circuit assuming that the load capacitance of each node is $C'$, independent of the fan-out. This step is repeated $k-1$ times where $k$ is the fan-out of the given circuit. This step is performed in Section 4.2. The effective circuit capacitance is the sum of the values obtained in step 1 and step 2.
In the following, we compute the effective circuit capacitance for the divide-and-conquer prefix circuit. The effective circuit capacitance for the other prefix circuits can be computed similarly (for details refer to [13]).

## 4.1 Step 1 - The Constant Output Capacitance

In this step, we assume that the physical output capacitance of each operation node is constant. Let $Kcap_{eff}(N)$ be the effective circuit capacitance under the constant output capacitance assumption, $depth(N)$ be the depth of the circuit, $w_i$ be the number of operation nodes in the circuit at level $i$, and $C_0$ as the assumed constant load capacitance of one node. Then from Eq. 3.2, $Kcap_{eff}(N) = \left( \sum_{i=1}^{depth(N)} i w_i \right) C_0$.

### 4.1.1. The Divide-and-Conquer Parallel Prefix Circuit

Let $N = 2^n$. From the layout of the divide-and-conquer prefix circuit, $DC(N)$, in Figure 3, $DC(N)$ is built from two $DC(N/2)$ circuits and by connecting output $1:N/2$ from the first $DC(N/2)$ to each of the output of the second $DC(N/2)$ at level $depth(N/2)+1 = \lg(N/2)+1 = \lg N$. Thus,

$$Kcap_{eff}(N) = \left(2Kcap_{eff}(N/2) + (N/2)\lg N\right) \cdot C_0, \qquad \text{with} \qquad Kcap_{eff}(2) = 1 \cdot C_0 .$$

The first part of $Kcap_{eff}(N)$ is the constant output capacitance from the two circuits with $(N/2)$ inputs while the second part is the capacitance from the last level of $DC(N)$. Solving this recurrence, we get

$$Kcap_{eff}(N) = (N/4)\left((\lg N)^2 + \lg N\right)C_0$$

$Kcap_{eff}(N)$ for the other prefix circuits can be computed similarly, although they are generally more challenging because $w_i$ is not always constant (for details refer to [13]).

## 4.2. Step2 – Capacitance of Residual Circuit

We have assumed that a node with fan-out $k \geq 1$, has a physical output capacitance given as $C_0 + (k-1)C'$. However, the capacitances computed in Section 4.1 for various circuits are based on the assumption that the capacitance of each node is $C_0$ irrespective of the fan-out of the node. We still need to account for the component $(k-1)C'$ for a node with fan-out $k$, $k > 1$. To get this value, we introduce the concept of the *residual circuit*. The residual circuit of a prefix circuit is the circuit obtained by eliminating one of the fan-outs from each operation node of the given prefix circuit. For example, Figure 16 shows the residual circuit of the divide-and-conquer prefix circuit. This residual circuit is the result of removing one of the fan-outs from each operation node of the circuit in Figure 3. We can compute the capacitance of this residual circuit, $Rcap_{eff}(N)$, by assuming constant output capacitance ($C'$) for all operation nodes. We then construct the residual circuit of the current residual circuit by removing one fan-out from each operation node and compute its residual output capacitance. We continue accumulating the capacitances after every reduction until there are no more fan-outs to remove. Thus, the effective circuit capacitance of the prefix circuit using the linear output capacitance assumption is given by

$$cap_{eff}(N) = Kcap_{eff}(N)C_0 + Rcap_{eff}(N)C' .$$

**4.2.1. The Divide-and-Conquer Parallel Prefix Circuit**

From the layout of the divide-and-conquer prefix circuit in Figure 3, an operation node at level $depth(N/2)$ has the maximum fan-out, which is $((N/2)+1)$. After removing the vertical fan-outs, the residual circuit is shown in Figure 16. The operation node of the residual circuit at level $depth(N/2)$ has the maximum fan-out, which is $(N/2)$.

Let $N = 2^n$. The capacitance of the residual circuit is as follows:

$$Rcap_{eff}(N) = \left(2Rcap_{eff}(N/2) + (N/2)\lg(N/2)\right)C', \quad \text{with} \quad Rcap_{eff}(2) = 0.$$

The first part of $Rcap_{eff}(N)$ is the residual output capacitance of the two circuits with $(N/2)$ inputs while the second part is the residual output capacitance of the last node in the fist residual circuit.

Solving the recurrence, we get

$$Rcap_{eff}(N) = \left(2Rcap_{eff}(N/2) + (N/2)\lg(N/2)\right)C' = (N/4)\left((\lg N)^2 - \lg N\right)C'.$$

Thus, the effective circuit capacitance for the divide-and-conquer prefix circuit is as follows.

$$cap_{eff}(N) = \left\{(N/4)\left((\lg N)^2 + \lg N\right)\right\}C_0 + \left\{(N/4)\left((\lg N)^2 - \lg N\right)\right\}C'.$$

To summarize, the divide-and-conquer prefix circuit has $O(N \lg N)$ size, $O(\lg N)$ depth, and $O(N(\lg N)^2)$ effective circuit capacitance. Table 2 provides a comparison of the effective circuit capacitance of the prefix circuits described in Section 2. The serial prefix circuit has the largest effective circuit capacitance ($O(N^2)$). All parallel prefix circuits have $O(N \lg N)$ effective circuit capacitance, except the divide-and-conquer prefix circuit and the $LF_0$ prefix circuit whose values are $O(N(\lg N)^2)$.

