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.

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.

0743-7315/02 $35.00

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® 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 3×4 process set (i.e., three rows and four columns for a total of 12 CNs). The 3×4 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

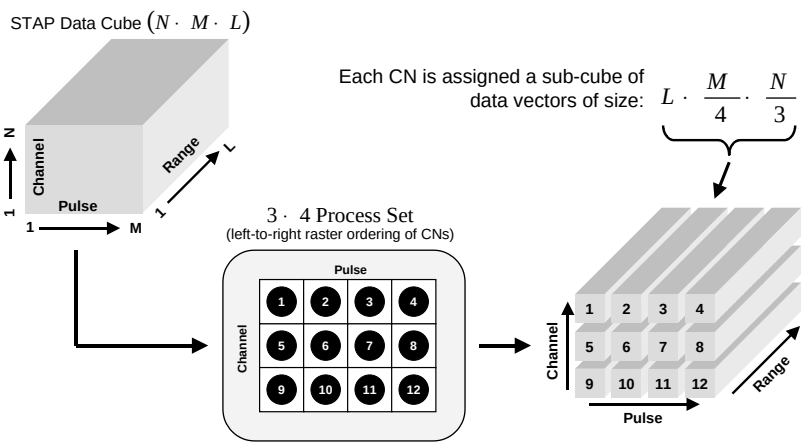

FIG. 1. Process set partitioning of a STAP data cube for range processing.

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

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

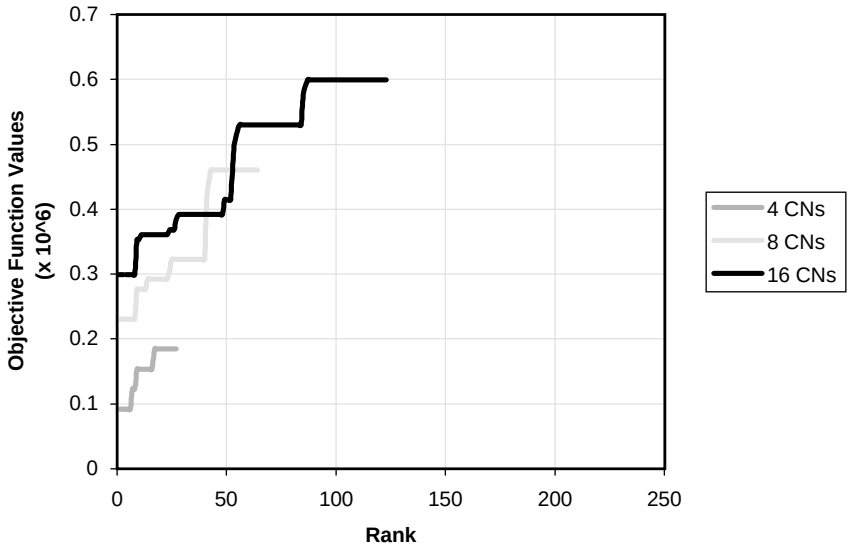

FIG. 2. Illustration of the mapping objective function values associated with mapping a $240 \times 32 \times 16$ STAP data cube mapped onto a 4, 8, and 16 CN system.

Figure 2 compares mapping objective function values associated with mapping a $240 \times 32 \times 16$ 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 ith chromosome. In addition, let f_i represent the fitness of the ith chromosome, and let the average fitness of the population be f_{avg}. In this selection scheme, the ith

