(2)

TR-457

DEVELOPMENT OF ADVANCED WTA ALGORITHMS FOR PARALLEL PROCESSING

FINAL REPORT

October, 1989

Prepared by

Dr. David A. Castañon

Submitted to:

Dr. Keith Bromley
Code 126
Office of Naval Research
800 North Quincy St.
Arlington, VA 22217-5000

Contract Number N00014-88-C-0718

ALPHATECH, Inc.
111 Middlesex Turnpike
Burlington, MA 01803
(617) 273-3388

ACKNOWLEDGEMENT

This effort is sponsored by the Strategic Defense Initiative Innovative Science and Technology Program managed through the agency of the Office of Naval Research (ONR) as Phase I of a Small Business Innovation Research (SBIR) Program grant. The COTR for this effort was Dr. Keith Bromley of ONR and the Naval Ocean Systems Center. We would like to thank Dr. Bromley for his direction during this effort.

We would like to acknowledge the invaluable assistance in this research provided by the Mathematics and Computer Science (MCS) Division of the Argonne National Laboratory (ANL). The ANL/MCS staff provided access, valuable training and consulting expertise in the use of the different parallel computers available at the Advanced Computer Research Facility (ACRF). In particular, we would like to acknowledge the contributions of Dr. James Boyle of ANL, Prof. Brian Smith of University of New Mexico, and Dr. Alan Wilson of Active Memory Technologies, who provided valuable guidance in the development of many of the parallel algorithms discussed in this study.

We would also like to thank Prof. Dimitri Bertsekas of MIT for assisting us with the development of the various asynchronous algorithms described in the papers, for providing efficient state-of-the-art sequential implementations of the AUCTION algorithm, and for numerous discussions concerning approaches for effective parallelization of these algorithms.

CONTENTS

LIST OF FIGURES

SECTION 1
OVERVIEW

1.1 INTRODUCTION

The objective of weapon-target assignment (WTA) in a ballistic missile defense (BMD) system is to determine how defensive weapons should be assigned to boosters and re-entry vehicles in order to maximize the survival of assets belonging to the U.S. and allied countries. The implied optimization problem requires consideration of a large number of potential weapon target assignments in order to select the most effective combination of assignments. The resulting WTA optimization problems are among the most complex encountered in mathematical programming [1,2]. Indeed, simple versions of the WTA problem have been shown to be NP-complete [3,4], implying that the computations required to achieve optimal solutions grow exponentially with the numbers of weapons and targets considered in the solution.

The computational complexity of the WTA problem has motivated the development of heuristic algorithms that are not altogether satisfactory for use in Strategic Defense Systems (SDS). Some special cases of the WTA problem are not NP-complete and can be solved using standard optimization algorithms such as linear programming [5] and maximum marginal return algorithms [6,7]; these algorithms enjoy low computational requirements and therefore have been adopted as heuristics for solving more general WTA problems. However, experimental studies [2,8,9,10] have demonstrated that these heuristic algorithms lead to significantly suboptimal solutions for certain scenarios.

In order to address this deficiency, the Strategic Defense Initiative Office initiated several research efforts to develop efficient, near-optimal boost-phase and midcourse-phase WTA algorithms for directed energy weapons [11] and kinetic energy weapons [1,2,9,11,12,13]. These programs developed advanced optimization-based WTA algorithms

which achieved improved performance over the existing SDS WTA algorithms, but which required increased computation in order to be implemented as part of a real-time system. Among the most successful WTA algorithms developed was the ILINE algorithm [8,9] and its subsequent extensions [2,10] for assignment of kinetic kill interceptors. The merits of the ILINE algorithm for Boost and Post-Boost WTA were established in the Air Force's Space-Based Experimental Version program [10] sponsored by ESD; in this program, various candidate WTA algorithms were studied, and ILINE was selected and implemented as the superior algorithm for performance of the WTA function. The ILINE algorithm was made available to the SDI Battle Management community, and was evaluated in both the Air Force's [10] and the Army's [14] BM/C3 Experimental Version programs for weapon-target assignment.

The major limitation of the ILINE algorithm for SDS WTA is the computation time required for selecting near-optimal weapon-target assignments in scenarios with large numbers of interceptors and targets. For Boost-Phase WTA, the ILINE algorithm may have to solve WTA problems with 800-1000 targets in the order of 1-3 seconds in order to fit within a reasonable fraction of the overall real-time planning cycle. For Midcourse WTA, the ILINE algorithm may be imbedded into a dynamic Battle Planning algorithm which requires 10-100 iterative applications of the ILINE algorithm (the extra iterations are required for adaptive preferential defense and predictive battle planning, as discussed in [2]). Each of these iterations require the application of ILINE for WTA problems with up to 10,000 targets; the overall dynamic Battle Planning algorithm computations must be completed within 2-10 seconds in order to fit within a reasonable fraction of the overall real-time midcourse planning cycle.

As a point of reference, the results of [2] indicate that the computation time for a single application of the ILINE algorithm for problems involving up to 4500 targets will require about 300 seconds of CPU time on a .5 MIPS sequential processor, and that the computation time grows near-linearly with the number of targets. This extrapolates to over 600 seconds of

computation time for each application of the ILINE algorithm for 10,000 targets. Thus, we need to achieve up to four orders of magnitude reduction in computation time from the sequential computation time on a .5 MIPS processor.

These lofty goals appear beyond the scope of single-processor technology in the near-future. However, the structure of the ILINE algorithm suggested that significant reductions in computation time could be achieved through parallel processing, so that a combination of processor technology improvements and parallel processing could be used to achieve the desired real-time computation goals. The purpose of the phase one research was to demonstrate the potential reductions in the computation time of the ILINE algorithm which can be achieved on different multiprocessor architectures by developing and benchmarking different parallel variations of the ILINE algorithm on commercial multiprocessors. The resulting parallel WTA algorithms provide the basis for real-time WTA algorithm development using multiprocessor architectures; furthermore, the benchmarking results can be used to identify characteristics of desirable computer architectures for efficient execution of WTA algorithms.

1.2 OVERVIEW OF PHASE 1 RESULTS

The basis of the ILINE WTA algorithm is to solve a sequence of linear assignment problems [15] using Bertsekas' AUCTION [16,17] algorithm (as extended by Bertsekas and Castañon [18]). Each application of ILINE requires the solution of 4-6 assignment problems using AUCTION. Depending on the size of the problem, over 95% of the overall ILINE computation time is spent in the AUCTION algorithm. Thus, the key to developing parallel versions of the ILINE algorithm is to develop parallel versions of the underlying AUCTION algorithm. The AUCTION algorithm is a recently-developed optimal algorithm for the solution of classical assignment problems (finding an optimal one-to-one match from n persons to n objects in order to maximize the sum of the individual benefits associated with each person-object match). Assignment problems are important in many aspects of SDS besides weapon-

assignment; these additional applications including single-sensor, multiple frame association for multiobject tracking and multi-sensor correlation.

The AUCTION algorithm has been shown to be a very effective sequential assignment algorithm, substantially outperforming its rivals for sparse problems. The algorithm operates like an auction, whereby at each iteration, unassigned persons bid simultaneously for objects thereby raising their prices. Objects are then awarded to the highest bidder. The AUCTION algorithm was also designed with an orientation towards parallel implementation, making it an ideal starting point for our investigations.

Through analysis of the structure of the AUCTION algorithm, we identified two different levels where parallel processing could be used to speed up the computations: a medium-grained level and a fine-grained level. The medium-grained level (referred to as the Jacobi level, due to its similarity to the iterative Jacobi algorithm for recursive solution of linear equations) consisted of parallel processing multiple weapon-target pairs simultaneously, while the fine-grained level (referred to as the Gauss-Seidel level) consisted of processing multiple targets for a single weapon simultaneously. Ideally, an effective parallel algorithm would combine the potential speedups achievable at each level in a multiplicative fashion.

In order to explore the potential for parallel implementation on different multiprocessor architectures, we developed and implemented the following versions of the AUCTION algorithm on different multiprocessor architectures at the Advanced Computing Research Facility (ACRF) at Argonne National Laboratory[1]

1. Two versions of Jacobi AUCTION on the Encore Multimax using sparse data structures (one synchronous, one asynchronous)
2. Gauss-Seidel AUCTION on the Encore Multimax using sparse data structures
3. Three versions of Hybrid AUCTION on the Encore Multimax using sparse data structures (one synchronous, two asynchronous)
4. Gauss-Seidel AUCTION on the Alliant FX/8 using sparse data structures

[1] Access to the ACRF was arranged through SDIO sponsorship by Capt. S. Johnson of SDIO.

5. Gauss-Seidel AUCTION on the Alliant FX/8 using dense data structures

6. Gauss-Seidel AUCTION on the Connection Machine CM-2 using dense data structures

7. Gauss-Seidel AUCTION on the DAP 510 using dense data structures

Figure 1-1 illustrates the speedups obtained from medium-grained parallelization (Jacobi parallelization) for several 800 and 1000 target assignment problems with different feasible intercept densities (average percentage of total targets which can be attacked by each weapon), as established by the shared-memory implementation of the Jacobi AUCTION algorithm on the Encore Multimax. As the results of Fig. 1-1 indicate, the maximum speedup achievable by Jacobi parallelization is approximately 4, independent of the density of feasible intercepts (and also nearly independent of the number of targets available!). This important limitation is due to the incremental nature of the AUCTION algorithm (described in greater

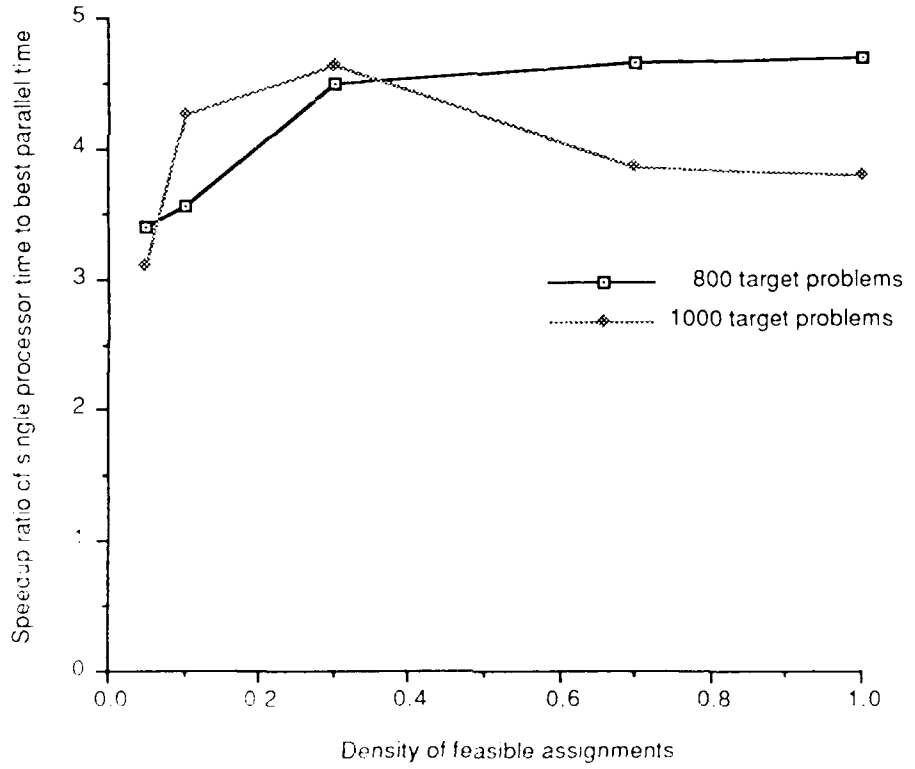

Figure 1-1. Speedup of parallel Jacobi AUCTION algorithm over the single-processor algorithm on the Encore Multimax as a function of the density of feasible interceptor assignments for problems with 800 and 1000 targets.

detail in Appendix A), whereby a complete assignment of interceptors is built incrementally using an iterative approach. Eventually, the number of unassigned weapons is smaller than the number of available processors, which effectively limits the potential for Jacobi parallelization.

Figure 1-2 illustrates the potential speedups available from fine-grained (Gauss-Seidel) parallelization for several 1000 target assignment problems. In order to evaluate speedup for SIMD architectures (such as the CM-2 and DAP 510), we compare the performance of the algorithms implemented on these architectures with the sequential performance of the AUCTION algorithm on a single processor of the Encore Multimax. As Fig. 1-2 illustrates, SIMD architectures are particularly effective for exploiting fine-grained parallelism. In particular, the DAP 510's architecture allows for speedups of over 60 when compared to the performance of the AUCTION algorithm on a single processor of the Encore Multimax. However, the speedup achievable from Gauss-Seidel parallelization depends explicitly on the

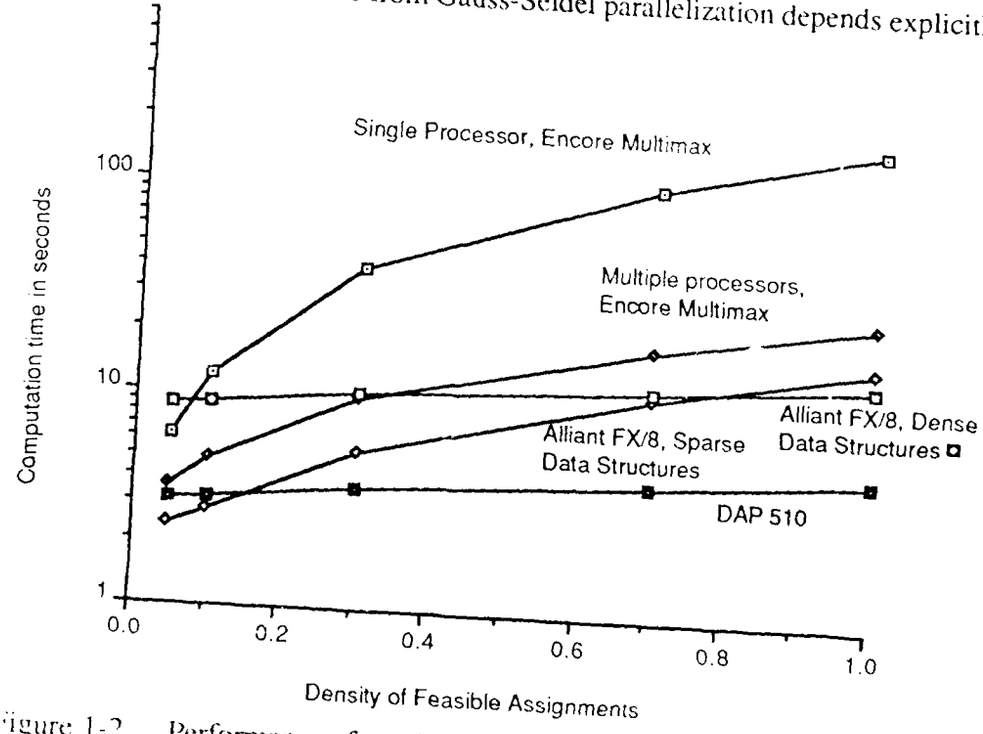

Figure 1-2. Performance of parallel Gauss-Seidel AUCTION algorithms on different parallel processors for 1000 target assignment problems as a function of the fraction of targets which can be assigned to each weapon.

density of feasible interceptor assignments (unlike the speedups obtained from Jacobi parallelization). Specifically, the speedups increase with the average number of targets which can be attacked by each interceptor. Thus, these speedups will increase as the number of targets grows as well as with increased feasible intercept density for a fixed number of targets.

Figure 1-2 also illustrates an interesting tradeoff between the use of sophisticated data structures and the speed of computation. For problems where the density of feasible interceptor-target assignments is less than one (i.e. each interceptor can only reach a fraction of the available targets), sparse data structures can be used to keep track only of the feasible intercepts for each weapon. However, such a representation hinders efficient computation in SIMD architectures with limited communications, because of the required data movements among processors. The most efficient SIMD implementations are those which avoid communications; however, this often requires alignment of the feasible intercept data, precluding the use of sparse data structures. On advanced MIMD processors such as the Encore Multimax and the Alliant FX/8, interprocessor communications are less costly, so that efficient implementations of the AUCTION algorithm using sparse data structures are possible. Note in particular the differences in performance of the Alliant Gauss-Seidel AUCTION algorithms using sparse and dense data structures.

An important result which was established in the research was the potential for combination of the Jacobi and Gauss-Seidel speedups. The hybrid AUCTION algorithm on the Encore Multimax was one such implementation, using two processors at a Jacobi level and a variable number of processors at the Gauss-Seidel level. Note, however, that the speedup of the hybrid AUCTION algorithm in Fig. 1-2 is far less than a multiplicative combination of the Gauss-Seidel and Jacobi AUCTION algorithm speedups. The principal limitation in this combination is the time required for synchronization of the various processors. Figure 1-3 illustrates the growth in the synchronization time of the hybrid AUCTION algorithm as the number of processors is increased.