## 5. SIMULATION STUDIES

In Section 4, the power modeling for various prefix circuits was proposed. This section deals with the circuit simulations (using PSpice) we conducted to investigate the prefix circuits' behavior to match with the prediction of the effective circuit capacitance. The degrees of freedom studied include different prefix circuit designs and voltage scaling. Voltage scaling is used because power consumption is a quadratic function of the voltage.

**Theoretical Results**

Figures 18, 20, and 22 give estimated delay, power consumption, and power-delay product obtained from our theoretical model in Section 4. Figure 18 is the result obtained by assuming the circuits' delay to be proportional to the circuits' depth and applying the normalized delay from Figure 17 in order to take the effect of the supply voltage on the delay. The power consumption is estimated using the formula of Eq. 4.1. For this study we used $C_0 = 0.9$ and $C' = 0.3$ [11]. For example, at a supply voltage of 2.8V., the normalized power consumed by the divide-and-conquer prefix circuit is:

$$P(normalized) = cap_{eff}(N)V_{dd}^2 f = \left(2{,}496C'\right)(2.8)^2 f/(C'f) \approx 19{,}569.$$

The estimated power consumption of parallel prefix circuits described in Section 2 is shown in Figure 20. According to the figure, the divide-and-conquer prefix circuit consumes the most power. Figure 22 illustrates the power-delay product. The Brent-Kung prefix circuit has the highest power-delay product while the divide-and-conquer and the $LF_0$ prefix circuits have the power-delay product lower than that of the Brent-Kung prefix circuit, the Snir prefix circuit, the Shih-Lin prefix circuit and the LYD prefix circuit.

Table 3 shows the estimated power consumption of the different prefix circuits at fixed and reduced supply voltage when $N = 64$. When the supply voltage is fixed at 2.8V, amongst parallel prefix circuits considered, the divide-and-conquer prefix circuit consumes more power than other circuits. To lower power consumption by reducing the supply voltage, let us assume a fixed acceptable delay. Further, assume that delay is proportional to depth and that a delay proportional to a depth of 10 with $V_{DD} = 2.8$ volts is acceptable. Thus the voltage of the Brent-Kung and Snir circuits cannot be lowered, and the delay of the serial circuits is not acceptable. Thus, the voltages of five prefix circuits (i.e., the divide-and-conquer prefix circuit, the $LF_0$ prefix circuit, the $LF_1$ prefix circuit, the Shih-Lin prefix circuit, and the LYD prefix circuit) can be dropped from 2.8V and still achieve the acceptable delay. For example, because the delay for the divide-and-conquer prefix circuit is proportional to 6 at 2.8V, the voltage can be dropped from 2.8V to 1.48V. The operating frequency can be decreased by a factor of 0.6. Thus the normalized power consumed by the divide-and-conquer prefix circuit is:

$$P(normalized) = cap_{eff}(N)V_{dd}^2 f = (2{,}496C')(1.48)^2(0.6f)/(C'f) \approx 3{,}280\,.$$

After scaling the supply voltage, there is a power improvement in the circuits having depth shorter than 10. Among these circuits, the $LF_0$ prefix circuit has a major reduction in power due to its shortest depth.

**Simulation Results**

PSpice simulation was carried out on different parallel prefix circuits with 64 inputs using XOR gate as an associative binary operation. Figures 19, 21, and 23 give delay, power consumption, and power-delay product obtained through the simulation over random inputs. As expected, amongst the parallel prefix circuits considered, the divide-and-conquer prefix circuit consumes the most power. As the supply voltage is reduced, power consumption is also reduced. Also, though the delay of the divide-and-conquer prefix circuit is the least for some values of the voltage supply, it is not so for lower voltages. This may be due to its very high fan-out compared to others ( $O(N)$ *vs* $O(\lg N)$ ). From the point of view of the power-delay product metric, the LYD prefix circuit is found to be the best across the entire voltage scaling. This means that the circuit provides the best trade-off between power and delay. Another result of the simulation studies shows that the power-delay product of the divide-and-conquer circuit is the highest, followed by that of the $LF_0$ circuit. This is at variance with our model prediction and may be due to the fact that these circuits have a very high fan-out (see Table 1 for fan-out). In our theoretical results, we do not take into account the effect of fan-out on the delay.

Also according to the simulation, with voltage-scaling technique, the LYD prefix circuit has the least power consumption compared to other circuits. For example, let us assume the maximum acceptable delay is 6.4 μs. From Figures 19 and 21, to achieve this time-delay, the supply voltage of the divide-and-conquer, $LF_0$, $LF_1$, Shih-Lin, and LYD prefix circuits can be 1.8V, 1.78V, 1.78V, 2V, and 1.8V, respectively. Therefore, the powers that the divide-and-conquer, $LF_0$, $LF_1$, Shih-Lin, and LYD prefix circuits consume are 2.25, 1.94, 1.59, 1.64, and 1.44 W, respectively. This shows that power reduction of about 1.6 times can be obtained without speed loss by using the LYD prefix circuit compared with using the divide-and-conquer prefix circuit by using appropriately chosen supply voltage.