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

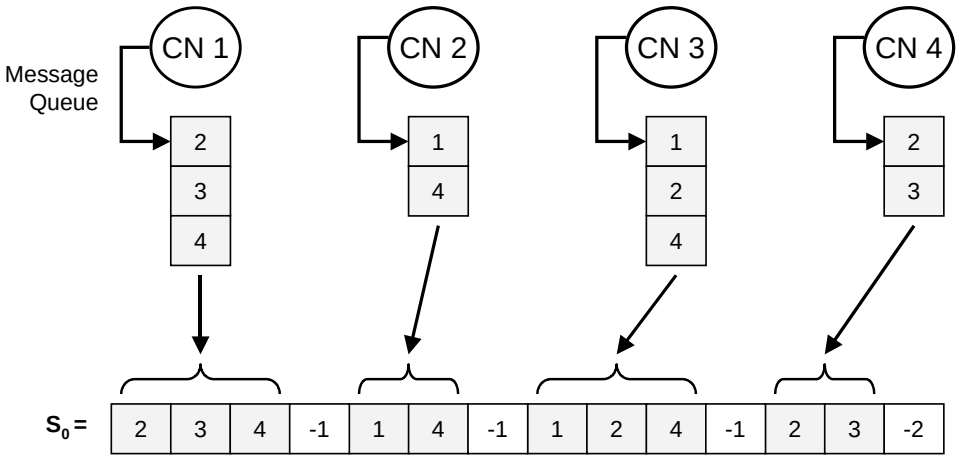

FIG. 3. Illustration of the encoding of a chromosome.

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

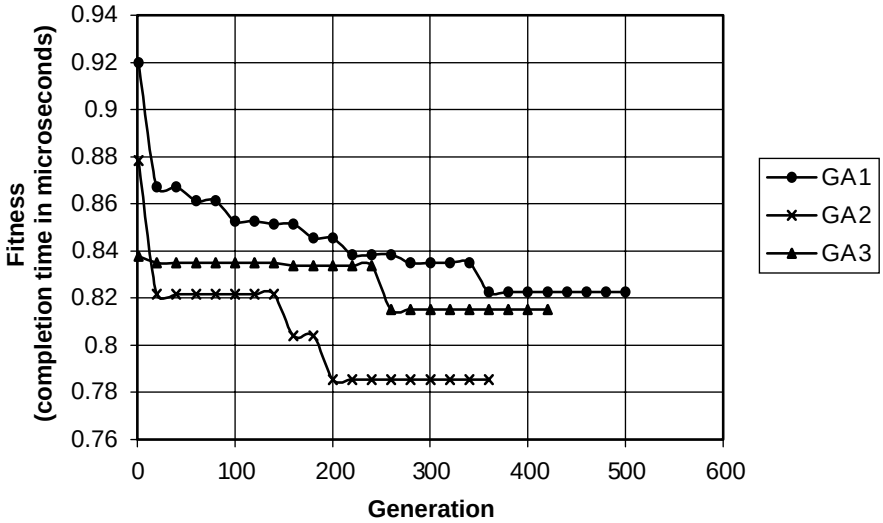

FIG. 4. Simulated communication completion time for the communication requirements for data redistribution after range processing and prior to processing in the pulse dimension. For GA 1, the crossover rate (P_{xover}) = 20% and the mutation rate (P_{mut}) = 4%. For GA 2, P_{xover} = 50% and P_{mut}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

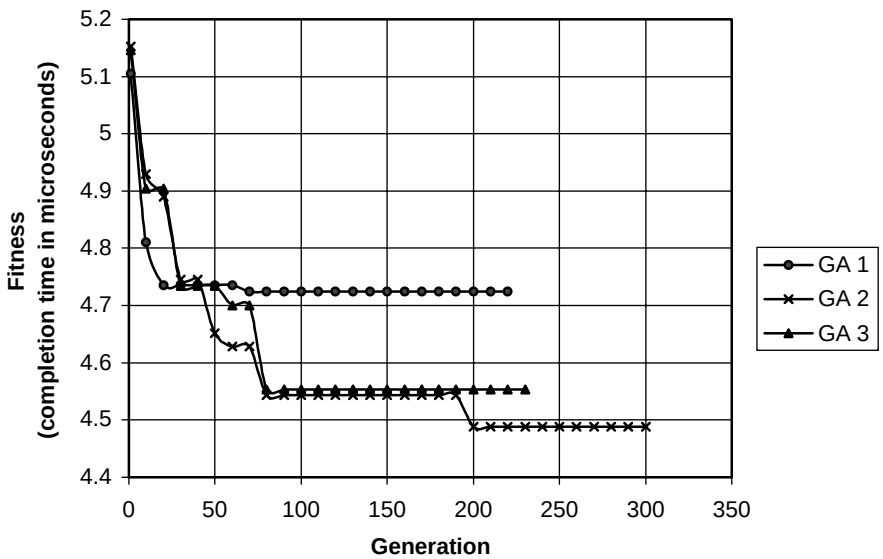

FIG. 5. Simulated communication completion time for the communication requirements for data redistribution after processing in the pulse dimension and prior to the final phase of processing. For GA 1, the crossover rate (P_{xover}) = 20% and the mutation rate (P_{mut}) = 4%. For GA 2, P_{xover} = 50%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