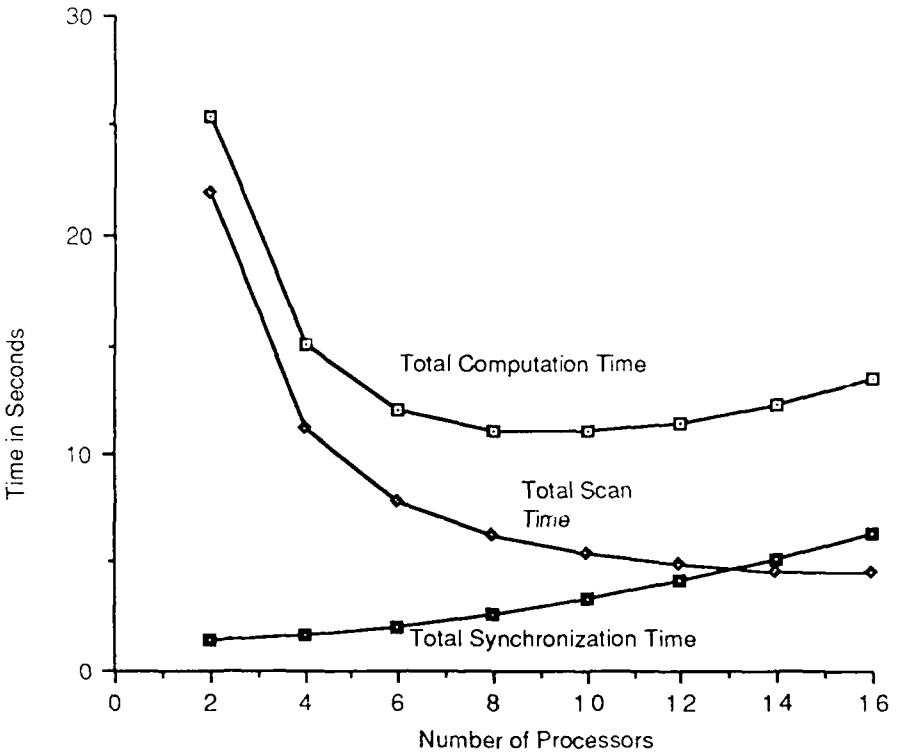

Figure 1-3. Performance of the synchronous Hybrid AUCTION algorithm as a function of the number of processors for 1000 target, 20% dense assignment problem.

In order to reduce the overall synchronization time of the hybrid AUCTION algorithm, we designed a new asynchronous version of the hybrid AUCTION algorithm and proved its convergence to a correct solution. We also implemented this asynchronous hybrid AUCTION algorithm and verified that significant performance improvements were possible over the synchronous hybrid AUCTION algorithm. Figure 1-4 illustrates the performance of the synchronous and asynchronous Hybrid AUCTION algorithms on the Encore Multimax for several 1000 target problems. As the results indicate, the asynchronous algorithms permit a more efficient utilization of large numbers of processors, by reducing the synchronization overhead, leading to significant reductions (nearly 50%) in computation time.

The results of Figs.1-1, 1-2, 1-3 and 1-4 illustrate the extent to which the research goals of phase I have been met. In essence, our results establish that significant speedups are possible for WTA algorithms using multiprocessor architectures; based on the expected size of

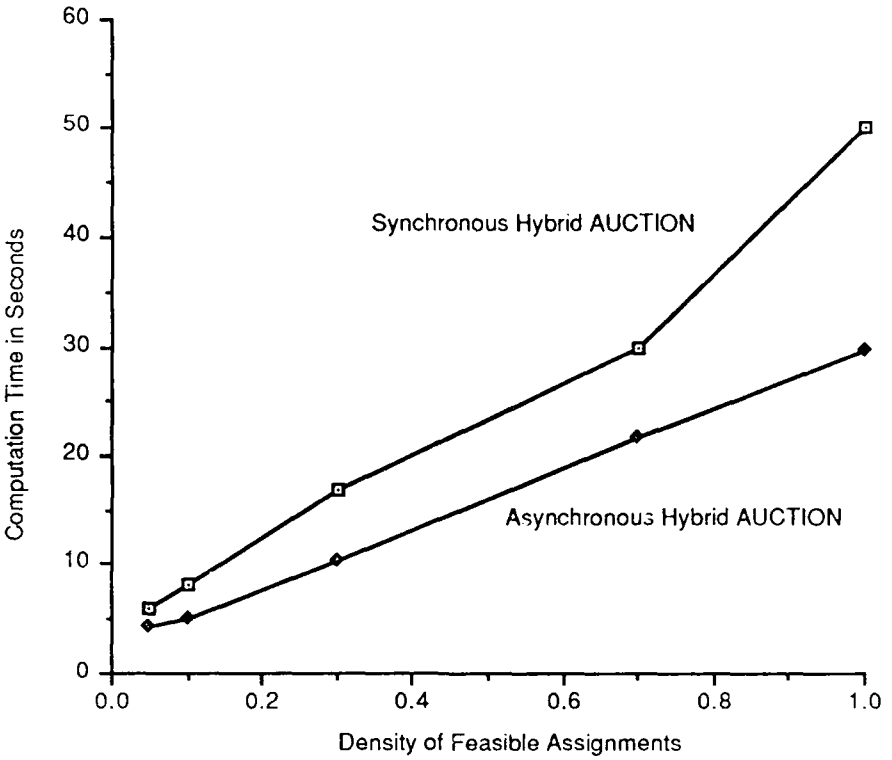

Figure 1-4. Performance of synchronous Hybrid AUCTION and asynchronous Hybrid AUCTION I algorithms on 1000 target assignment problems of varying density.

the scenarios, proper choice of multiprocessor architecture and parallel algorithm implementation ought to reduce the overall WTA computation requirements to fit as part of real-time Battle Management processing software.

The results of this research suggest that a superior architecture for assignment problems using the AUCTION algorithm must be able to exploit both Jacobi and Gauss-Seidel parallelism. Exploitation of Gauss-Seidel parallelism is best done by SIMD processors capable of simultaneous associative processing for vectors of significant length (such as the DAP 510). Exploitation of Jacobi parallelism is best done by MIMD processors with flexible communications structure, capable of fast interprocessor communication. Our prototype algorithm benchmarks indicate that architectures which successfully combine these features should reduce the computation requirements of the AUCTION algorithm by two orders of magnitude when compared to a Von Neumann architecture for problems involving 1000

targets. For larger problems involving 10,000 targets, the potential speedups from Gauss-Seidel parallelization should increase by an order of magnitude, leading to reductions of the computation requirements of the AUCTION algorithm by nearly three orders of magnitude. These reductions approach the real-time computation requirements (four orders of magnitude reduction to the .5 MIPS sequential processing time) discussed earlier; coupled with advances in individual processor technology, the parallel algorithms (implemented in appropriate multiprocessor architectures) can be projected to meet the required real-time deadlines.

1.3 REVIEW OF RELATED PARALLEL ALGORITHM WORK

Development of parallel WTA algorithms has been recognized as a difficult problem; in essence, the nature of the WTA problem requires that a global search among many alternatives be conducted in order to obtain a set of near-optimal assignments. The global nature of this processing makes efficient distribution among multiple processors a difficult task. Indeed, several early efforts at developing parallel WTA algorithms (based on the AUCTION algorithm) conducted at Los Alamos National Laboratory [19] and Argonne National Laboratory [20] obtained very limited speedups using shared-memory MIMD architectures. A similar study conducted at the Jet Propulsion Laboratory (JPL) of the California Institute of Technology [21] using a heuristic WTA algorithm implemented on a message-passing MIMD multiprocessor achieved no significant speedup.

In addition to SDS-sponsored efforts on parallel algorithms, there has been a set of recent research results on the development of parallel algorithms for assignment problems. Kempa, Kennington and Zaki [22] have reported on the parallel performance of the AUCTION algorithm on dense assignment problems when implemented on the Alliant FX/8. The particular variation of the AUCTION algorithm which they implemented addressed only fully dense assignment problems, and did not include sparse data structures or address the issue of algorithms for different multiprocessor architectures. In their implementation of the Jacobi AUCTION algorithm on the Alliant FX/8, they used a synchronous hybrid algorithm which

uses the vector processing capability of each of the Alliant's processors to scan the admissible objects for each bid, and uses multiple processors to process several bids in parallel. This hybrid algorithm is similar in spirit to the recommended approach of combining the SIMD and MIMD speedups. However, their hybrid algorithm only achieved a speedup of near 8 for 1000 person assignment problems when compared with the single processor version of the same algorithm because of the short length of the vector processors on the Alliant FX/8. Furthermore, they did not compare their parallel algorithm results with an efficient sequential algorithm implementation, so they may have overestimated the true speedups achieved on the Alliant FX/8.

Recently, Balas, Miller, Pekny and Toth [23] have developed a synchronous parallel assignment algorithm based on a successive shortest path algorithm (rather than the AUCTION algorithm) and have implemented it successfully on a 14-processor Butterfly Plus computer. Their algorithm is the extension of Jacobi parallelization for successive shortest path methods, since it handles the assignment of multiple weapons in parallel. However, the synchronization required in the algorithm limits the effective speedups of the parallel shortest path algorithm to under two for problems with 1000 persons. Unlike the AUCTION algorithm theory described subsequently, a theory of asynchronous assignment algorithms based on successive shortest paths is not available at this time.

Kennington and Wang [24] have also reported on parallel implementation of a successive shortest path algorithm (the JV algorithm) for dense assignment problems on the 8-processor Sequent Symmetry S81. In their implementation, multiple processors are used to construct shortest paths from a single unassigned person. This is the extension of the Gauss-Seidel parallelization for successive shortest path methods. For problems with 1000 persons, Kennington and Wang obtained a speedup factor of 3.7 using 8 processors on the Sequent Symmetry.

For SIMD architectures, Zenios and Phillips [25] have experimented with variations of the Jacobi AUCTION algorithm on the Connection Machine CM-2. By spreading the

information corresponding to potential individual assignments over large numbers of processors, they are able to implement a SIMD variation of our Hybrid AUCTION algorithm. However, the performance of their implementation has been disappointingly slow (even though it was implemented in the C-Paris assembly language); for problems involving 1000 persons, their computation time on the CM-2 achieves a speedup factor of under 3 when compared with the sequential computation time of our Gauss-Seidel AUCTION algorithm on a single processor of the Encore Multimax!

The results presented in this report extend and unify a number of the above studies using the AUCTION algorithm. By studying carefully the structure of the AUCTION algorithm, we have identified superior designs for parallel algorithms which can be tailored to each multiprocessor architecture. Our comparative study of different implementations of the Gauss-Seidel AUCTION on different multiprocessor architectures provides interesting insights into the specific advantages and disadvantages of each multiprocessor architecture, rather than reflect on the specifics of any one implementation on a single architecture. Indeed, our results suggest that many of the speedups obtained in previous results can be attributed to poor implementation of the sequential algorithms. In contrast, we have used the most efficient variations of the sequential AUCTION algorithms for our benchmarks; these variations were developed in cooperation with Prof. D. Bertsekas of MIT, the originator of the AUCTION algorithm.

Furthermore, the theory and benchmarking results developed for the asynchronous variation of the Hybrid AUCTION algorithm provides the basis for the design of asynchronous AUCTION algorithms which will operate efficiently with greatly reduced communications and synchronization. These asynchronous algorithms should be suitable for implementation in distributed memory MIMD architectures or in more advanced hybrid architectures which combine desirable features of SIMD and MIMD architectures.

1.4 IDEAS FOR FOLLOW-ON RESEARCH

The results obtained under this phase I research study provide ample evidence that, with a proper combination of parallel WTA algorithm and multiprocessor architecture, development of real-time Battle Planning software which incorporates advanced WTA algorithm technology is a feasible goal for realistic problem sizes. However, the phase I research has focused only on parallel implementation of the core WTA algorithm (ILINE); in order to develop real-time Battle Planning software, this core WTA must be integrated successfully with parallel algorithms for other Battle Planning functions (such as computation of feasible intercepts) or within recursive Battle Planning algorithms such as the adaptive preferential defense algorithms or the anticipative algorithms discussed in [2].

One potential direction for continuation of this research into Phase II would be to extend the Phase I results and develop an integrated parallel Battle Planning algorithm on an advanced multiprocessor architecture which incorporates the various Battle Planning functions which interact with WTA. This Battle Planning algorithm could be focused either on Boost and Post-Boost defense (as in [10]) or on Midcourse and Terminal defense (as in [2], [14]). The choice of problem area will depend on the criticality of parallel processing technology for achieving real-time performance in this problem; the Boost and Post-Boost problem may have more modest computation requirements because of its shorter time scale and smaller number of targets than the corresponding Midcourse and Terminal problems, but the real-time computation cycle may be shorter. The goal of such a Phase II effort would produce a prototype Battle Planning algorithm design (based on advanced WTA algorithm technology) and associated software which could be used as the basis for Battle Manager software and processor design effort. Part of this effort would involve selection of an appropriate multiprocessor architecture, as well as development of the appropriate parallel Battle Planning software.

A second direction for Phase II continuation would involve extension of the Phase I work on the core parallel ILINE algorithms to produce advanced parallel WTA algorithms capable of addressing important requirements such as adaptive preferential defense, anticipative Battle Planning, nuclear interference avoidance and Battle Planning with discrimination uncertainty. In [2], a theoretical structure was presented for incorporating the ILINE algorithm into more general recursive WTA algorithms capable of addressing these important SDS requirements. Furthermore, extensive testing with sequential versions of these algorithms indicated that significant SDS effectiveness improvements would result from the use of these advanced algorithms. The goal of this Phase II effort would be to extend the Phase I efforts in parallel designs for the core ILINE algorithm in order to produce working prototypes of these advanced WTA algorithms which can be executed in real time on commercial parallel computers. Such prototypes can be incorporated into future Command Center designs for SDS.

1.5 ORGANIZATION OF THIS REPORT

The remainder of this report is of a technical nature, and serves to document the advances accomplished under phase I of this research. In Section 2, we describe the variation of the WTA problem which is the focus of this study, and discuss the ILINE algorithm. In Section 3, we describe the design of the various synchronous parallel AUCTION algorithms which were implemented on different multiprocessors; we also describe the benchmarks obtained on the different multiprocessors. In Section 4, we overview the theory and design of the asynchronous parallel AUCTION algorithms implemented on the Encore Multimax, and discuss the benchmark results obtained from our implementations. Appendix A contains a discussion of the theory of the AUCTION algorithm, including some new results concerning the validity of an asynchronous variation of the algorithm. These results are part of a paper [26] which will be submitted for publication.

SECTION 2

THE ILINE ALGORITHM FOR WTA

In this section, we provide a mathematical description of the WTA problem, and discuss the ILINE algorithm for obtaining a near-optimal solution of this problem.

2.1 MATHEMATICAL DESCRIPTION OF THE WTA PROBLEM

Consider the following target-oriented weapon-target assignment problem. The objective is to minimize the weighted expected leakage of targets through the defense

$$(NP) \qquad \min_{x_{ij}} \sum_{i=1}^{T} V_i \prod_{j=1}^{W} (1 - p_{ij})^{x_{ij}} \tag{2-1}$$

where T is the number of targets, W is the number of weapon farms/platforms, x_{ij} is the number of interceptors assigned from weapon farm/platform j to target i, p_{ij} is the probability of kill of an interceptor assigned from weapon farm/platform j to target i, and V_i is the value associated with failure to destroy target i. The constraints on problem (NP) are

$$\sum_{i=1}^{T} x_{ij} \leq M_j \tag{2-2}$$

for all weapon farm/platforms j, and to the constraint that interceptors are assigned in integer quantities; that is,

$$x_{ij} \quad \varepsilon \quad \{ 0, 1, ..., M_j \} \tag{2-3}$$

The problem NP subject to the constraints of Eqs. 2-2, 2-3 is a nonlinear integer programming problem; a recent result by Lloyd and Witsenhausen [3] established that this

problem NP-complete[1]. A simpler version of this problem introduces the additional constraint

$$\sum_{j=1}^{W} x_{ij} \leq 1 \tag{2-4}$$

With this additional constraint, problem (NP) becomes equivalent to the following problem:

$$\text{(LP)} \qquad \max_{x_{ij}} \sum_{i=1}^{T} V_i \, p_{ij} \, x_{ij} \tag{2-5}$$

subject to the constraints of Eqs. 2-2, 2-3, 2-4. Problem (LP) is a linear integer programming problem of network type, for which efficient algorithms exist.

Figure 2-1 illustrates the structure of the resulting linear integer programming problem. This type of linear program is known as a transportation problem. In essence, a maximizing set of flows x_{ij} must be found between a set of source nodes (representing the targets in our

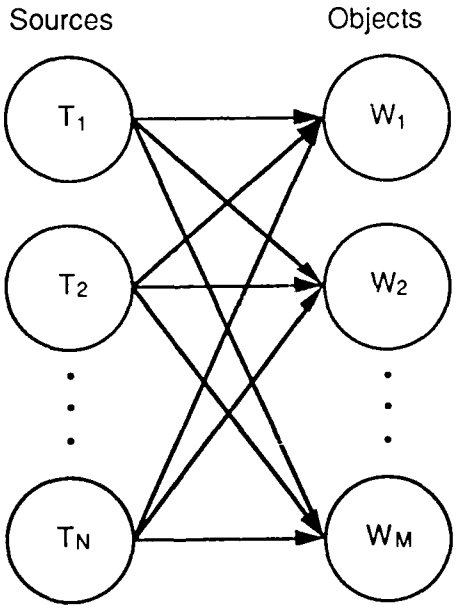

Figure 2-1. The structure of transportation problems divides the graph into two sets of nodes (target nodes T_i and farm/platform nodes W_j) with arcs in between.