## 6. CONCLUSIONS

The power consumption and the power-delay product of seven parallel prefix circuits were compared. We have shown that the use of our effective circuit capacitance provides results that are accurate when compared to PSpice simulations. We have also shown that parallelism at a certain level coupled with the use of low supply voltage can be used to reduce the power consumption in the circuit without throughput loss. The main discrepancy between the model and the simulation is the power-delay product metric. This may be due to the fact that the fan-out of the divide-and-conquer and the $LF_0$ prefix circuit is very high as compared to other circuits. In this analysis, we have assumed that the delay is uniquely determined by the depth of the circuit. The results of the simulation of the divide-and-conquer circuit in particular indicate that large fan-out in addition to contributing to more power may also indirectly affect the delay.

## ACKNOWLEDGEMENTS

This work was supported by DARPA under contract No. F30602-97-2-0297.

## REFERENCES

1. C. Belady, "Cooling and Power Consideration for Semiconductors Into the Next Century", *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pp.100-105, 2001.
2. R. P. Brent, and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Transactions on Computers*, Vol. 31, pp. 260-264, 1982.
3. Cadence Design Systems, Inc., *PSpice User's Guide Manual*, Version 9.2, San Jose, CA, January 2000.
4. T. K. Callaway, *Area, Delay, and Power Modeling of CMOS Adder and Multipliers*, Ph.D. Dissertation, The University of Texas at Austin, 1996.
5. A. P. Chandrakasan, and R. W. Brodersen, *Power Digital CMOS Design*. Kluwer Academic Publishers, Norwell, MA, 1995.

6. R. E. Ladner, and M. J. Fischer, "Parallel Prefix Computation", *Journal of ACM*, Vol. 27, pp. 831-838, 1980.
7. S Lakshmivarahan, C. M. Yang, and S. K. Dhall, "Optimal Parallel Prefix Circuits with $(size, depth) = 2N-2$ and $\lceil \log N \rceil \leq depth \leq \lceil 2\log N \rceil - 3$", *Proceedings of the International Conference on Parallel Processing*, pp. 58-65, 1987.
8. S. Lakshmivarahan, and S. K. Dhall, *Parallel Computing Using the Prefix Problem*. Oxford University Press, New York, NY, 1994.
9. Y. M. Lin, and C. C. Shih, "A New Class of Depth-Size Optimal Parallel Prefix Circuits", *Journal of Supercomputing*, Vol. 14, pp. 39-52, 1999.
10. J. M. Rabaey, A. Chandrakasan, and B. Nikolic, "Chapter 6: Designing Combinational Logic Gates in CMOS", *Digital Integrated Circuits A Design Perspective*, early draft of the 2nd edition, April 2001, http://bwrc.eecs.berkeley.edu/Classes/IcBook/2ndEdition.html.
11. M. Smith, *Application-Specific Integrated Circuits*, Addison Wesley, Menlo Park, CA, 1997.
12. M. Snir, "Depth-Size Tradeoffs for Parallel Prefix Computation", *Journal of Algorithms*, Vol. 17, pp. 185-201, 1986.
13. S. Vanichayobon, *Power-speed Trade-off in Parallel Prefix Circuits*, Ph.D. Dissertation, School of Computer Science, The University of Oklahoma, 2002.
14. N. H. E. Weste, and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, Addison-Wesley, MA, 1993.

## Figures

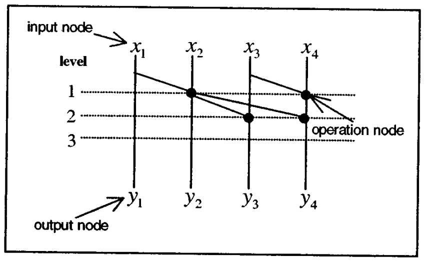


**Figure 1:** An illustration of the prefix circuit's layout.

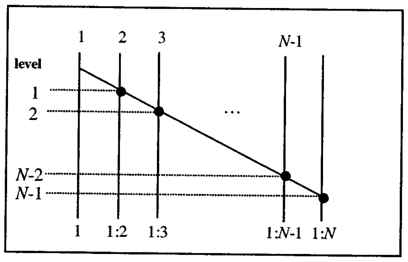


**Figure 2:** An illustration of the serial prefix circuit, $S(N)$.

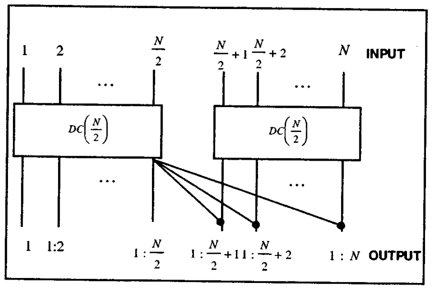

1 2 ... $\frac{N}{2}$ $\frac{N}{2}+1$ $\frac{N}{2}+2$ ... $N$ INPUT
$DC\left(\frac{N}{2}\right)$
$DC\left(\frac{N}{2}\right)$
1 1:2 $1:\frac{N}{2}$ $1:\frac{N}{2}+1$ $1:\frac{N}{2}+2$ $1:N$ OUTPUT