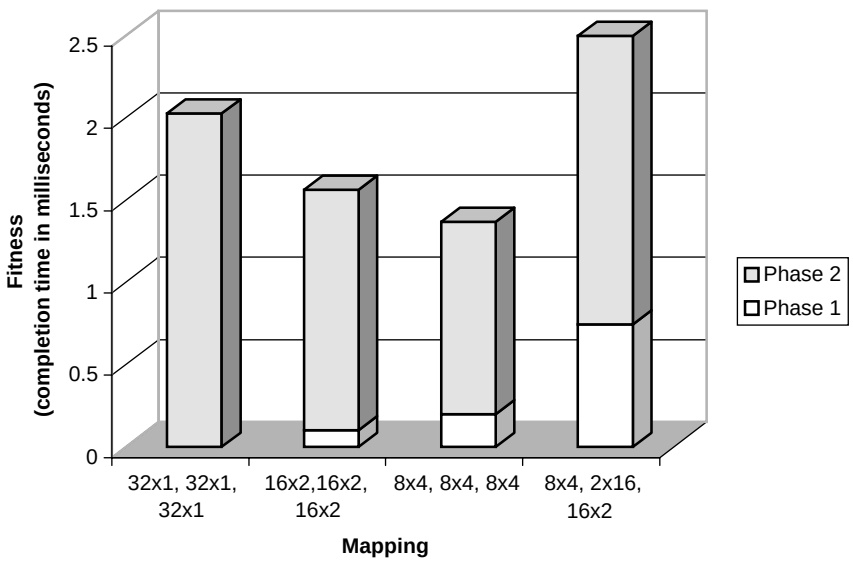

FIG. 6. Illustration comparing communication times associated with four different mappings schemes using the best GA-based schedules for each mapping. The STAP data cube is of size $32 \times 32 \times 32$ and there are 32 CNs in the multiprocessor system.

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

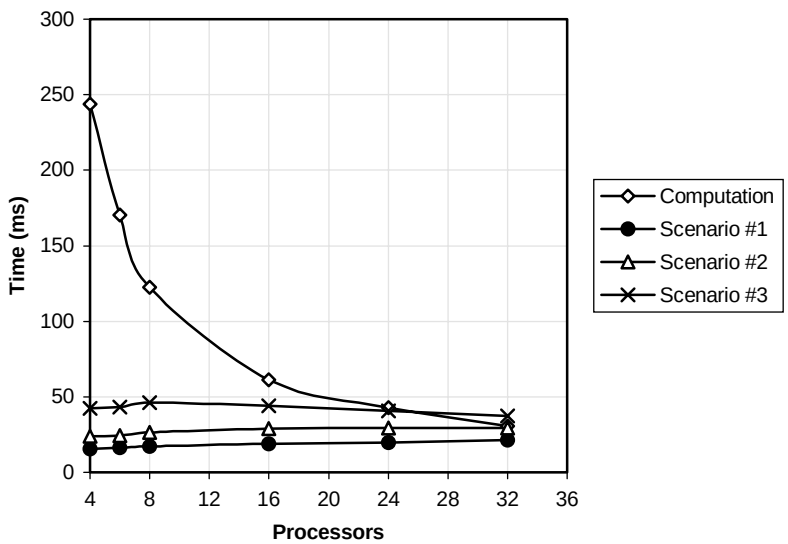

FIG. 7. Comparison of the computation time and communication times for three mapping/scheduling scenarios for a STAP data cube composed of 480 range bins, 64 pulses, and 16 channel elements.