[1] This implies that the time required to find an optimal solution is likely to grow exponentially in the numbers of weapons and targets in the problem.

set of flows x_{ij} must be found between a set of source nodes (representing the targets in our problem) and a set of sink nodes (representing the interceptor platforms and farms). The overall flows must satisfy the conservation of flow constraints (cf. Eqs. 2-2 to 2-4), so that the overall flow out of a target source cannot exceed 1, and the overall flow into weapon object j cannot exceed its available interceptor inventory M_j. When all of the available weapon inventories M_j are equal to one, the resulting optimization problem is known as an assignment problem; in this case, each interceptor is modeled as a separate weapon platform.

The structure of the constraints of transportation problems[1] is such that the integrality constraints of Eq. 2-3 can be relaxed to allow for fractional interceptor assignments x_{ij}. That is, Eq. 2-3 can be replaced by the constraints

$$0 \leq x_{ij} \leq M_j \tag{2-6}$$

With this relaxation, an optimal solution can be found for which all of the x_{ij} are integer. This allows for the development of efficient algorithms by using the duality theory of linear programming. One such efficient algorithm is the AUCTION algorithm developed by Bertsekas [16] for assignment problems and extended Bertsekas and Castañon [18] for transportation problems. We overview the AUCTION and ILINE algorithms in the next subsections.

2.2 DESCRIPTION OF THE ILINE ALGORITHM

The basis of ILINE is to solve Problem (NP) by a successive linearization procedure, whereby Problem (NP) is approximated at each stage by Problem (LP). The solution of Problem (LP) is computed using AUCTION, and a fixed number of the assignments are implemented. Based on these assignments, a new linearized version of Problem (NP) is generated (a new Problem LP), and the procedure is repeated until all interceptors have been assigned. Figure 2-2 illustrates the structure of the ILINE algorithm. The key computation-

[1] This structure is known as unimodularity [15].

intensive step is the solution of Problem (LP), which must be performed several times in the procedure. The AUCTION algorithm provides a practical approach for repeated solutions of Problem (LP), by reusing most of the previous solution as an initial point for obtaining a new solution.

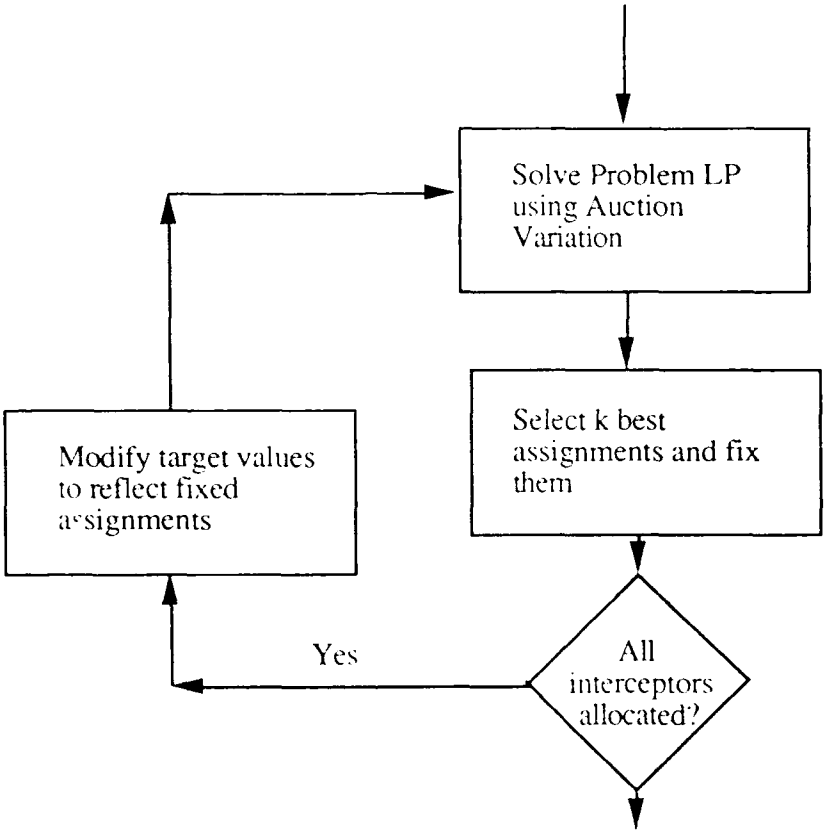

Figure 2-2. Structure of the ILINE Algorithm

At each iteration of the ILINE algorithm, a subset of interceptor assignments x^*_{ij} have already been fixed. Based on these fixed assignments, the ILINE algorithm computes an expected probability of survival for each target i, as

$$P_S(i) = \prod_{j=1}^{W} (1 - p_{ij})^{x^*_{ij}} \qquad (2\text{-}7)$$

The linearization of Problem NP is based on using the expected probabilities of survival for each target, resulting in the following optimization problem:

$$\text{(SLP)} \qquad \max_{x_{ij}} \sum_{i=1}^{T} V_i P_{S(i)} p_{ij} x_{ij} \tag{2-8}$$

subject to

$$\sum_{i=1}^{T} (x_{ij} + x^*_{ij}) \leq M_j \tag{2-9}$$

and the constraints of Eqs. 2-3 and 2-4.

Denote the optimal solution to Problem (SLP) by x^o_{ij}. For each pair ij, the ILINE algorithm ranks the nonzero assignments ($x^o_{ij} > 0$) in nonincreasing order according to the marginal return $p_{ij} P_S(i) V_i$. The top k assignments according to this order are selected and added to the corresponding permanent assignments x^*_{ij}. If additional interceptors remain to be assigned, then a subsequent iteration of the above procedure is conducted.

The key operation of the ILINE algorithm is the optimal solution of the linearized Problems (SLP). The algorithm used inside of ILINE is a variation of the AUCTION algorithm, discussed next.

2.3 THE AUCTION ALGORITHM

The original AUCTION algorithm was described by Bertsekas [16] for assigning individual bidders (corresponding to interceptors or targets) to individual objects (corresponding to targets or interceptors). The theory of the AUCTION algorithm is discussed in detail in Appendix A. In this subsection, we briefly overview the computations of the AUCTION algorithm.

The classical assignment problem consists of finding a one-to-one match between a list of n persons and n objects such that the sum of the benefits of the individual matches is maximized. Denote the individual benefits of assigning person i to object j as a_{ij}. Then, the classical assignment problem can be stated as follows.

$$\max_{x_{ij}} \sum_{i=1}^{n} a_{ij} x_{ij} \tag{2-10}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1, \; j = i, \ldots, n; \tag{2-11}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, \ldots, n; \tag{2-12}$$

$$x_{ij} \in \{0,1\}, \; i = 1, \ldots, n; \; j = 1, \ldots, n. \tag{2-13}$$

Note the similarity between the objective in the WTA problem of Eq. 2-8 and the objective in the classical assignment problem: the benefit a_{ij} of assigning interceptor j to target i is given by $p_{ij} V_i P_S(i)$. Note also that the constraints in Eqs. 2-12 and 2-13 require that an equal number of interceptors and targets be present. This represents no loss in generality, since targets with value 0 or interceptors with 0 probability of kill can be introduced to balance an uneven assignment problem.

Ideally, the maximum benefit is obtained when each person i is assigned to an object j offering maximal individual benefit a_{ij}. However, such an assignment is likely to violate the constraints in Eq. 2-12 which require that each target be assigned an interceptor. In order to resolve such conflicts, the AUCTION algorithm assigns a *price* p_j to each object j which reflects the degree to which an object is in demand by different persons. The key observation in the AUCTION algorithm is that there exists a set of prices p_j such that *the optimal assignment* has the property that each person i is assigned to the object j(i) which offers the *highest net profit* $a_{ij(i)} - p_{j(i)} = \max_j (a_{ij} - p_j)$. This is a consequence of the celebrated duality theorem of linear programming [5]. The AUCTION algorithm consists of a search for the right

level of object prices p_j; this search takes the form of an auction, where unassigned persons "bid" for objects and raise the prices of the objects accordingly.

The AUCTION algorithm can be described in terms of a sequence of iterations. During each iteration, the price p_j of some object j is raised; in addition, tentative assignments of objects to persons which have offered the highest prices for those objects are made. Each iteration can be described in terms of two distinct phases:

a. *Bid Phase*: In this phase, a subset I of persons which do not have a tentative assignment (unassigned persons) to any objects will offer bids for objects. Each person i computes his bid as follows, based on the current object prices p_j.

   1. Person i must determine the object j(i) offering the maximum net profit based on the current prices; that is,

   $$j(i) = \arg\max_j \{a_{ij} - p_j\}$$

   2. Person i must determine the price level b(i) which it will bid for object j(i); this price level is determined by computing the two highest net profit levels as follows:

   $$v(i) = \max_j \{a_{ij} - p_j\}$$

   $$w(i) = \max_{j \neq j(i)} \{a_{ij} - p_j\}$$

   $$b(i) = p_{j(i)} + v(i) - w(i) + \varepsilon$$

   where $\varepsilon > 0$ is a positive parameter, chosen small enough to guarantee convergence to an optimal solution.

b. *Auction Phase*: In this phase, each object j which received a bid in the Bid Phase selects the highest bid and is tentatively assigned to the person i which offered the highest bid. If the object was previously assigned to a different person i', this assignment is deleted, so person i' will become unassigned for the next iteration. This auction process is summarized below.

   For each object j, define the set $I(j) = \{i \in I \mid j(i) = j\}$ to be the set of bidders currently bidding for object j. If $I(j) = \Phi$ (the empty set), leave p_j unchanged and x_{ij} unchanged, i = 1, ..., n. If $I(j) \neq \Phi$, update the price of object j as

   $$p_j = \max_{i \in I(j)} b(i)$$

   If object j was previously assigned to person i' (i.e. $x_{i'j} = 1$), remove that assignment (i.e. set $x_{i'j} = 0$). Assign object j to one of the persons offering the highest bid for object j; that is,

   $$i^*(j) = \arg\max_{i \in I(j)} b(i)$$

   Set $x_{i^*(j)j} = 1$

The above bid and auction steps are repeated until each person is assigned to an object. As discussed in Appendix A, proper choice of the constant ϵ is required for this procedure to converge to an optimal assignment. In particular, if the benefits a_{ij} are all integer, the constant ϵ must be chosen to be smaller than 1/n, where n is the total number of persons. For integer benefits a_{ij}, by scaling all of the benefits by multiplication by (n+1), the AUCTION algorithm can be conducted using only integer arithmetic. This was the approach used in our implementations.

An important issue which affects algorithm performance on different multiprocessor architectures is the selection of data structures for the implementation of the AUCTION algorithm. Specifically, there are many WTA problems where certain interceptor-target assignments are known to be infeasible and should not be represented as part of the problem. In the assignment problem, this is represented by a set A(i) of admissible objects for the assignment of person i. Thus, the assignment $x_{ij} = 0$ unless $j \in A(i)$. The sets A(i) can be represented explicitly using sparse data structures, or they can be represented implicitly by selecting the benefit $a_{ij} = -\infty$ for $j \notin A(i)$ and using dense data structures. For sequential computation, sparse data structures provide a considerable advantage over dense data structures; for parallel computation, use of sparse data structures may require interprocessor movement of data which can reduce efficiency.

Note that any nonempty subset I of unassigned persons may submit a bid at each iteration. This gives rise to a variety of possible implementations, named after their analogs in relaxation and coordinate descent methods for solving systems of equations or unconstrained optimization problems (see e.g.[27,28]):

a. The *Jacobi* implementation, where I is the set of all unassigned persons at the beginning of the iteration.

b. The *Gauss-Seidel* implementation, where I consists of a single person, who is unassigned at the beginning of the iteration.

c. The *block Gauss-Seidel* implementation, where I is a subset of the set of all unassigned persons at the beginning of the iteration. (The method for choosing the

persons in the subset I may vary from one iteration to the next, so this implementation contains the preceding two as special cases.)

Generally, in a serial computation environment, the Gauss-Seidel implementation tends to be the fastest, but with a parallel machine, the choice is unclear because all the bids of the persons in I may be calculated in parallel. It is important to consider all these different versions because they provide starting points for different synchronous and asynchronous parallel implementations.

Figure 2-3 illustrates the Gauss-Seidel variation of the AUCTION algorithm. In this variation, an unassigned bidder is selected from a queue; this bidder selects the most desirable object (based on the object's perceived value and its price) and selects a bid price for this object which outbids every other bidder by as much as possible. Thus, if in a previous iteration another bidder had successfully bid for this object, this bidder is now rejected and joins the bidders' queue for future iterations. The auction proceeds until the bidders' queue is empty.

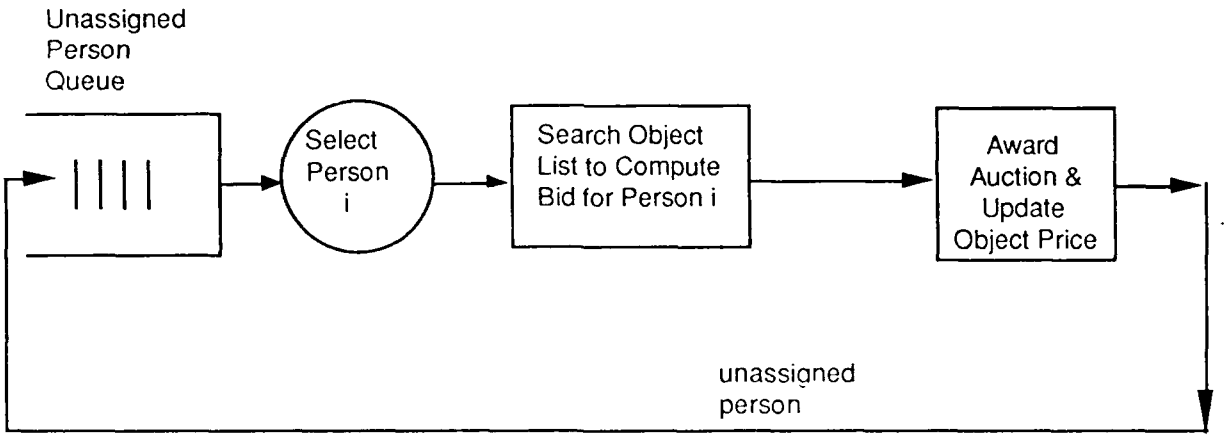

Figure 2-3. Structure of the Gauss-Seidel variation of the AUCTION Algorithm

The Jacobi variation of the AUCTION algorithm is similar, but assumes that all of the bidders on the bidding queue bid simultaneously; thus, an object may be bid on by more than one bidder at a time. In contrast with the Gauss-Seidel algorithm, a bidder is no longer assured of winning his bid, since other bidders may bid on the same object at the same time. Similarly, after all of the bidders have completed their bid, the objects are awarded to the bidder with the

highest offered price, and a new round of bidding is initiated. Figure 2-4 illustrates the structure of the Jacobi variation; in this figure, the number of parallel bidders p is equal to the number of unassigned persons in the unassigned person queue. Figure 2-4 also represents the block Gauss-Seidel variation when the number of parallel bidders is selected to be a number p which is smaller than the number of unassigned persons in the queue.

Sequential implementations of the Gauss-Seidel and Jacobi variations have shown that the Gauss-Seidel variation is 15-20% faster. In both variations, the key computation-intensive step is the computation of new bids for each bidder. In the Gauss-Seidel variation, this step encompasses over 95% of the total computation time of the AUCTION algorithm. In the Jacobi variation, the awarding of new auctions is harder, so the computation of new bids comprises only 85% of the total computation time.

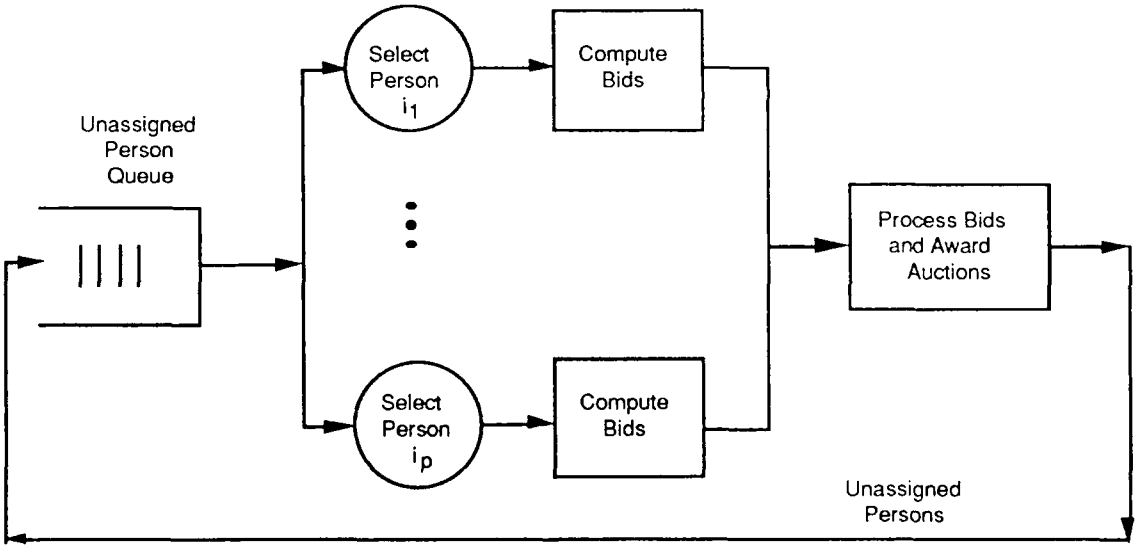

Figure 2-4. Structure of the Jacobi variation of the AUCTION algorithm. In this figure, p is the number of unassigned persons present in the unassigned persons queue.