**Figure 3:** An illustration of the divide-and-conquer prefix circuit,

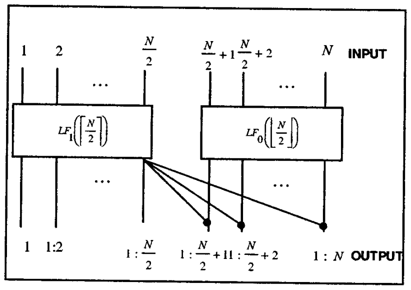

1 2 ... $\frac{N}{2}$ $\frac{N}{2}+1$ $\frac{N}{2}+2$ ... $N$ INPUT
$LF_1\left(\left\lceil\frac{N}{2}\right\rceil\right)$
$LF_0\left(\left\lfloor\frac{N}{2}\right\rfloor\right)$
1 1:2 $1:\frac{N}{2}$ $1:\frac{N}{2}+1$ $1:\frac{N}{2}+2$ $1:N$ OUTPUT

**Figure 4:** An illustration of the Ladner-Fischer parallel prefix circuit when $k = 0$, $LF_0(N)$, derived from [6].

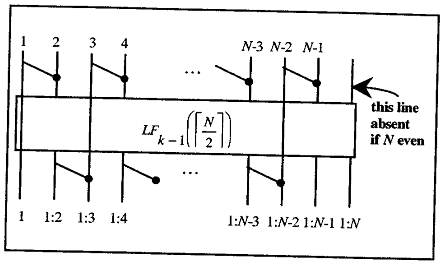

1 2 3 4
N-3 N-2 N-1
this line absent if N even
$LF_{k-1}\left(\left\lceil \frac{N}{2} \right\rceil\right)$
1 1:2 1:3 1:4
1:N-3 1:N-2 1:N-1 1:N

**Figure 5:** An illustration of the Ladner-Fischer parallel prefix circuit when $k \neq 0$, $LF_k(N)$, derived from [6].

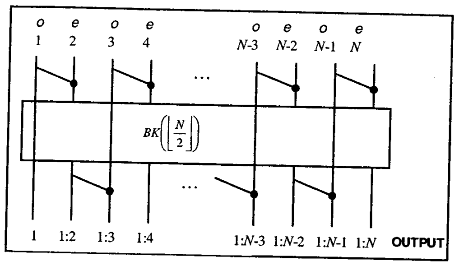

o e o e o e o e
1 2 3 4 N-3 N-2 N-1 N
$BK\left(\left\lfloor \frac{N}{2} \right\rfloor\right)$
1 1:2 1:3 1:4
1:N-3 1:N-2 1:N-1 1:N OUTPUT

**Figure 6:** A Brent-Kung parallel prefix circuit, $BK(N)$, based on divide-and-conquer strategy ($o$ = odd, $e$ =even).

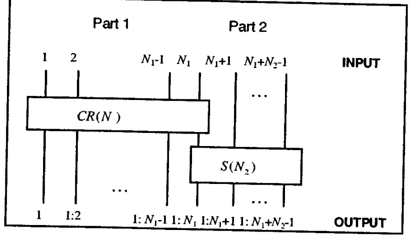

Part 1
Part 2
1 2 $N_1$-1 $N_1$ $N_1$+1 $N_1$+$N_2$-1
INPUT
CR(N)
S($N_2$)
1 1:2 1:$N_1$-1 1:$N_1$ 1:$N_1$+1 1:$N_1$+$N_2$-1
OUTPUT

**Figure 7:** An illustration of the Snir prefix circuit, $SN(N)$.

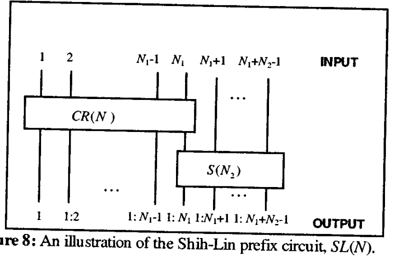

1 2 $N_1$-1 $N_1$ $N_1$+1 $N_1$+$N_2$-1
INPUT
CR(N)
S($N_2$)
1 1:2 1:$N_1$-1 1:$N_1$ 1:$N_1$+1 1:$N_1$+$N_2$-1
OUTPUT

**Figure 8:** An illustration of the Shih-Lin prefix circuit, $SL(N)$.

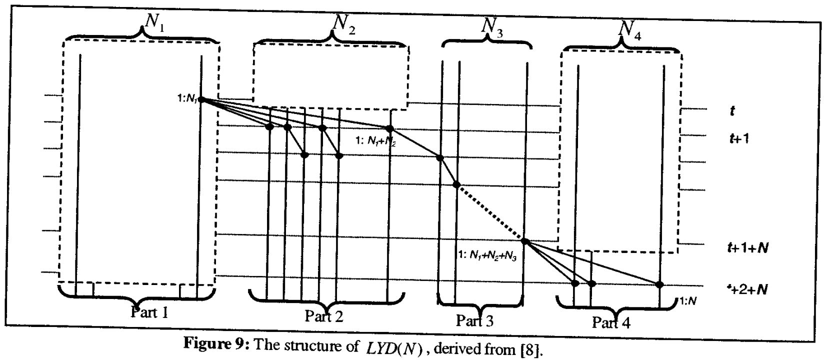


**Figure 9:** The structure of $LYD(N)$, derived from [8].

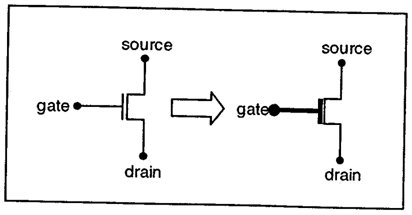


**Figure10:** An illustration of capacitance charging current.

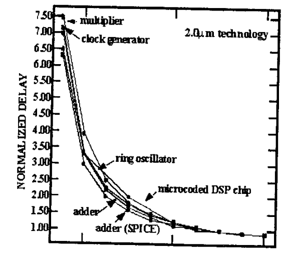


**Figure 11:** Plots of normalized delay vs. supply voltage ($V_{dd}$) for a variety of different logic circuits, derived from [5].