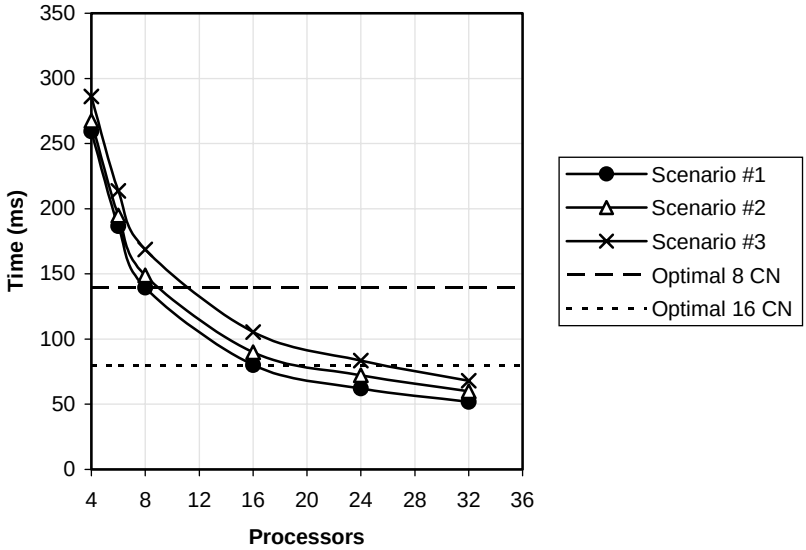

FIG. 8. Comparison of the overall completion times, including both computation and communication, for the three mapping/scheduling scenarios of Fig. 7. The optimal mapping/scheduling for 8-processor and 16-processor systems are indicated with dashed lines.

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

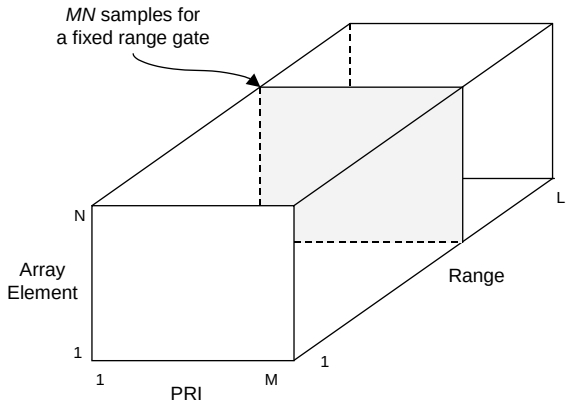

FIG. A1. The STAP CPI three-dimensional data cube (derived from [6]).

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 nth array element and the mth pulse at the lth 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 mth pulse and the lth 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 lth 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 re-distributed 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.

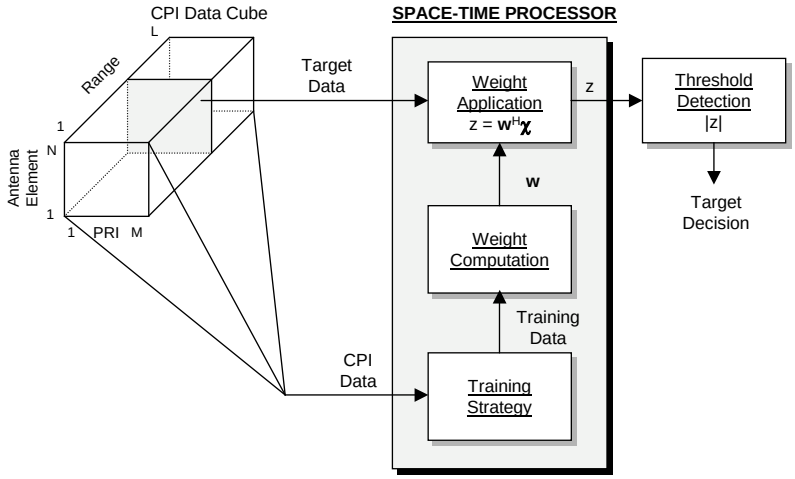

FIG. A2. Generic space–time adaptive processor (derived from [6]).