2.4 VARIATIONS OF AUCTION FOR WTA PROBLEMS

The original AUCTION algorithm was designed to solve assignment problems, which correspond to WTA problems when the interceptor platform inventories are all uniformly equal

to one. For WTA problems, weapon platforms often carry more than one interceptor, so the platform inventories are larger than 1, and there are fewer weapons W than targets T. This asymmetry can be exploited to yield more efficient algorithms; the theory of these algorithms is described in Bertsekas and Castañon [18]. It also creates variations of the AUCTION algorithm with different structure, depending on whether we assign the targets to be the persons, or whether we assign the weapons to be the persons. The choice of variation for parallel processing depends on the level of parallelism which one is interested in exploiting.

For medium-grained parallelism using shared-memory MIMD processors, the structure of Fig. 2-4 appears more amenable for parallel processing than the structure of Fig. 2-3. In this structure, bid tasks for separate persons can be executed in parallel; similarly, auction tasks for separate objects can be executed in parallel. However, an important limit in the amount of parallelism which can be obtained from this approach is the average length of the unassigned persons queue. This limits the number of parallel bid tasks, which in turn limits the number of parallel auction tasks. Figure 2-5 shows a typical histogram of the queue length for a sequential implementation of the Jacobi AUCTION algorithm as a function of the number of

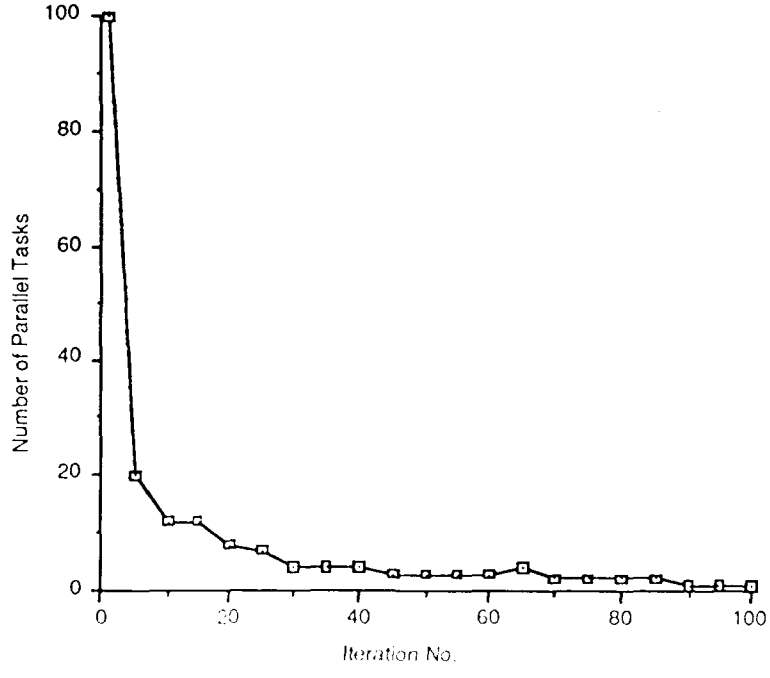

Figure 2-5. Length of Unassigned Persons Queue versus iteration number for Jacobi AUCTION.

iterations (the test problem involved 100 persons). As Fig. 2-5 indicates, the average speedup obtainable by this approach is limited to near 3-4, because of the dynamic load imbalance across iterations.

For medium-grained parallelism using multiple bid tasks, the variation of the AUCTION algorithm which should be most successful is one which maximizes the length of the unassigned persons queue across iterations. This is accomplished by selecting the targets as bidders, since there are more targets than weapons, leading to longer average queue lengths. Either the block Gauss-Seidel or the Jacobi variation of the AUCTION algorithms would then be used, depending on the available number of processors and the overhead required for interprocessor synchronization. When synchronization overhead is high, an asynchronous implementation of the AUCTION algorithm may be preferred; the theory of such an asynchronous implementation is described in Appendix A.

For fine-grained parallelism using SIMD architectures, the structure of Fig. 2-3 is superior to the structure of Fig. 2-4. In the Gauss-Seidel variation, most of the time is spent in the computation of individual bids. This operation is similar to finding a maximum value and maximum element of a list of objects. The amount of parallel work increases with the length of the object lists. Thus, the preferred variation of the AUCTION algorithm for exploiting fine-grained parallelism is to use a Gauss-Seidel variation, with weapons as persons. In this manner, the number of objects (corresponding to targets) is increased, thereby increasing the size of the bid tasks for fine-grained parallelism.

In the subsequent sections, we describe the design of the various parallel AUCTION algorithm variations developed under this contract.

SECTION 3

SYNCHRONOUS PARALLEL AUCTION ALGORITHMS

3.1 INTRODUCTION

In this Section, we overview the designs of the various synchronous parallel AUCTION algorithm implementations, and discuss the benchmarking results obtained. We first discuss the parallel AUCTION algorithms designed for MIMD architectures (the Encore Multimax and the Alliant FX/8). Several parallel AUCTION algorithms were developed and benchmarked; these algorithms differ in the degree to which fine-grained and coarse-grained parallelism is used. In later subsections, we discuss the parallel AUCTION algorithms designed for SIMD architectures (DAP 510 and CM-2); these algorithms were based on exploiting fine-grained parallelism, and are similar in design across the different multiprocessors. The benchmarking results illustrate the differences in performance which can be achieved on different multiprocessor architectures.

3.2 SYNCHRONOUS AUCTION ALGORITHMS ON THE ENCORE MULTIMAX

In synchronous shared memory implementations of the AUCTION algorithm, all bidding and assignment phases are separated by a synchronization point. There are two basic ways to parallelize the bidding phase for the set of unassigned persons I and a combination of the two:

a. *Parallelization across bids (or Jacobi parallelization)*: Here the calculations involved in the bid of each person $i \in I$ are carried out by a single processor. If the number of persons in I, call it |I|, exceeds the number of processors p, some processors will execute the calculations involved in more than one bid. (This will typically happen in the early stages of a Jacobi-type algorithm where I is the set of all unassigned persons.) If $|I| < p$, then $p - |I|$ processors will be idle during the bidding phase, thereby reducing efficiency. This will typically happen in the late stages of a Jacobi-type algorithm.

b. *Parallelization within a bid (or Gauss-Seidel parallelization)*: Here the set I consists of a single person as in the Gauss-Seidel implementation. The calculations involved in the bid of each unassigned person i are shared by the p processors of the system. Thus the set of admissible objects A(i) is divided in p groups of objects $A_1(i)$, $A_2(i)$, ..., $A_p(i)$. The best object, best value, and second best value are calculated within each group in parallel by a separate processor. We call the calculations within a group a *search task*. After all the search tasks are completed (a synchronization of the processors is required to check this) the results are "merged" by one of the processors who finds the best value over all best group values, while simultaneously computing the corresponding best object and size of bid. (It is possible to do the merging in parallel using several processors, but this is inefficient when the number of processors is small, as it was in our case, because of the extra synchronization and other overhead involved.) The drawback of this method over the preceding one is that it typically requires a larger number of iterations, since each iteration involves a single person. Even though each Gauss-Seidel iteration may take less time because it is executed by multiple processors in parallel, the synchronization overhead is roughly proportional to the number of iterations.

c. *Hybrid approach (or block Gauss-Seidel parallelization)*: In this approach, the bid calculations of each person are parallelized as in the preceding method,but the number of processors used per bid is k, where $1<k<p$. We will assume that k divides evenly p, so we can compute the bids of p/k persons in parallel, assuming enough unassigned persons are available for the iteration ($|I| \geq p/k$). With proper choice of k, this method combines the best features and alleviates the drawbacks of the preceding two.

Once the bidding phase of an iteration is completed (a synchronization point),the assignment phase is executed. This phase is typically carried out by a single processor in our synchronous implementations. While it is possible to consider using multiple processors to execute the assignment phase in parallel, the potential gain from parallelization is modest while the associated overhead more than offsets this gain in our system.

In the subsequent subsections, we describe the designs and benchmark results obtained from different parallel AUCTION algorithm designs for the Encore Multimax based on Gauss-

Gauss-Seidel parallelism, Jacobi parallelism and Block Gauss-Seidel parallelism. All algorithms were coded in Fortran 77 using the same sparse data structures.

3.2.1 Gauss-Seidel AUCTION Algorithm

The synchronous Gauss-Seidel AUCTION algorithm generates a single bid at a time, and uses multiple processors to search the possible objects n order to generate that bid. The premise of the parallel Gauss-Seidel AUCTION algorithm is to use multiple processors to reduce the computation time associated with computing each bid. The flexibility of a shared-memory MIMD architecture allows for the efficient use of sparse data structures. The parallel algorithm design includes synchronization in order to guarantee that the bids generated by the parallel algorithm are independent of the number of processors used, and thus represent a faithful replication of the sequential Gauss-Seidel AUCTION algorithm.

Figure 3-1 illustrates the percentage of the total computation time which is spent in searching the list of admissible objects A(i) for several 1000 person assignment problems with varying degrees of sparsity. As Fig. 3-1 illustrates, the sequential Gauss-Seidel AUCTION algorithm spends between 92 - 99 % of its computation time (depending on the problem size and the density of feasible assignments) searching the list A(i). This percentage increases with the average number of elements in the admissible assignments A(i), so that greater speedups are possible for larger problems.

The design of the synchronous Gauss-Seidel AUCTION algorithm is illustrated in Fig. 3-2. The majority of the AUCTION algorithm is conducted on a single processor (called the parent processor). Multiple processors are used to assist the parent processor in computing each bid in parallel using a "divide and conquer" strategy: each processor is assigned to search a fixed part of the list of objects A(i) which can be assigned to person i. Two synchronization points are included in each bidding iteration. The first synchronization point is a barrier (based on the barrier monitor developed at ANL/MCS [29]) which serves to delay the start of the search of admissible objects until the previous price update is completed. The second

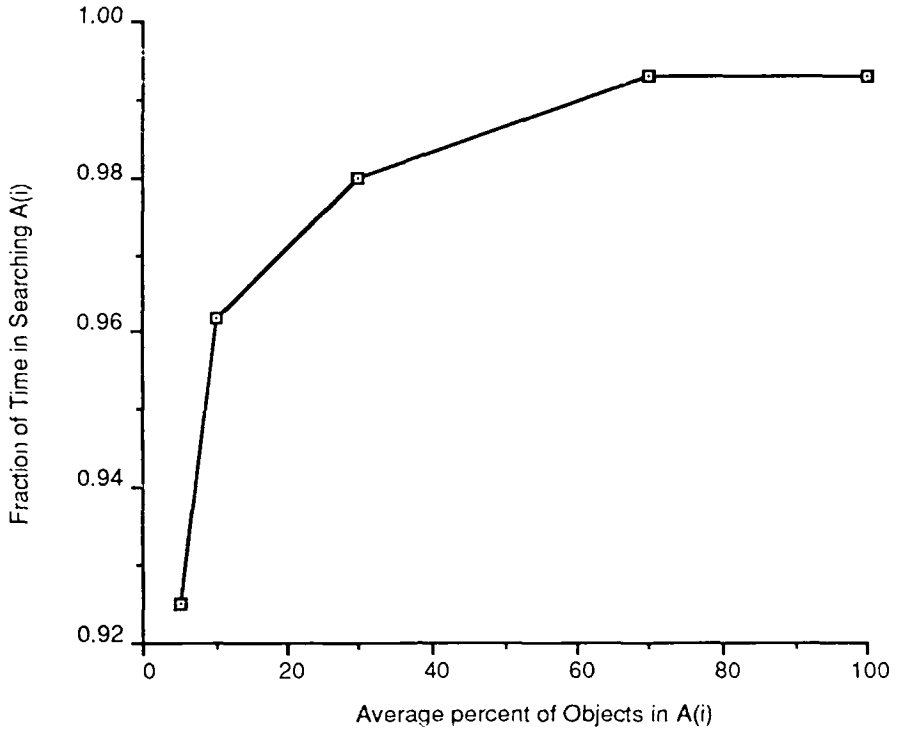

Figure 3-1. Percentage of total Gauss-Seidel AUCTION computation time spent in searching the lists of admissible objects for 1000 person assignment problems, benefit range 1-1000, as a function of the density of feasible assignments.

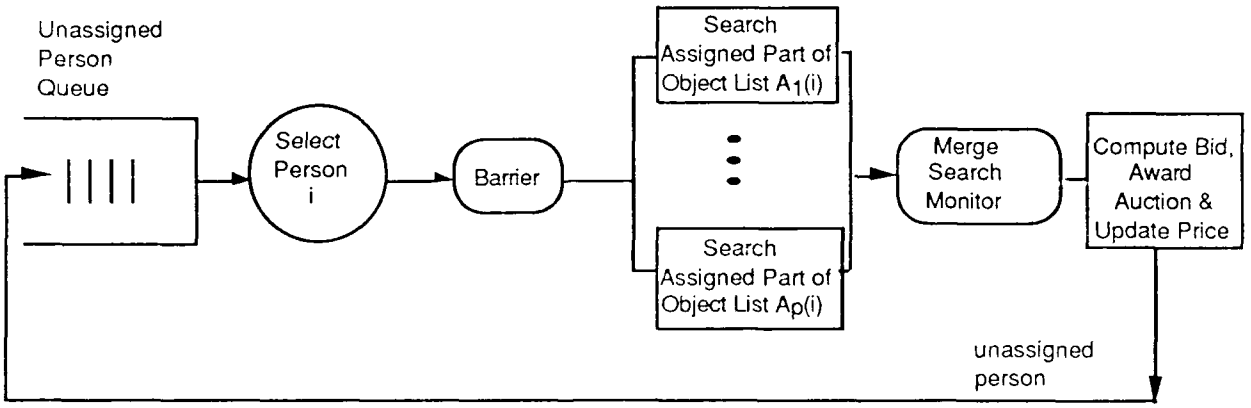

Figure 3-2. Design of the parallel synchronous Gauss-Seidel AUCTION algorithm. Multiple processors are used to search the list of admissible objects for a person; the results of the searches are merged to compute a person's bid, and the rest of the bid and auction cycles are conducted by a single processor.

synchronization point is a monitor which is an extension of the Argonne monitors for portable parallel programming [29]. The merge search monitor allows each processor, upon completion of its search of $A_k(i)$, to merge the results of its search (the highest and second highest net profit levels in the sublist, and the object which provided the highest net profit level) with the results of other processors which have completed their search, and then proceed to a barrier to wait for all of the processors to complete their search. The monitor sequences the merging of the processor searches to guarantee that the results of the merged search are identical with the one-processor Gauss-Seidel algorithm.

In order to understand the potential performance of the parallel Gauss-Seidel AUCTION algorithm, we have constructed an empirical model for the computation time per iteration with p processors per bid. This time is given by

$$T(p)=S(p)+M(p)+C(p)+V$$

where

S(p)= Time for completing the search tasks

M(p)= Time for merging the results of search tasks

C(p)= Time for synchronization

V= Constant overhead per iteration.

Let us assume for convenience that each set of admissible objects A(i) has the same number of elements, say n. By counting the number of operations and by assuming perfect load balancing between the search tasks (i.e., an equal number of objects n/p in each of the groups $A_1(i)$, ..., $A_p(i)$, we have estimated roughly that the search time per iteration is

$$S(p) = \text{Constant} \bullet (n/p + \log(n/p) + \log(\log(n/p)) \qquad (3\text{-}1)$$

(The logarithmic terms account for the calculations involving the second best value.) The merging time is proportional to p, while the synchronization time using software barriers is roughly proportional to p.

It can be seen that, given n, there is an optimal value of p that minimize the total time per iteration. For example, if p is large, the increase of the synchronization and merging times may offset the potential gains from parallelization of the search tasks. Because of various constants involved in the preceding estimates of the search, merging, and synchronization times, it is difficult to estimate a priori the optimal value of p required to solve the problem.

Figure 3-3 illustrates the performance of the synchronous Gauss-Seidel AUCTION algorithm for a 1000 person, 20% dense assignment problem with benefits in the range [1,1000]. All of the times reported in the figure are measured in terms of the parent processor (the processor which executes the sequential part of the algorithm). The scan time is the time which the parent processor (processor 1) spends in searching its part of the admissible object lists $A_1(i)$. The predicted relationship between scan time and the number of processors is derived from Eq. 3-1. The synchronization time for a bid by person i is measured as the time from which the parent processor finished scanning the subset of objects $A_1(i)$ until the time the parent processor is released from the merge search monitor to continue with the auction process. As the results of Fig. 3-3 indicate, the achievable speedup for this problem is limited to a factor of nearly 3 because of the increase in synchronization time required to merge the results of the various searches. This factor will increase as the number of elements in the sets A(i) increases.

Figure 3-4 illustrates the conjectured theoretical behavior of the total scan, synchronization and computation times, based on fitting the models described in the previous section with appropriate constants to match the problem size. Note the close correspondence between the predictions of Fig. 3-4 and the empirical results of Fig. 3-3. The only minor discrepancy is that the empirical synchronization time grows superlinearly with the number of processors; this is probably due to increased contention for access to critical sections in the barrier and merge search monitors. Similar phenomena were observed by Dritz and Boyle [30] in their experiments using the Encore Multimax.