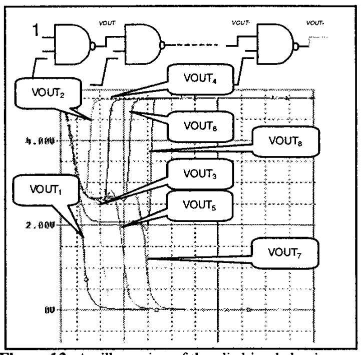


**Figure 12:** An illustration of the glitching behavior of a chain of eight NAND gates, derived from [10].

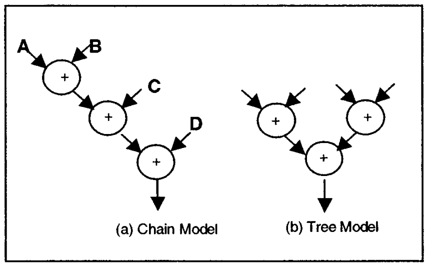


**Figure 13:** An illustration of extra transition activity, derived from [5].

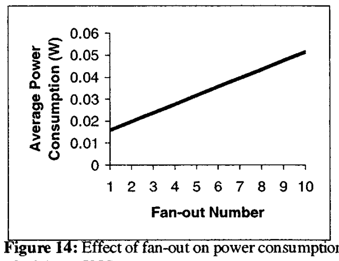


**Figure 14:** Effect of fan-out on power consumption of a 2-input XOR gate.

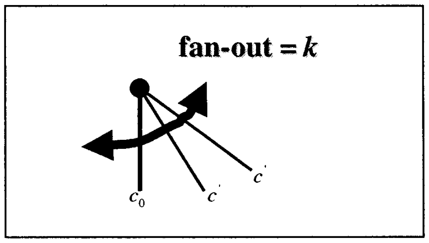
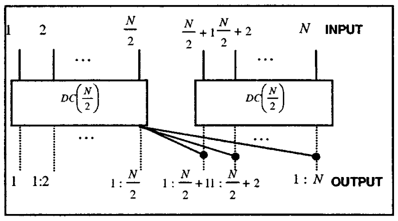


**Figure 15:** The load capacitance of a node with fan-out *k*.



1 2 $\frac{N}{2}$ $\frac{N}{2}+1 \frac{N}{2}+2$ $N$ INPUT
$DC\left(\frac{N}{2}\right)$
$DC\left(\frac{N}{2}\right)$
1 1:2 $1:\frac{N}{2}$ $1:\frac{N}{2}+1 1:\frac{N}{2}+2$ $1:N$ OUTPUT