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

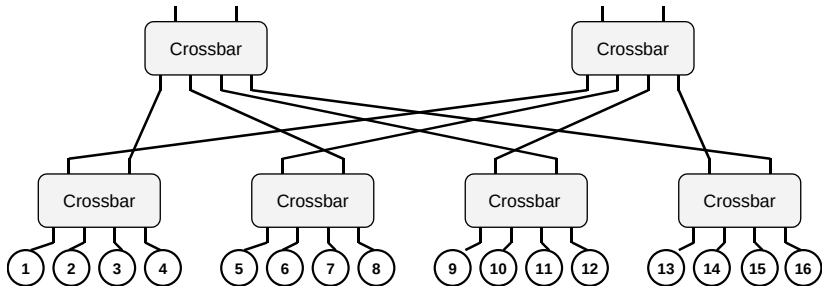

FIG. B1. Mercury RACE® fat-tree architecture configured with 16 CNs.

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

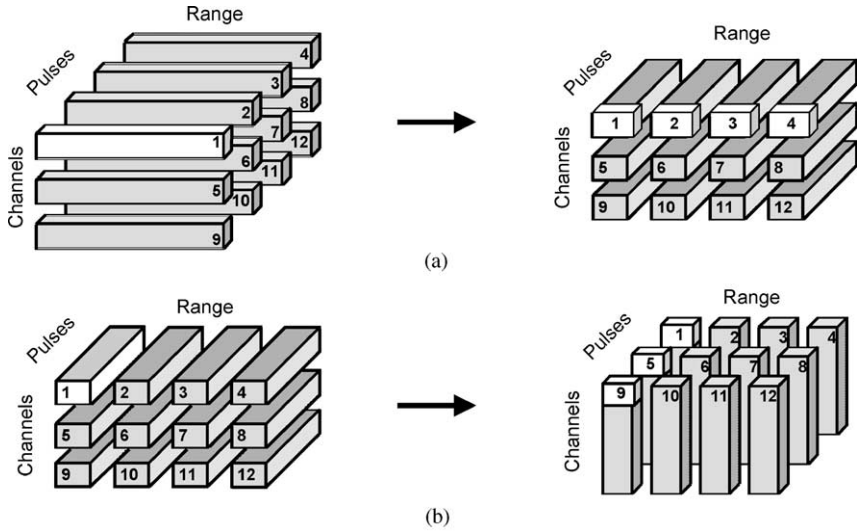

FIG. B2. Illustration of the sub-cube bar mapping technique for the case of 12 CNs. The mapping of the sub-cube bars to CNs defines the required data communications. (a) Example illustration of the communication requirements from CN 1 to the other CNs (2, 3, and 4) after completion of the range-dimensions 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 3×4 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)	$2 \rightarrow 6$, $3 \rightarrow 7$	0.41119	0.40013	2.69
II (contending)	$2 \rightarrow 4$, $3 \rightarrow 4$	0.84948	0.79608	6.29
III (contending)	$2 \rightarrow 3$, $3 \rightarrow 4$ $4 \rightarrow 2$	1.19329	1.19097	0.19
III (contending)	$2 \rightarrow 6$, $3 \rightarrow 6$ $6 \rightarrow 4$	1.28852	1.21279	5.88
IV	$5 \rightarrow \{6, 7, 8\}$ $6 \rightarrow \{5, 7, 8\}$ $7 \rightarrow \{5, 6, 8\}$ $8 \rightarrow \{5, 6, 7\}$	3.67124	3.40914	7.14
IV	All-to-all personalized communication involving CNs 2 through 8	9.52672	10.12816	−6.31
V	$2 \rightarrow \{4, 6, 8\}$ $3 \rightarrow \{5, 7\}$ $4 \rightarrow \{2, 6, 8\}$ $5 \rightarrow \{7, 3\}$ $6 \rightarrow \{8, 4, 2\}$ $7 \rightarrow \{5, 3\}$ $8 \rightarrow \{6, 4, 2\}$	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.