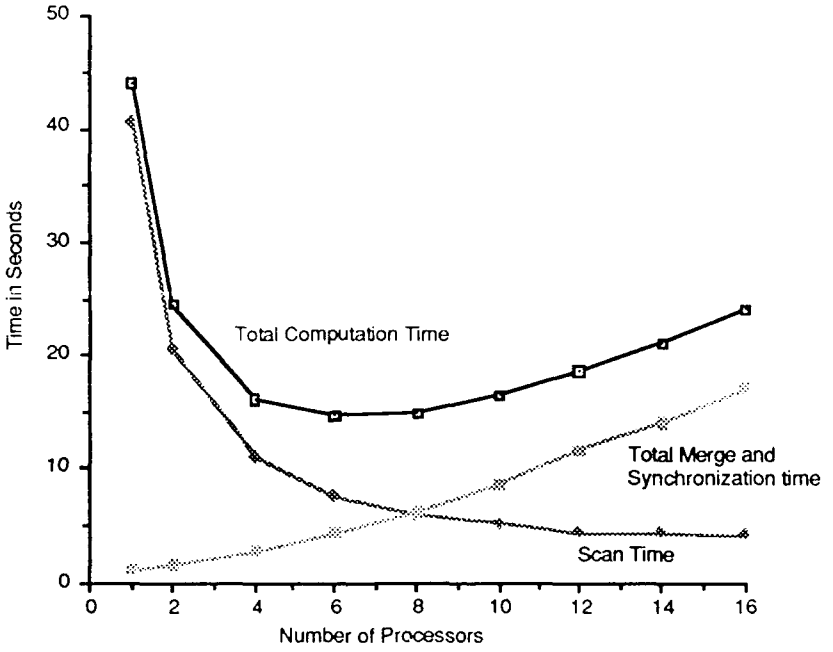

Figure 3-3. Performance of the Synchronous Gauss Seidel AUCTION algorithm on the Encore Multimax as the number of processors increases for a 1000 person, 20% dense assignment problem with benefit range [1,1000]. Note the growth in merge and synchronization time required as the number of processors increases. This limits the maximum speedup to a factor of approximately 3.

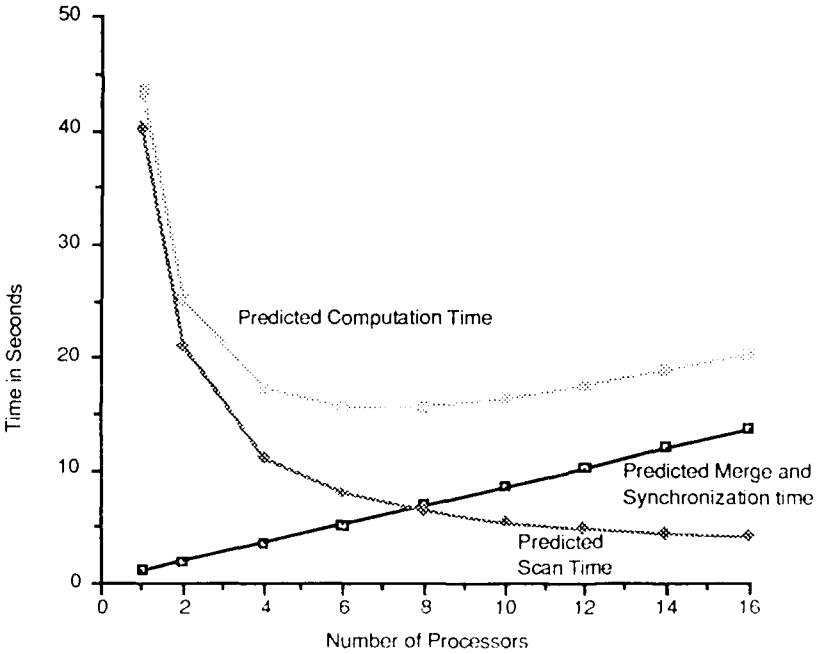

Figure 3-4. Theoretical behavior of synchronous Gauss-Seidel AUCTION algorithm with increasing number of processors. The constants have been matched to fit the times of a 1000 person, 20% dense assignment problem with benefit range [1,1000].

Figure 3-5 illustrates the effective speedup achieved by the parallel Gauss-Seidel AUCTION algorithm as a function of the density of feasible assignments for an 800 person assignment problem. As the density decreases, the potential for parallel work decreases also. However, for denser problems, speedups approaching factors of 6 are possible using up to 10 processors. Figure 3-6 illustrates similar results for larger, 1000 person assignment problems. Note that the sequential computation time for this larger problem has nearly doubled. This increase in computation time is due to the an increase in the number of feasible assignments which must be considered in the problem (which has also nearly doubled, from 640,000 to 1,000,000 for fully dense problems); the empirical computation time grows near-linearly with the number of feasible assignments to be considered. For the larger 1000 person assignment problem, a speedup of nearly 6.7 was achieved for the fully dense problem.

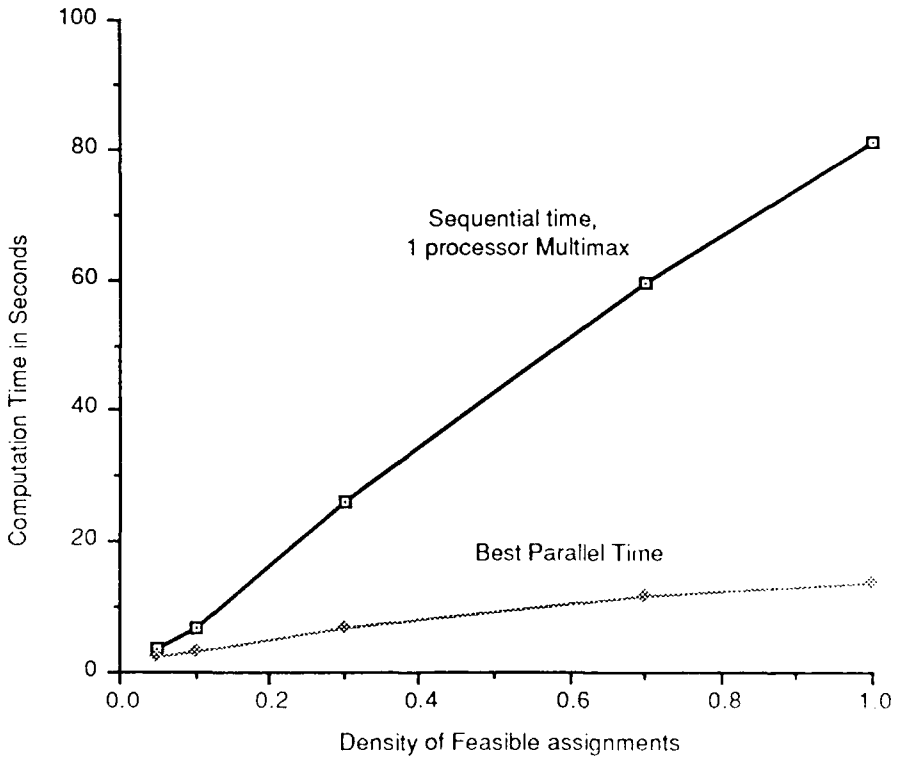

Figure 3-5. Comparison of best parallel and sequential times for Gauss-Seidel AUCTION algorithm on Encore Multimax for 800 person assignment problem, benefit range [1,1000] as a function of the density of feasible assignments. The maximum number of processors used for the parallel Gauss-Seidel AUCTION algorithm was 10 processors for the fully-dense problem.

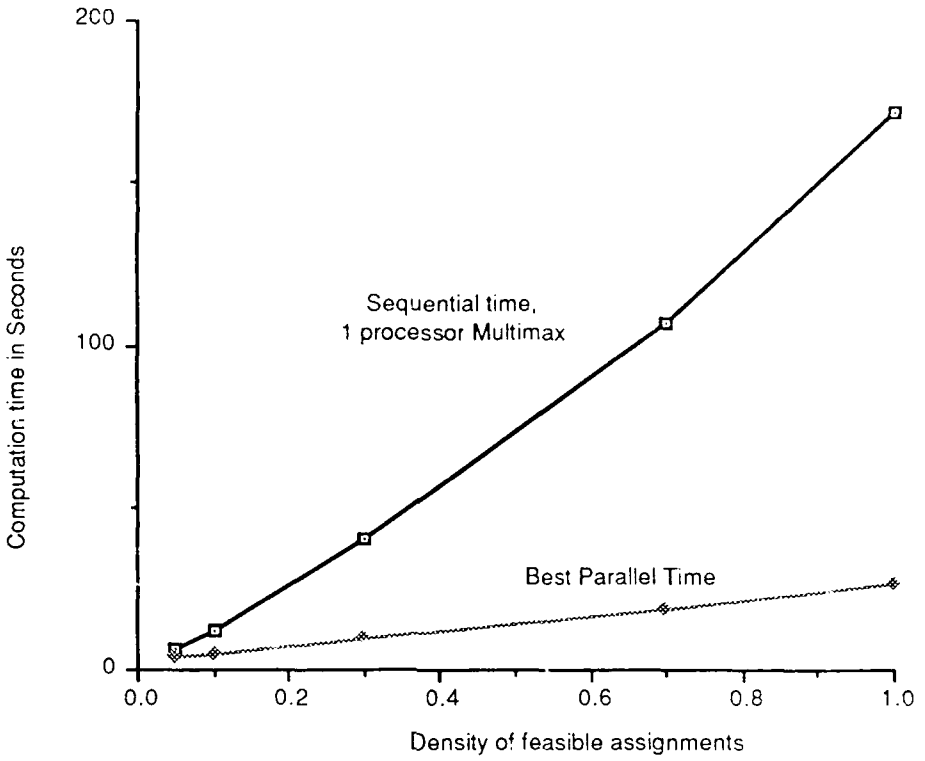

Figure 3-6. Comparison of best parallel and sequential times for Gauss-Seidel AUCTION algorithm on Encore Multimax for 1000 person assignment problem, benefit range [1,1000] as a function of the density of feasible assignments. The maximum number of processors used for the parallel Gauss-Seidel AUCTION algorithm was 10 processors for the fully-dense problem.

As Figs. 3-5 and 3-6 indicate, the potential speedup on MIMD architectures for the Gauss-Seidel AUCTION algorithm depends critically on the density of the feasible assignments (the speedup depends on the average number of feasible assignments for each person, which is the product of the density times the total number of objects). For many WTA problems, we expect the density of feasible assignments to be in the 10-70% range. This limits the overall speedup for 1000 interceptor assignment problems to factors between 2.5 and 5.5. These factors will increase as the numbers of interceptors and targets increase, since the overall spatial volume of interest remains constant (thereby preserving the overall density of the feasible assignments); in essence, the synchronization overhead in Fig. 3-3 will remain constant (depending only on the number of processors used), while the parallelizable work for

the searches will increase proportionately to the number of objects. For problems with 10,000 objects, the overall speedup should mirror the speedups in the scan time, suggesting that speedups of over 10 will be possible using 16 processors.

3.2.2 Jacobi AUCTION Algorithm

The second synchronous implementation of the AUCTION algorithm was the synchronous Jacobi AUCTION algorithm. In this algorithm, multiple processors are used to generate bids simultaneously for different persons. The number of simultaneous bids generated is equal to the minimum of the number of processors used and the number of unassigned persons; in this manner, object prices are updated as soon as possible, leading to an expected reduction in the overall number of bids required to converge to an optimal solution. Each processor computes the bid associated with a different person. The resulting bids are then processed sequentially in order to award new auctions and to update the list of unassigned persons. Sequential processing of the bids guarantees that the number of iterations required for convergence of the Jacobi AUCTION algorithm is independent of the order in which processors finish their computations.

The design of the synchronous Jacobi AUCTION algorithm is illustrated in Fig. 3-7. Again, there are two synchronization points for each iteration of the algorithm, before and after the compute bids operation. However, both synchronization points are implemented with the extensions of the barrier monitors discussed previously. In particular, the synchronization after the compute bids operation is only a barrier monitor because no merging of the individual computations by each processor is required (unlike the synchronous Gauss-Seidel AUCTION algorithm). This reduces the overall synchronization overhead by reducing the length of the critical section in the synchronization monitor. After the bids have been computed, the parent processor conducts the auction for each bid and places unassigned persons back into the queue.

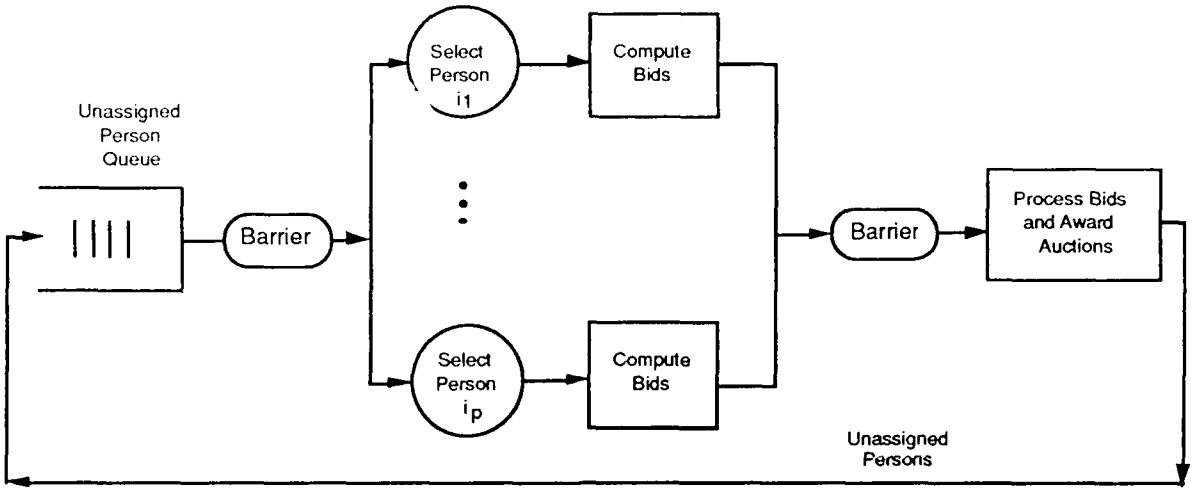

Figure 3-7. Design of synchronous Jacobi AUCTION algorithm. Multiple processors are used to compute bids for multiple persons simultaneously. The parent processor then processes sequentially the bids

An important aspect of the synchronous Jacobi AUCTION algorithm is that the amount of potential parallel work varies across iterations; specifically, it depends on the number of remaining unassigned bidders. When the number of unassigned bidders is less than the number of available processors, some of the processors will be idle. Figure 3-8 illustrates the number of unassigned bidders per bid iteration for a 1000 person, 20% dense assignment problem, benefit range [1,1000] using 10 processors. In order to prevent idle processors for competing for shared resources such as synchronization locks, the size of the synchronization barriers was adaptively modified to match the number of non-idle processors. Idle processors were diverted to a rest barrier, waiting to rejoin the computation when the number of unassigned persons grew larger than the number of available processors (at the beginning of a new ε - scaling phase; see Appendix A).

Figure 3-9 illustrates the performance of the Synchronous Jacobi AUCTION algorithm. Again, scan time is measured in terms of the time required for the parent processor to compute a bid; scan time is decreased with the number of processors because the parent processor computes less bids (other processors compute bids simultaneously). Synchronization time is measured in terms of the time spent by the parent processor at the two synchronization

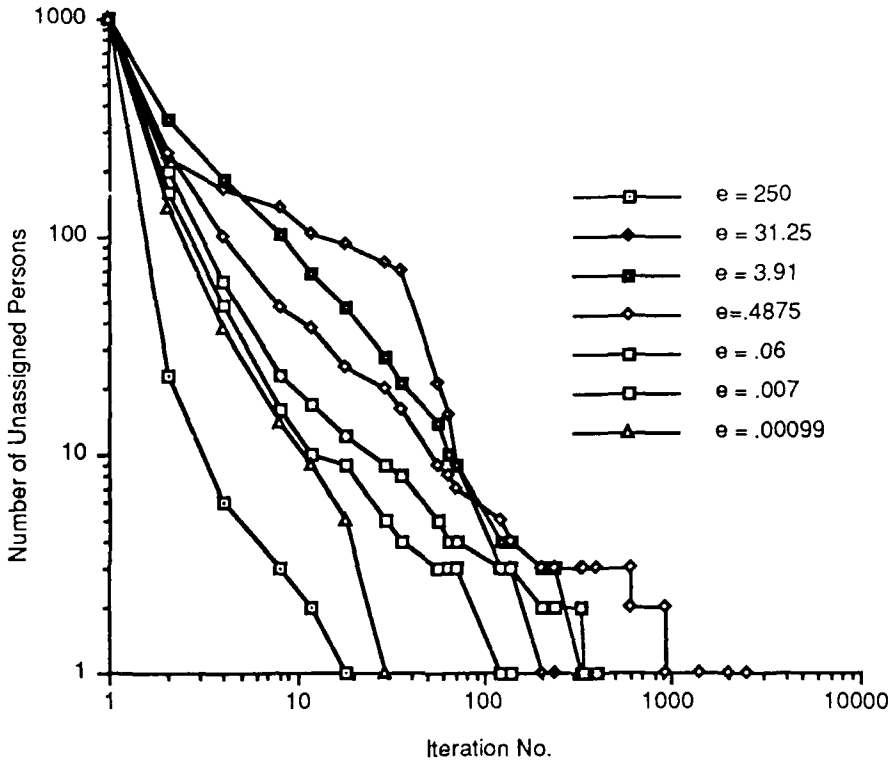

Figure 3-8. Jacobi AUCTION number of unassigned persons versus iteration number for 1000 person, 20% dense assignment problem, benefit range [1,1000] using 10 processors. Curves illustrate the number of unassigned persons for different values of ε corresponding to different ε-scaling cycles. Note the small fraction of iterations for which the number of unassigned persons exceeds the number of available processors (10).

barriers. Note that, unlike the synchronous Gauss-Seidel algorithm, scan time cannot be reduced arbitrarily by increasing the number of processors. In the Jacobi AUCTION algorithm, increasing the number of processors generally reduces the overall number of iterations required to converge (by computing multiple bids in parallel); however, for iterations where the number of unassigned persons is less than the number of processors, increasing the number of processors has no effect on the number of parallel bids computed, thereby limiting the reduction possible in scan time.