**Figure 16:** The residual circuit of the divide-and-conquer prefix circuit, *DC*(*N*), shown in solid lines.

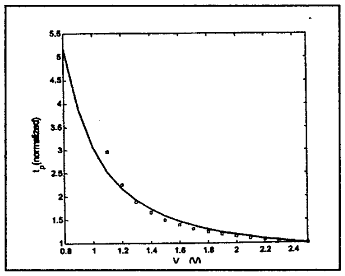


**Figure 17:** Plot of supply voltage vs. normalized delay from [5].

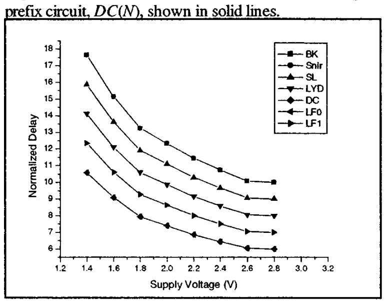


**Figure 18:** Estimated delay of parallel prefix circuits when *N*=64.

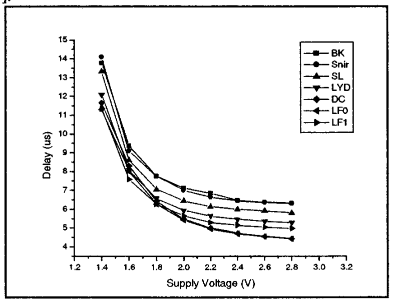


**Figure 19:** Delay of the 64-bit XOR parallel prefix

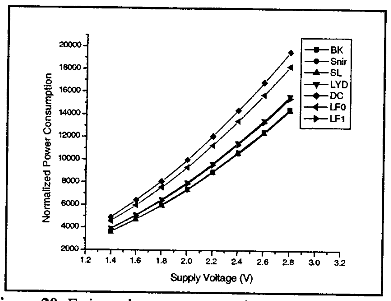


**Figure 20:** Estimated power consumption of parallel prefix circuits when $N$=64.

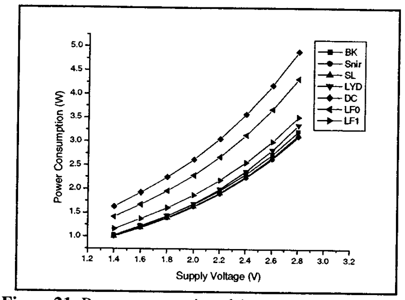


**Figure 21:** Power consumption of the 64-bit XOR parallel prefix circuits, obtained through PSpice simulation.

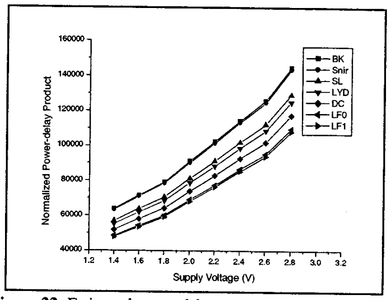


**Figure 22:** Estimated power-delay product of parallel prefix circuits when $N$=64.

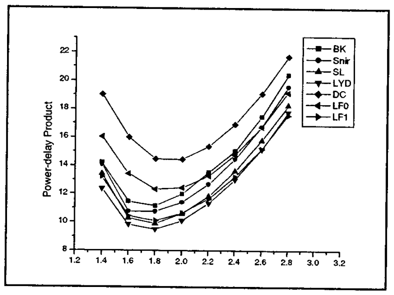


**Figure 23:** Power-delay product of the 64-bit XOR parallel prefix circuits, obtained through PSpice simulation.

**Table 1:** A Comparison of the six prefix circuits illustrated in Section 2, when $N$=$2^n$.

| Prefix Circuit | Size | Depth | Fan-out | (size, depth)-optimal |
|---|---|---|---|---|
| Serial | $N-1$ | $N-1$ | 2 | Yes |
| Divide-and-Conquer | $(N/2)\lg N$ | $\lg N$ | $(N/2)+1$ | No depth-optimal |
| $LF_0$ | $4N-F(5+\lg N)+1$ | $\lg N+k$ | $(N/2^{k+1})+k$ | No |
| $LF_k$ when $0<k<\lg N$-2 | $2N(1+(1/2^k))-F(5+\lg N-k)-k+1$ | | | |
| $LF_k$ when $k\geq\lg N$-2 | $2N-\lg N-2$ | $2\lg N-2$ | $\lg N+1$ | |
| Brent-Kung | $2N-\lg N-2$ | $2\lg N-2$ | $\lg N+1$ | No |
| Snir | $2N-2-depth$ | $\max(\lg N, 2\lg N-2)$ $\leq depth\leq N-1$ | $\lg N+1$ | Yes |
| LYD | $2N-2-depth$ | $2\lg N-6\leq depth\leq 2\lg N-3$ | $2\lg N-2$ | Yes |
| Shih-Lin | $2N-2-depth$ | $2\lg N-5\leq depth\leq 2\lg N-3$ | $\lg N+1$ | Yes |

**Table 2:** Comparison of effective circuit capacitance of prefix circuits

| **Prefix Circuit** | ***cap*$_{eff}$*(N)*** |
|---|---|
| Serial | $\{\frac{N(N-1)}{2}\}C_0+\{\frac{(N-1)(N-2)}{2}\}C'$ |
| Divide-and-Conquer | $\{\frac{N}{4}((\lg N)^2+\lg N)\}C_0+\{\frac{N}{4}((\lg N)^2-\lg N)\}C'$ |
| Brent-Kung | $\{1+\frac{3}{2}N\lg N-\frac{1}{2}[2N+(\lg N)^2+\lg N]\}C_0+\{3(1+\frac{N}{2}\lg\frac{N}{2})-\frac{1}{2}(3N+(\lg\frac{N}{2})^2+\lg\frac{N}{2})\}C'$ |
| *LF$_k$* | $\{\frac{N}{4}((\lg N)^2+\lg N)\}C_0+\{\frac{N}{4}((\lg N)^2-\lg N)\}C'\leq LF_k\leq\{1+\frac{3}{2}N\lg N-\frac{1}{2}[2N+(\lg N)^2+\lg N]\}C_0+$<br>$\{3(1+\frac{N}{2}\lg\frac{N}{2})-\frac{1}{2}(3N+(\lg\frac{N}{2})^2+\lg\frac{N}{2})\}C'$ |
| Snir | $\{[1+\frac{3}{2}N_1(\lg N_1)]-[\frac{1}{2}[2N_1+(\lg N_1)^2+(\lg N_1)]]+[N_2\lceil(\lg N_1)\rceil-\lceil(\lg N_1)\rceil+(\frac{N_2^2-N_2}{2})]\}C_0+$<br>$\{[3(1+\frac{N_1}{2}\lg\frac{N_1}{2})-\frac{1}{2}(3N_1+(\lg\frac{N_1}{2})^2+\lg\frac{N_1}{2})]+[\lceil\lg N_1\rceil(N_2-1)+\frac{1}{2}(N_2^2-3N_2+2)]\}C'$ |
| Shih-Lin | $\{[1+\frac{3}{2}N_1(\lg N_1)]-[\frac{1}{2}[2N_1+(\lg N_1)^2+(\lg N_1)]]+[N_2\lceil(\lg N_1)\rceil-\lceil(\lg N_1)\rceil+(\frac{N_2^2-N_2}{2})]\}C_0+$<br>$\{[3(1+\frac{N_1}{2}\lg\frac{N_1}{2})-\frac{1}{2}(3N_1+(\lg\frac{N_1}{2})^2+\lg\frac{N_1}{2})]+[\lceil\lg N_1\rceil(N_2-1)+\frac{1}{2}(N_2^2-3N_2+2)]\}C'$ |
| LYD | $\{[1+\frac{3}{2}N_1\lg N_1]-[\frac{1}{2}[2N_1+(\lg N_1)^2+\lg N_1]]+[\frac{2\lceil\lg N_1\rceil^3}{3}+2\lceil\lg N_1\rceil^2+\frac{4\lceil\lg N_1\rceil}{3}+1]+$<br>$[(N_3+N_4)(\lceil\lg N_1\rceil+\frac{3}{2})+\frac{1}{2}(N_3^2+N_4^2)+(N_3N_4)]\}C_0+$<br>$\{[3(1+\frac{N_1}{2}\lg\frac{N_1}{2})-\frac{1}{2}(3N_1+(\lg\frac{N_1}{2})^2+\lg\frac{N_1}{2})]+[\frac{2}{3}\lceil\lg N_1\rceil+\lceil\lg N_1\rceil^2+\frac{\lceil\lg N_1\rceil}{3}]+$<br>$[\frac{N_3}{2}(2\lceil\lg N_1\rceil+N_3+1)+\frac{N_4^2}{2}+N_4\lceil\lg N_1\rceil+N_3N_4+1-\frac{N_4}{2}]\}C'$ |

**Table 3:** Estimated power consumption based on Eq. 3 for various prefix circuits for $N = 64$, $C_0 = 3C'$.

| **Prefix Circuit** | **Depth** | ***cap*$_{eff}$(64)** | **Power (normalized) V$_{dd}$= 2.8V** | **NewPower (normalized) after reducing V$_{dd}$** |
|---|---|---|---|---|
| Serial | 63 | $2016C_0+1953C'$ | 62,728 | - |
| Divide-and-Conquer | 6 | $672C_0+480C'$ | 19,569 | 3,280 V$_{dd}$= 1.48V |
| Brent-Kung | 10 | $492C_0+372C'$ | 14,488 | 14,488 V$_{dd}$= 2.8V |
| *LF$_0$* | 6 | $625C_0+457C'$ | 18,283 | 3,065 V$_{dd}$= 1.48V |
| *LF$_1$* | 7 | $527C_0+390C'$ | 15,453 | 3,987 V$_{dd}$= 1.7V |
| Snir | 10 | $487C_0+371C'$ | 14,363 | 14,363 V$_{dd}$= 2.8V |
| Shih-Lin | 9 | $487C_0+370C'$ | 14,355 | 9,491 V$_{dd}$= 2.4V |
| *LYD* | 8 | $528C_0+410C'$ | 15,633 | 6,381 V$_{dd}$= 2V |

**Appendix K:** Jack M. West, Hongping Li, Sirirut Vanichayobon, Jeffrey T. Muehring, John K. Antonio, and Sudarshan K. Dhall, “A Hybrid FPGA/DSP/GPP Prototype Architecture for SAR and STAP,” *Proceedings of the Fourth Annual High Performance Embedded Computing Workshop*, sponsors: U.S. Navy and Defense Advanced Research Projects Agency (DARPA), MIT Lincoln Laboratory Publications, Group 18, Lexington, MA, Sep. 2000, pp. 29-30.

# A Hybrid FPGA/DSP/GPP Prototype Architecture for SAR and STAP

Jack M. West, Hongping Li, Sirirut Vanichayobon, Jeffrey T. Muehring,
John K. Antonio, and Sudarshan K. Dhall

School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019-6151
Phone: 405-325-7859
Fax: 405-325-4044
antonio@ou.edu

*HPEC 2000*
*The Fourth Annual Workshop on High-Performance Embedded Computing*

**Abstract**

A prototype system is described that demonstrates the advantages and trade-offs associated with the combined use of different hardware technologies for two embedded radar processing applications. The primary metrics of interest are size, weight, and power utilizations. The system can be configured with FPGAs (field programmable gate arrays), DSPs (digital signal processors), and/or GPPs (general purpose processors). The two radar applications evaluated are SAR (synthetic aperture radar) and STAP (space-time adaptive processing). Although the prototype system is not evaluated through actual fielded studies, experiments involving continuous input streams at relatively high rates are conducted in the laboratory using stored and unprocessed radar data as input.

The FPGA components of the prototype system are commercially available WildOne and WildForce boards (from Annapolis Microsystems) populated with 4000-series Xilinx parts. The WildForce boards each have four 4085-series FPGAs plus one control FPGA. The DSP/GPP components of the system are within a Mercury Race Multicomputer configured with both SHARC and PowerPC compute nodes. The Mercury system can be configured with up to eight PowerPC nodes and eight SHARC compute nodes (each SHARC compute node actually contains three SHARC DSP chips). An overview of the overall architecture is depicted in Figure 1.

The source PC is responsible for initially loading unprocessed radar data (from disk) into a circular buffer within its main memory. Once the input data is loaded into the circular buffer, the source PC then continuously (and repeatedly) streams this data into the front-end FPGA subsystem, denoted as (F) in Figure 1. It was necessary to locate the input data in a large main memory buffer in order to achieve realistic data throughput rates, which would otherwise not be possible if the data were streamed directly from the disk of the source PC.