Note the relatively low level of synchronization required for the Jacobi AUCTION algorithm when compared to the Gauss-Seidel AUCTION algorithm. This is due to three

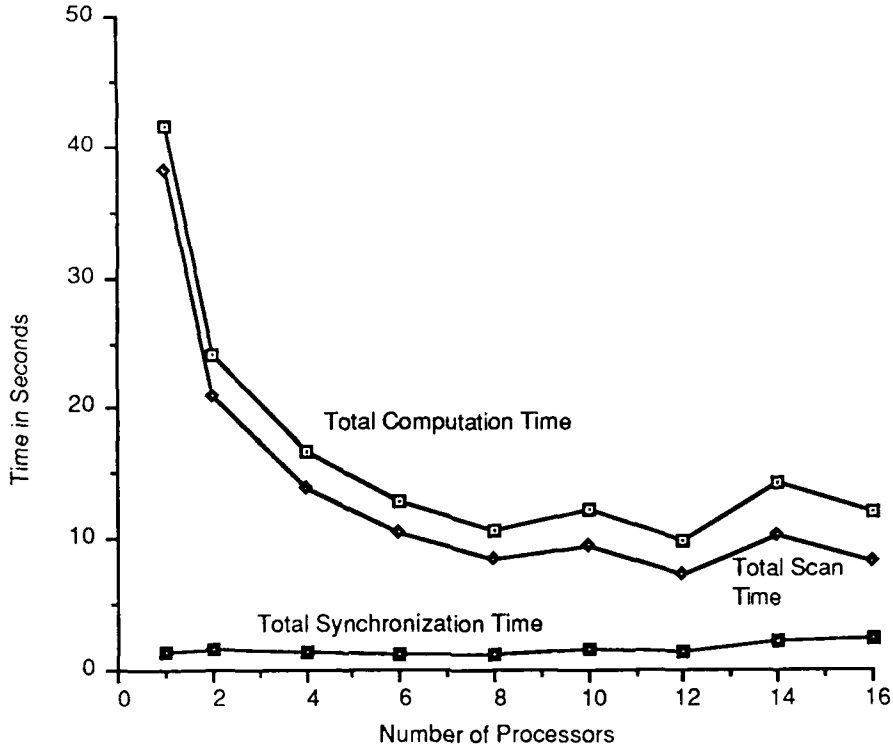

Figure 3-9. Performance of the synchronous Jacobi AUCTION algorithm for 1000 person, 20% dense assignment problem, benefit range [1,1000] as a function of number of processors.

factors. First, the synchronization after computing bids is simpler because no merging of the results of the processors is required. Second, the number of synchronization calls is reduced because the total number of iterations is reduced by processing multiple bids in parallel. Finally, the number of processors which contend for a synchronization lock is reduced adaptively when the numbe- of unassigned persons is less than the number of processors, leading to simpler synchronization (with reduced contention) at each iteration.

The results in Fig. 3-9 indicate an interesting anomaly which is typical of the AUCTION algorithm: increasing the number of processors sometimes produces an apparent increase in computation time, as indicated in the difference between the 10 processor times and the 8 processor times. The reason for this increase is that the number of iterations required for convergence with 10 processors increased significantly over the number of iterations required for convergence with 8 processors. This is because the computation of bids with 10

processors is based on a potentially different set of object prices than the computation of bids with 8 processors. This fluctuation in the number of iterations required for convergence will become a dominant factor in the performance of the asynchronous AUCTION algorithms discussed in Section 4.

Figure 3-10 illustrates the speedups achieved by the Jacobi AUCTION algorithm for several 800 person and 1000 person assignment problems. The curves indicate that the effective speedup from Jacobi parallelization in the 10% -70% density range is not likely to vary much with either the number of persons or the density of the problem (although the potential speedup will decrease for very sparse assignment problems). This is in contrast with Gauss-Seidel parallelism, where the speedups possible increased with problem density and with assignment problem size.

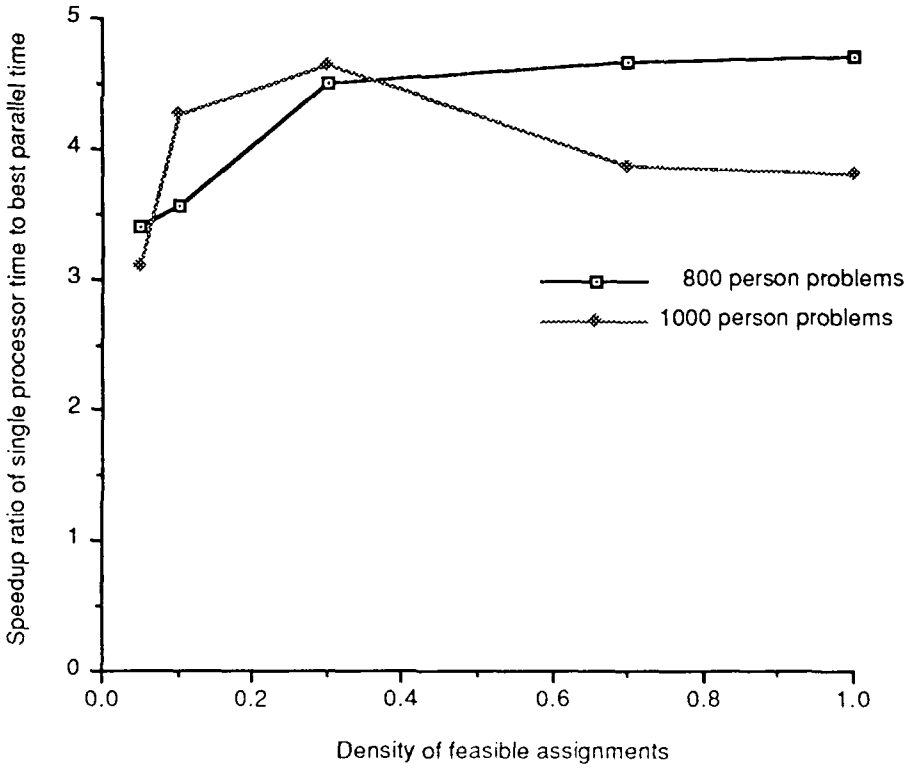

Figure 3-10. Speedup of parallel Jacobi AUCTION algorithm over the single-processor algorithm as a function of the density of feasible assignment problems for problems with 800 and 1000 persons, benefit range [1,1000].

3.2.3 Hybrid AUCTION Algorithm

The results obtained with tne previous two synchronous algorithms suggest that an efficient parallel implementation should combine the speedups available from Gauss-Seidel parallelization and Jacobi parallelization. In particular, by computing multiple bids simultaneously, and by using multiple processors to compute each bid, a multiplicative effect may be achievable where the overall speedup is the product of the Gauss-Seidel speedup and the Jacobi speedup. The synchronous Hybrid AUCTION algorithm is an attempt to demonstrate this multiplicative speedup; in this algorithm, persons are selected two at a time, and two bids are computed in parallel (Jacobi parallelization with two processors). For each person i, the admissible objects A(i) are searched in parallel by multiple processors (Gauss-Seidel parallelization).

The overall design of the synchronous hybrid AUCTION algorithm is illustrated in Fig. 3-11. There are three synchronization points per iteration. An initial barrier is included to delay the start of the object searches until all of the object prices are updated from the previous iteration. A separate merge search monitor is included for each person, and a synchronization barrier is used to wait until both bids are computed before proceeding to award the auctions. The size of the barriers and monitors were tailored to the number of processors which rendezvous at each synchronization point. Thus, the first barrier synchronized 2k processors, the merge search monitors k processors and the last barrier only two processors, thereby keeping the synchronization overhead to a minimum. The predicted speedup from the hybrid approach should be 1.75 for the use of Jacobi parallelization with two bids computed simultaneously, multiplied times the appropriate speedup (cf. Fig. 3-3) for using k processors to compute each bid. Thus, when 12 total processors are used, the overall speedup should be approximately 1.75 x 2.75 (from using 6 processors per bid) = 4.8125.

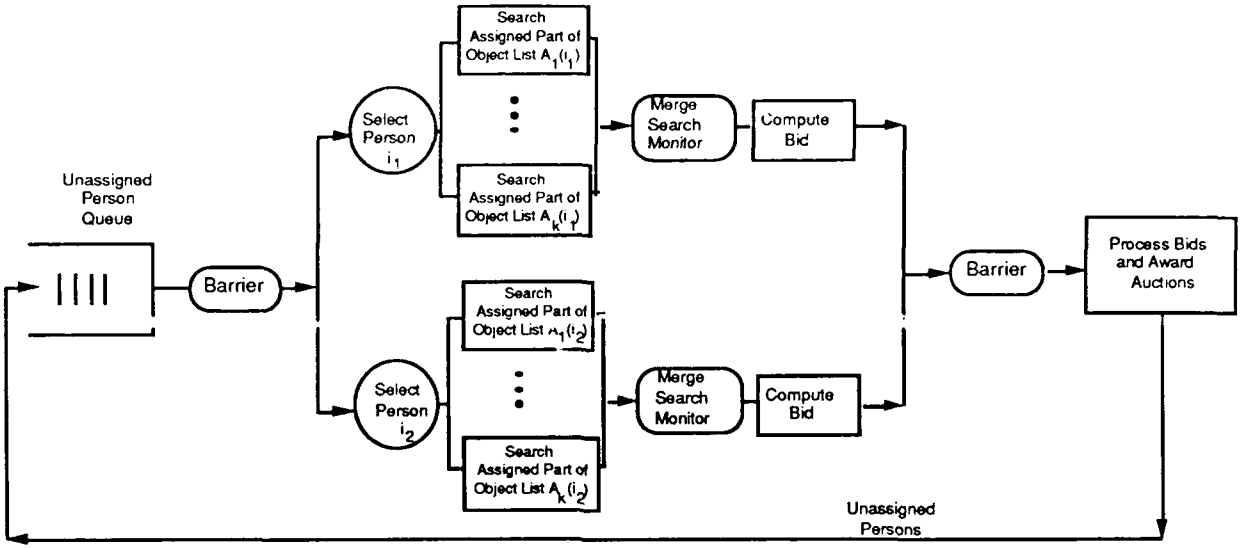

Figure 3-11. Design of the synchronous Hybrid AUCTION algorithm.

Figure 3-12 illustrates the performance of the synchronous Hybrid AUCTION algorithm as a function of the total number of processors used for the same 1000 person, 20% dense assignment problem described previously. The single processor time for this algorithm is 44 seconds. The synchronization time is again measured in terms of the parent processor, and represents the total time that the parent processor spends at the different synchronization points. As the curves in Fig. 3-12 indicate, the achieved speedup is much lower than the anticipated multiplicative speedup from combining the Jacobi and Gauss-Seidel speedups. For example, the actual speedup using 12 processors is under 4, whereas the predicted speedup is over 4.8. The explanation for this loss of effectiveness can be seen in the growth of the synchronization time with the total number of processors used, even though the total number of iterations has been reduced by a factor of 1.83. This synchronization time represents the dominant part of the overall computation time for large number of processors, and prevents effective combination of the speedups possible from Gauss-Seidel and Jacobi parallelization. This motivated the development of asynchronous Hybrid AUCTION algorithms with reduced synchronization overhead, using the theory developed in Appendix A. These algorithms will be discussed further in Section 4.

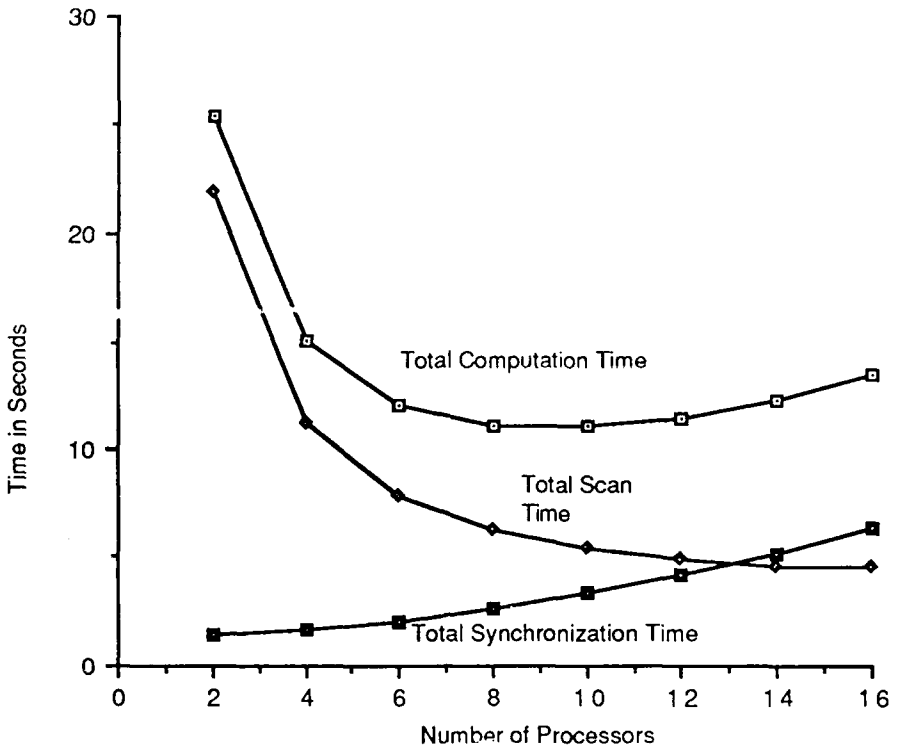

Figure 3-12. Performance of the synchronous Hybrid AUCTION algorithm on Encore Multimax as a function of the number of processors for 1000 person, 20% dense assignment problem, benefit range [1,1000].

Figure 3-13 illustrates the performance of the parallel Hybrid AUCTION algorithm and the parallel Gauss-Seidel AUCTION algorithm as a function of the number of processors. As expected, the Hybrid AUCTION algorithm can use a larger number of processors in a more effective manner, since the merge and synchronization time is significantly reduced by having a smaller number of overall iterations (from computing bids two at a time) and by merging the results of only half the number of processors. However, Fig. 3-13 also illustrates the absence of a multiplicative speedup; the ratio of the best Gauss-Seidel AUCTION time to the best Hybrid AUCTION time is about 1.35, which is smaller than the 1.75 factor.

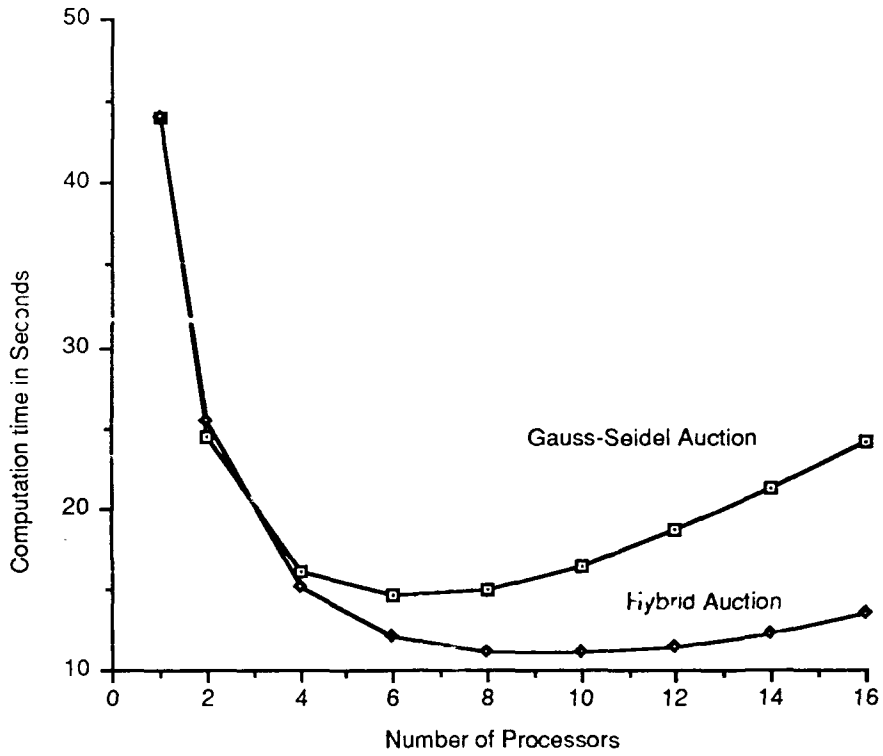

Figure 3-13. Comparison of Hybrid AUCTION and Gauss-Seidel AUCTION algorithms for similar numbers of processors for 1000 person, 20% dense assignment problem, benefit range [1,1000].

3.3 SYNCHRONOUS AUCTION ALGORITHMS ON THE ALLIANT FX/8

The parallel algorithms discussed in Section 3.2 were implemented on the Encore Multimax with no assistance from any automated parallelization tools. Parallel processing was implemented by generating parallel tasks, and having the operating system of the Encore Multimax schedule these tasks concurrently on multiple processors. Synchronization of these tasks was achieved by writing explicit software monitors, using a spinlock mechanism as the basic synchronization primitive provided by the Multimax.