All of the Annapolis FPGA boards are PCI-based and reside on the data source and/or data sink PCs. A total of four WildForce boards are available, and zero or more of these may reside on the source and sink PCs. The source and sink PCs also contain one WildOne board each. The WildOne boards are not used for computation, but handle the

data communication (through the PCI bus) between the PCs and the FPGA subsystems. The data communication among all FPGA boards is through two types of 36-bit wide connectors, one called systolic and one called SIMD.

The data communication between the front-end FPGA subsystem (F) and the DSP/GPP subsystem is a custom interface developed using the systolic connector from Annapolis and the RIN-T input device from Mercury. Similarly, the data communication between the DSP/GPP subsystem and the back-end FPGA subsystem (B) is through a custom interface developed using the ROUT-T output device from Mercury and the systolic connector from Annapolis.

Figures 2 and 3 illustrate how the major computational components of the SAR and STAP applications can be mapped onto the prototype system. A candidate mapping is defined by assigning the computations of each major component to one or both of the symbols shown in each block (which correspond to one of the FPGA or DSP/GPP subsystems). Using SAR to illustrate, one mapping would be to perform all of the range compression on the front-end FPGA subsystem (F) and then perform all azimuth processing on the DSP/GPP subsystem. Another possible mapping is defined by using the FPGA subsystems and the DSP/GPP for both components of computation. It is also possible to use only the DSP/GPP subsystem for both components of computations.

The SAR studies were designed by adapting the RASSP (Rapid Prototyping of Application Specific Signal Processors) benchmark developed originally by Lincoln Laboratory at MIT. The benchmark, which was originally implemented in serial C code, was first modified to execute on the parallel DSP/GPP subsystem. A data-streaming component was also added so that input data can be sent continuously from the data source of the prototype system. Core computations from the range compression and azimuth processing components were implemented for the FPGA subsystems.

The STAP studies were designed by adapting the RT_STAP (Real Time STAP) benchmark developed originally at MITRE. This benchmark was already implemented for parallel execution on a PowerPC-based Mercury system. This implementation was expanded to also enable execution on SHARC compute nodes. The same basic data streaming component that was developed for SAR was also adapted to enable the STAP input data to be sent continuously from the data source. Finally, core computations from the range compression and weight computation components from the STAP processing flow were implemented for the FPGA subsystems.

The size, weight, and power utilizations of various mappings and problem instances are under investigation. Initial indications are that heterogeneous configurations, which utilize two or more hardware technologies of the prototype system, are preferred over homogeneous configurations.

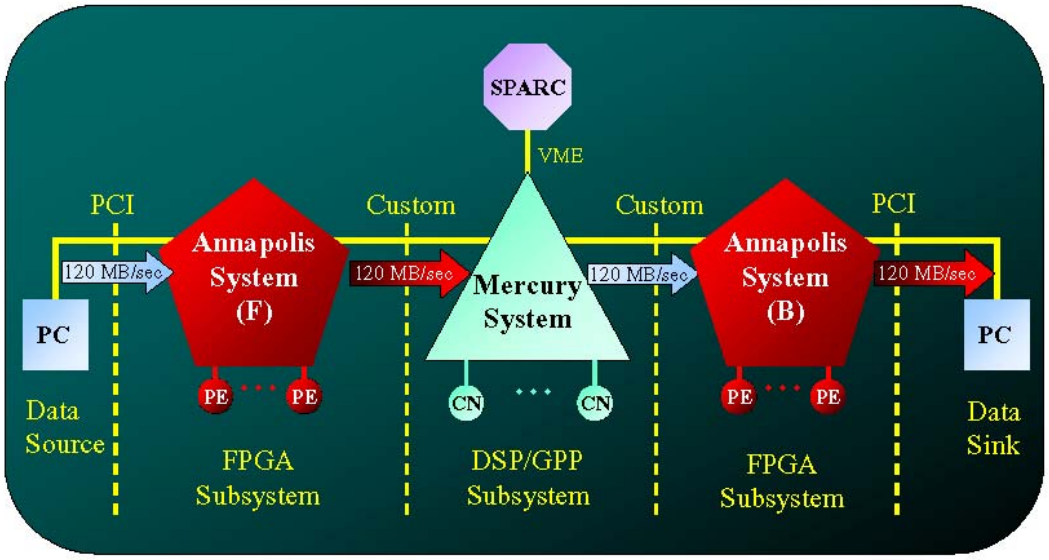


Figure 1. Overview of the architecture of the prototype system.

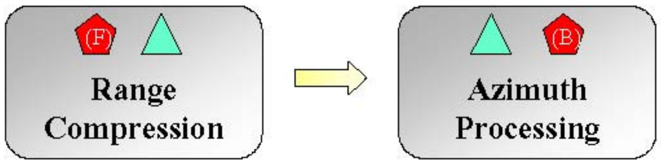


Figure 2. Major computational components of SAR processing flow.

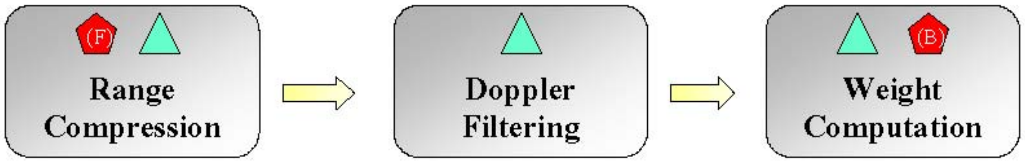


Figure 3. Major computational components of STAP processing flow.