A different approach for parallel algorithm development is to use a parallelizing compiler, which searches for work to do in parallel, and automatically distributes parallel work across processors. The Alliant FX/8 computer has a Fortran compiler with this capability for automatic parallelization; furthermore, the Alliant FX/8 had other interesting architectural features which made it an interesting candidate for investigation. These features are:

1. The automatic parallelizing Fortran compiler;

2. The Alliant architecture is designed to implement several synchronization primitives in hardware, thereby reducing the overhead required for interprocessor synchronization;

3. Each of the Alliant FX/8's processors is a vector processor, which is a particular type of SIMD architecture. Thus, the Alliant FX/8 is a hybrid architecture, capable of multiprocessor MIMD and SIMD processing;

4. The Alliant FX/8 has a high-level array language (Fortran 8X) which is similar to the array languages used on SIMD architectures such as the DAP 510 or the CM-2.

Thus, conducting experiments on the Alliant FX/8 provided a natural transition from MIMD architectures to SIMD architectures, and allowed us to evaluate the potential effectiveness of vector-processing and automatic parallel compilation for implementation of parallel AUCTION algorithms.

On the Alliant FX/8, we experimented only with the Gauss-Seidel AUCTION algorithm. Four different versions of the algorithm were developed:

1. Sequential Gauss-Seidel AUCTION, a Fortran 77 version using sparse data structures which corresponded to the most effective sequential implementation;

2. Parallel Gauss-Seidel AUCTION, a Fortran 77 version using sparse data structures, which was rewritten to avoid data dependencies which restricted the parallelization capable of the automated compiler.

3. Gauss-Seidel AUCTION 8X, a Fortran 8X version using dense data structures which was written to represent the AUCTION algorithm using array operations.

The sequential Gauss-Seidel AUCTION algorithm was identical to the sequential version used in the Encore Multimax, and required no further development. There is a key aspect to the sequential algorithm which must be understood in order to identify the transformations required for developing the parallel Gauss-Seidel AUCTION algorithm. As Fig. 3-1 illustrates, the key operation which consumes most of the computation time is the computation of a bid. Referring to the description of this operation in Section 2.3, the computations required for a bid from person i are:

$$j(i) = \arg\max_j \{a_{ij} - p_j\} \qquad (3\text{-}2)$$

$$v(i) = \max_j \{a_{ij} - p_j\} \tag{3-3}$$

$$w(i) = \max_{j \neq j(i)} \{a_{ij} - p_j\} \tag{3-4}$$

$$b(i) = p_{j(i)} + v(i) - w(i) + \varepsilon \tag{3-5}$$

The difficult computations are in Eqs. 3-2, 3-3, and 3-4. Each of these computations requires searching the list of admissible objects for person i, and is a reduction operation which maps a long vector of numbers into a single scalar. In the sequential implementation of the Gauss-Seidel AUCTION algorithm, all three quantities (j(i), v(i), w(i)) are computed in a single search of the object list. However, this computation introduces data dependencies which prevent the automatic parallelization of these operations on the Alliant FX/8.

In order to achieve maximum speedup and concurrency on the Alliant FX/8, the quantities j(i), w(i) and b(i) must be computed using three separate searches of the object list (a fourth array operation is also required, so the total computation is nearly four times longer). Thus, the parallel Gauss-Seidel AUCTION algorithm on the Alliant FX/8 is significantly slower when executed on a single sequential processor than the sequential Gauss-Seidel AUCTION algorithm. Similarly, the Gauss-Seidel AUCTION 8X algorithm requires three different array search operations to compute a bid for person i. We defer discussion of the implementation of the Gauss-Seidel AUCTION 8X algorithm until the next subsection, when we discuss array language implementations for the SIMD architectures.

Figure 3-14 illustrates the performance of the three algorithms for 800-person assignment problems with variable feasible assignment density. Note the logarithmic scale of the vertical axis. Three different compiled versions of the parallel Gauss-Seidel AUCTION algorithm were used: the version compiled to execute on one sequential processor (AUCTION 1S), the version compiled to execute on one vector processor (AUCTION 1V), and the version compiled to execute on all 8 vector processors (AUCTION VC). The other curves correspond

to the sequential Gauss-Seidel AUCTION algorithm (SAUCTION) and the Gauss-Seidel AUCTION 8X algorithm (AUCTION 8X).

Note the similar behavior in Fig. 3-14 of the AUCTION 1S, AUCTION 1V and SAUCTION algorithms as a function of feasible assignment density. In essence, the ratios of computation times between these algorithms is a constant factor, which reflects the additional number of searches of the object list required by the parallel Gauss-Seidel AUCTION algorithm! As predicted, the AUCTION 1S computation times are nearly four times slower than the SAUCTION computation times. Surprisingly, the use of vectorization is insufficient to fully compensate for this difference, so the AUCTION 1V times are about 10% slower than the SAUCTION times. When both vectorization and concurrency are used, the AUCTION VC times are faster than the SAUCTION times, but the speedup depends explicitly on the density of the feasible assignments. The maximum speedup (achieved for the fully dense problem) was nearly a factor of 4 (significantly smaller than the speedup on the Encore Multimax using only scalar processors). On the other hand, when referenced with respect to the AUCTION 1S times, the AUCTION VC times achieve a speedup of over 15 for dense

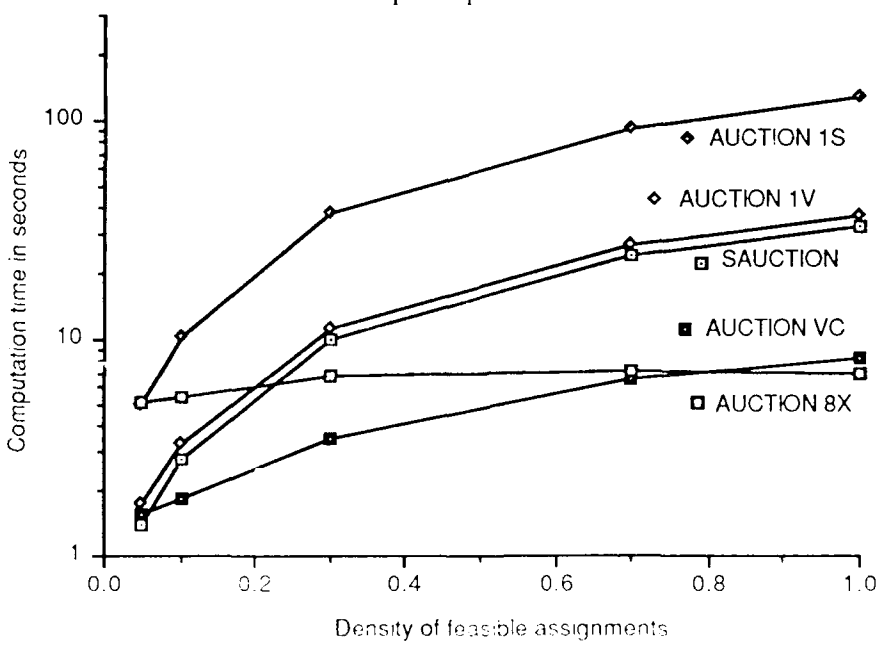

Figure 3-14. Performance of the different Gauss-Seidel AUCTION algorithms on the Alliant FX/8 for 800 person assignment problems with benefit range [1,1000], as a function of the density of feasible assignments.

assignment problems! This emphasizes the importance of using an efficient sequential implementation of the AUCTION algorithm as a scalar benchmark.

Figure 3-14 also illustrates the relative advantages of using sparse data structures versus dense data structures. Note the relatively flat AUCTION 8X computation times as a function of feasible assignment density, when compared with the curves of the other algorithms implemented using sparse data structures. Note that, for fully dense problems, a small efficiency is achieved by using dense data structures (roughly 15% of the overall computation time. However, once the problems become moderately sparse (below 80% dense), the sparse AUCTION VC implementation is significantly faster than the dense AUCTION 8X implementation.

Figure 3-15 illustrates the performance of the same 5 algorithms on a set of 1000 person assignment problems. Again, the AUCTION 1S times are nearly four times bigger than the SAUCTION times, and the use of vectorization in AUCTION 1V is insufficient to compensate for the loss of efficiency required by scanning the admissible object list an increased number of times. The larger problem size results in an increased maximum speedup of the AUCTION VC time (nearly 4.5 for fully dense problems) when compared with the SAUCTION times. Note the interesting anomaly present for the 5% dense problem. The vector-concurrent version of the parallel Gauss-Seidel AUCTION algorithm is slower than both the vector version of the same algorithm running on a single processor and the sequential Gauss-Seidel AUCTION algorithm. This reflects the compiler's inability to select dynamically how many parallel processors should be used in the computation. For this problem, the object lists averaged 50 objects; vectorization of the searches using length 32 vectors is more efficient than use of multiple processors, given the small number of objects to be searched.

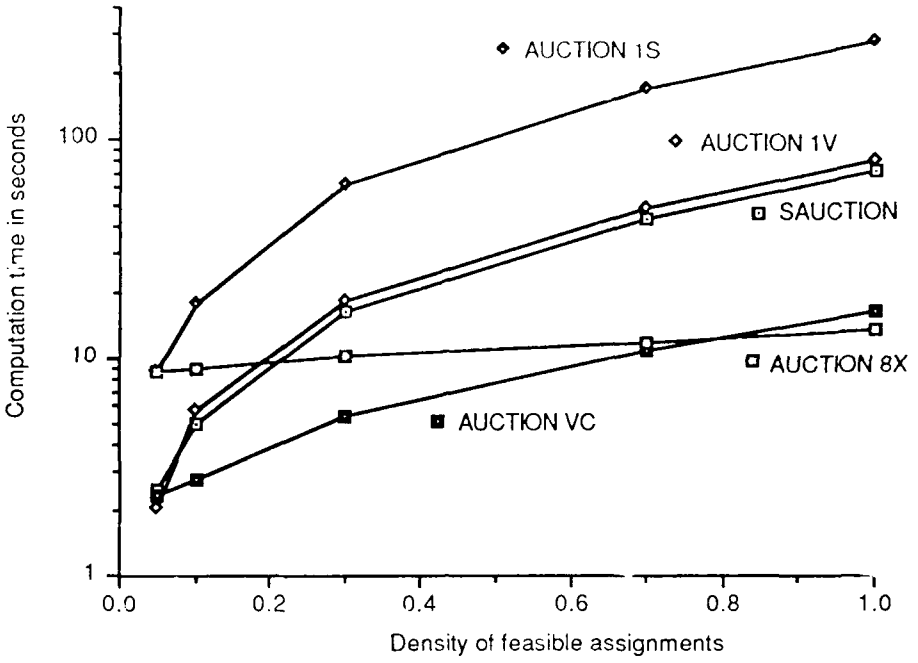

Figure 3-15. Performance of the different Gauss-Seidel AUCTION algorithms on the Alliant FX/8 for 1000 person assignment problems with benefit range [1,1000], as a function of the density of feasible assignments.

3.4 SIMD AUCTION ALGORITHMS

As discussed previously, the majority of the computation time of the sequential Gauss-Seidel AUCTION algorithm is spent in the scan operation, which consists of searching each object list A(i) in order to find the object offering the maximal net profit, and the two highest profit levels. The goal of our single instruction stream, multiple data stream (SIMD) implementations is to reduce the overall time associated these searches. An important aspect of doing this is to minimize movement of data between processors. Thus, the SIMD parallel algorithms were designed without the use of sparse data structures.

Figure 3-16 illustrates the basic concept of the SIMD Gauss-Seidel AUCTION algorithm design. The SIMD architecture is viewed as a long vector of processors. Figure 3-16 shows a number of processors which equals the number of persons in the assignment problem; this was the case for the benchmark problems and the algorithms implemented in the DAP 510 and the Connection Machine CM-2. Viewing the benefits a_{ij} as a matrix, each

processor contains the j column of the matrix (that is, $\{a_{ij}, i = 1, ..., n\}$) and the price p_j of that column. In this manner, each processor can form independently the net profit $a_{ij} - p_j$. The maximum value of the net profit and the location of a maximal argument are obtained by reductions of the array of net profits into scalar values using the interprocessor communication network. Since the Gauss-Seidel AUCTION algorithm operates on only a single person i at a time, the relevant data is spread maximally across processors, thereby maximizing the potential speedup.

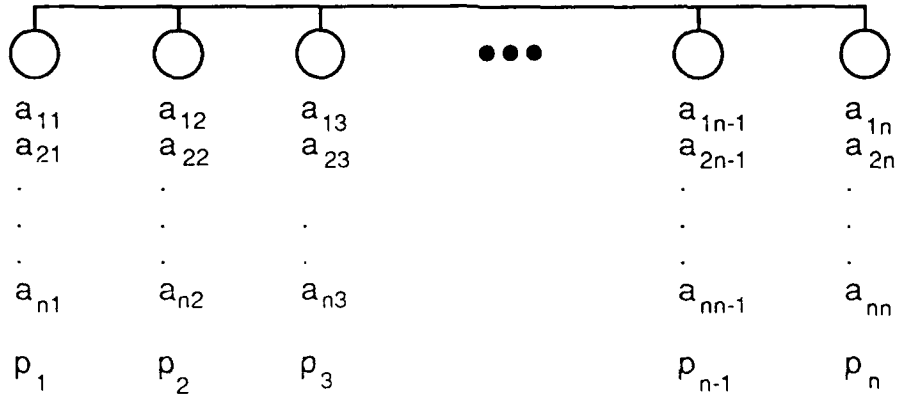

Figure 3-16. Illustration of the data mapping into processors for parallel SIMD Gauss-Seidel AUCTION algorithm. Each processor receives a column of the benefit matrix (corresponding to a single object), as well as the price of the object corresponding to that column.

With this data-mapping concept, we designed the parallel implementations of the SIMD Gauss-Seidel AUCTION algorithm on the various architectures by using the appropriate array language extensions to implement the array arithmetic and reduction operations. On the DAP 510, the array language used was Fortran Plus; on the Connection Machine CM-2, we used the C* language (its Fortran 8X compiler was still under development). In order to illustrate the array operations required, we provide Fortran 8X versions of the key computations required for a bid by person i, and discuss the similar operations required for implementation in C* and Fortran Plus.

In the bid phase of the Gauss-Seidel AUCTION algorithm, the critical computations are associated with computing v(i), j(i) and w(i) as in Eqs. 3-2, 3-3 and 3-4. The calculation of v(i) in Fortran 8X is programmed as:

```
MARGINS = A(i,:) - P
v(i) = maxval(MARGINS)
```

As may be seen, FORTRAN 8X permits direct calculation to be made on vectors and arrays. Thus, P and MARGINS are length n vectors, A is an n x n matrix, and v(i) is a scalar. The construct A(i,:) refers to the ith row of the matrix A. The function maxval is a reduction operator which returns the value of the largest element contained in its vector or array argument. Both the C* and Fortran Plus languages contain reduction operators (>?= in C*, maxv in Fortran Plus) which are equivalent to the above maxval operator.

Computation of j(i) can now be evaluated in Fortran 8X as:

```
MBIDS = MARGINS .eq. v(i)
TEMP = ∞
where(MBIDS) TEMP = INDICES
j(i) = minval(TEMP)
```

In this excerpt, MBIDS is a logical vector marking all occurrences of v(i) in MARGINS, INDICES is a vector storing the indices {1, 2, ..., n} and TEMP is set to the integer indices of these occurrences by the where statement. Thus, j(i) is the index of the first occurrence of v(i) in MARGINS. Again, both C* and Fortran Plus contain masked assignment operators corresponding to the Fortran 8X construct.

The remaining parameter required for the computation of a bid is w(i). The Fortran 8X code for the computation of the remaining parameter is given by:

```
w(i)= maxval(MARGINS, mask=INDICES.ne.j(i))
```

The additional feature of the Fortran 8X code to be noted here is that the maximum be taken of a specified subset of elements of MARGINS. This is accomplished by the keyword argument mask = <>. Similar masked reduction operators exist in C* and Fortran Plus.

Figure 3-17 illustrates the performance of the SIMD Gauss-Seidel AUCTION algorithms on the DAP 510 and the Connection Machine CM-2 for 800 person assignment problems as a function of feasible assignment density. For comparison, we have included the times of the sequential Gauss-Seidel AUCTION algorithm on the Encore Multimax and Fortran 8X implementation of the Gauss-Seidel AUCTION on the Alliant FX/8 using 8 vector processors. Note that the use of dense data structures in the SIMD algorithms results in a computation time which is effectively constant with feasible assignment density. In essence, only a fraction of the processors (equal to the feasible assignment density) are doing useful work in the sparse assignment problems. In contrast, the sequential Gauss-Seidel AUCTION algorithm uses sparse data structures, and its overall computation time is reduced significantly for sparse assignment problems. Figure 3-18 illustrates similar results for 1000 person assignment problems.

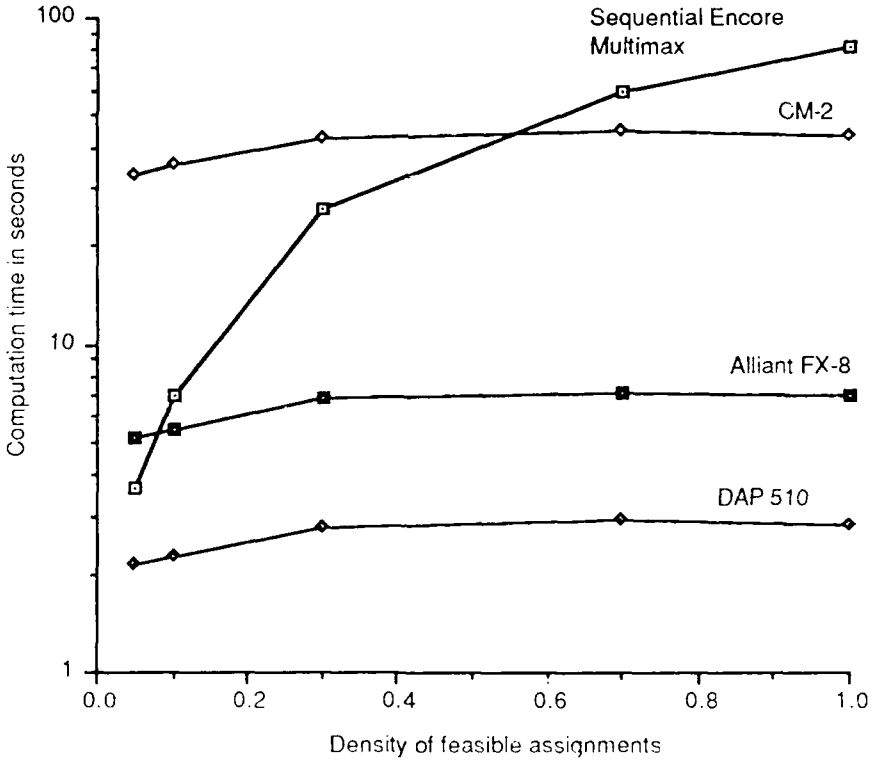

Figure 3-17. Computation times of SIMD Gauss Seidel AUCTION algorithms for 800 person assignment problems (benefit range [1,1000]) as a function of feasible assignment density.

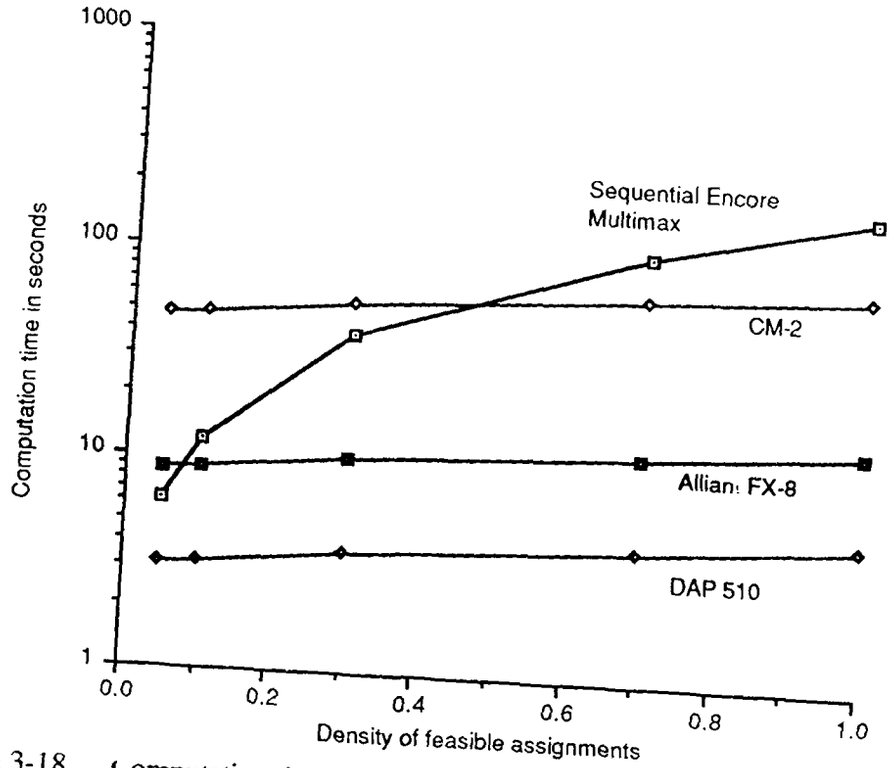

Figure 3-18. Computation times of SIMD Gauss Seidel AUCTION algorithms for 1000 person assignment problems (benefit range [1,1000]) as a function of feasible assignment density.

There are two important observations to make about the results of Figs. 3-17 and 3-18. First, the results on the CM-2 were obtained using a Sun 4 front end as the sequential processor; nevertheless, the performance of the Gauss-Seidel AUCTION algorithm on the CM-2 was substantially slower than the performance on the comparable DAP 510 using a similar number of processors. After consulting with Thinking Machines personnel, they suggested that a major source of inefficiency was the language that the algorithm was coded in. In essence, the C* compiler on the CM-2 is well-known to generate inefficient code, and is currently undergoing major revisions. In order to achieve optimal performance, Thinking Machines recommended the use of the C-PARIS assembly language; due to time limitations, writing such assembly code was beyond the scope of this effort and remains a topic for future investigation. However, we point out that similar times were reported by Phillips and Zenios [25] in their implementation of the Jacobi AUCTION algorithm (using both Gauss-Seidel and

Jacobi parallelization) in the C-PARIS language for assignment problems with comparable numbers of persons and similar benefit range.

The second observation is aimed at explaining the exceptionally fast performance of the DAP 510 on this class of algorithms. In essence, the DAP 510 communications architecture allows it to execute reduction operations such as minval in a time which is independent of the number of processors used for the reduction operation. Furthermore, these reductions use a specific bit-level algorithm across all processors which allows the computation of the minimum of 1024 32 bit numbers in about 12 microseconds.

To illustrate this algorithm, consider taking the minimum of the following list of 4 numbers: (6, 2, 10, 2). The binary representation of these numbers is

Decimal	Binary
6	0110
2	0010
10	1010
2	0010

The minval routine on the DAP 510 employs only logical bit operations and succeeds in locating all occurrences of the smallest value. Specifically, this routine computes a vector of bits (of length equal to the number of elements) whose 0-bits locate the minima present in the list. Note that, in the list of binary numbers above, the column of most-significant bits 0010 contains the information that the third number in the list cannot be the smallest. Therefore, 0010 locates the minimum as being among the first, second and fourth numbers in the list. In a second application of the same reasoning, note that a Boolean OR combination of the first column with the second column (0010 OR 1000 = 1010) further narrows the choices for minimum to the second and fourth numbers.

The complete algorithm for minval on the DAP 510 is essentially equivalent to OR together all of the bit columns of the analogous binary representation of a list of numbers, starting at the most significant end. Some care must be taken in order to avoid obtaining a

vector of all 1's. Whenever the running result comes up all ones, its previous value must be used to continue the algorithm. That would be necessary, for example, in the next step of the sample calculation above. Detecting such a condition can be done efficiently in the DAP 510 because of its ability to efficiently test bits across all processors.

A similar approach is used for implementing the maxval reduction operation which is used in the AUCTION algorithm. Thus, as long as the number of processors is larger than the number of objects, the DAP architecture provides a near-optimal match to the computation requirements of the Gauss-Seidel AUCTION algorithm for dense assignment problems.

There are several unresolved issues associated with the use of SIMD architectures for implementation of the AUCTION algorithm. The first issue involves the potential use of sparse data structures. For large sparse assignment problems, a lot of the available memory is wasted in each processor when using dense data structures. However, using sparse data structures will require data movements, which will reduce the efficiency of the SIMD architectures. For applications using sparse data structures, the more flexible communication network structure of the CM-2 (versus the grid structure of the DAP 510) may offer some advantages.

The second issue concerns whether a combination of the speedups possible from Gauss-Seidel and Jacobi parallelism on a SIMD architecture is possible. This requires the ability to compute bids for multiple persons simultaneously. Although such an arrangement is possible on the Connection Machine CM-2 by careful arrangement of the data across different processors (see [25] for a discussion), the persons which will be unassigned at any one iteration are not known apriori, so that, in practice, data movements among processors may be required to achieve this combination. Again, this would lead to a decrease in the overall efficiency of the parallel SIMD AUCTION algorithm.

In spite of these unresolved issues, SIMD architectures offer the promise of significant computation reduction for large assignment problems. Figures 3-17 and 3-18 illustrate that the Gauss-Seidel AUCTION algorithm was nearly two orders of magnitude faster on the DAP 510

than on a sequential processor. Figures 3-19 and 3-20 illustrate the relative performance of the DAP 510 algorithm when compared with the fastest MIMD algorithms on the Alliant FX/8 and the Encore Multimax. Even for very sparse problems (density 5%), the computation time on the DAP 510 using dense data structures was comparable to the computation times achieved by the fastest parallel MIMD algorithms.

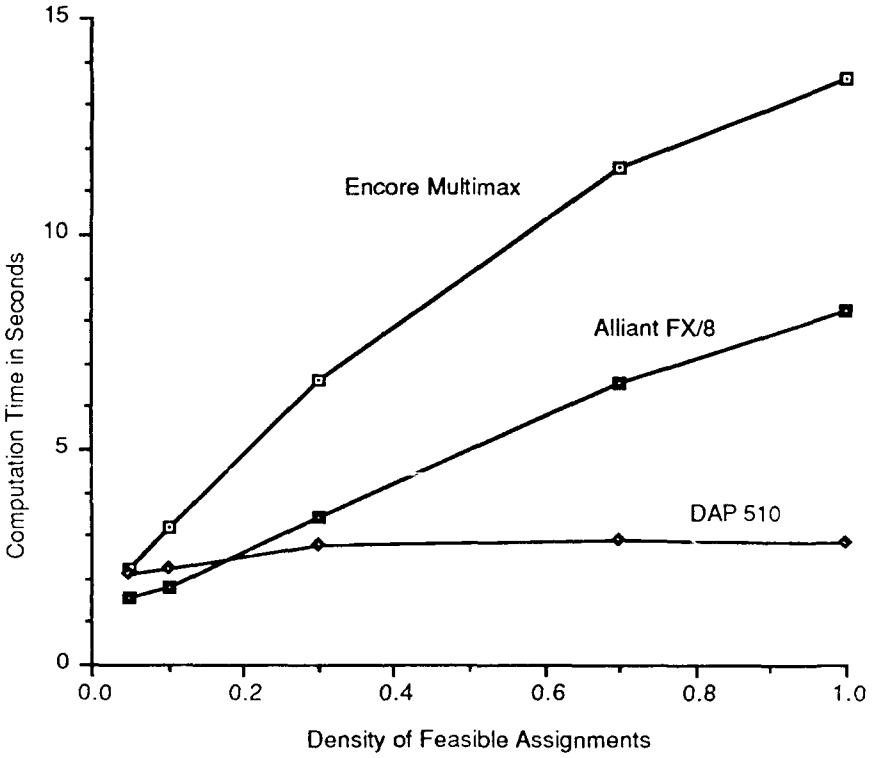

Figure 3-19. Performance of best MIMD and SIMD Gauss-Seidel AUCTION algorithms for 800 person assignment problems, benefit range [1,1000]

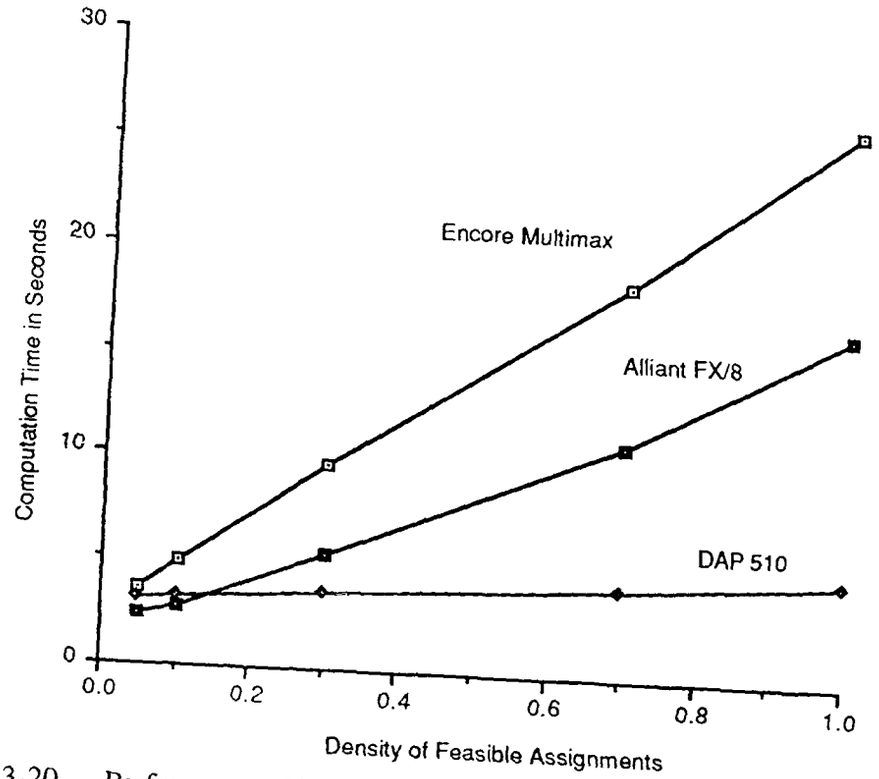

Figure 3-20. Performance of best MIMD and SIMD Gauss-Seidel AUCTION algorithms for 1000 person assignment problems, benefit range [1,1000]

SECTION 4

ASYNCHRONOUS PARALLEL AUCTION ALGORITHMS

4.1 INTRODUCTION

In the previous Section, we discussed our designs of parallel AUCTION algorithms for implementation on MIMD and SIMD machines. The design of these parallel algorithms included sufficient synchronization in order to guarantee that the bids generated by the parallel and sequential algorithms were identical. However, this synchronization often prevents efficient distribution of the computational load across processors, thereby reducing the efficiency of the parallel AUCTION algorithms.

The AUCTION algorithm is a natural candidate for asynchronous implementation, as discussed in Appendix A. In an asynchronous implementation, bid calculations may be done with out-of-date object price information and the highest bidder awards and subsequent price adjustments may be done with out-of-date bid information. The potential advantage of an asynchronous implementation is a reduction of the, so-called, *synchronization overhead.* This is the delay incurred when several processors synchronize to calculate in parallel a single person bid, when several processors calculating separate person bids in parallel, wait to make sure that up-to-date price information is available, and when the processors calculating in parallel the highest bidder awards wait for all bids to come in. Asynchronous algorithms are discussed in detail in [28], which gives many other references.

In this section we explore the merits of various asynchronous implementations of the AUCTION algorithm in a shared memory MIMD multiprocessor: the Encore Multimax. The validity of such an asynchronous implementation is established in Appendix A. We compare the performance of the synchronous and asynchronous implementations of the AUCTION algorithm, in an effort to quantify the tradeoffs between Jacobi and Gauss-Seidel parallelization, as well as the effects of asynchronism. To our knowledge, this is the first work

to report on the practical performance of asynchronous versions of the AUCTION algorithm in a real parallel machine.

4.2 ASYNCHRONOUS IMPLEMENTATION OF THE AUCTION ALGORITHM

In this subsection, we describe the asynchronous implementations of the AUCTION algorithm using the model for asynchronous computation described in Appendix A. As in the synchronous AUCTION algorithms, we describe the asynchronous algorithms in terms of the bid phase and the auction phase of each iteration. The difference between the synchronous and asynchronous algorithms is that the information used in the bid and auction phases may be out of date, as discussed in Appendix A.

In our asynchronous implementations, the bid calculations for a person i are divided into two types of tasks: search tasks, corresponding to searching a subset of the feasible objects A(i), and bid tasks, corresponding to merging the results generated by the various search tasks corresponding to person i and generating a bid for person i. These tasks are organized in a first in -- first out queue. When a processor becomes free it starts executing the top task of the queue if the queue is nonempty and otherwise it checks whether a termination condition is satisfied. The algorithm stops when all processors encounter the termination condition. Similarly to the synchronous Gauss-Seidel implementation, each set of admissible objects A(i) is divided in k groups of objects $A_1(i)$, ..., $A_k(i)$. The calculation of the bid of a person i is divided into k tasks, where each task involves a different group of objects. To perform one of these tasks, a processor must calculate and store in memory the best value, second best value, and best object within the corresponding object group.

In addition to the search tasks, a bid task is created for each unassigned person i. This bid task reads the results of the individual searches stored in memory and completes the bid of person i by merging the individual group search results, that is, by finding the best object and bid for person i based on the currently stored group results. The bid task also includes raising

the price of the best object and changing the assignment of the object (assuming the calculated bid is larger than the best object's price by at least ε).

There are two sources of asynchronism in this implementation. First, it is possible for some prices to be changed between the time a search task is completed and the time the results of that task are used to calculate a person bid. Second, it is possible that the bid task of a person is carried out before some of the search tasks associated with that bid are completed. In both cases the bid may reflect out-of-date price information and may prove ineffective in that it yields a bid that does not exceed the corresponding best object's price by at least ε. The advantage of the asynchronous implementation is that processors do not remain idle waiting to get synchronized with other processors or waiting for merging tasks to be completed.

The above implementation can be specialized to implement asynchronous algorithms which are equivalent to the Gauss Seidel, Jacobi and Hybrid AUCTION algorithms by controlling the number of search tasks generated for each unassigned person and the distribution of tasks among processors. If a single search task is generated per unassigned person, and this search task and its corresponding bid task are assigned to a single processor, the resulting algorithm corresponds to an asynchronous implementation of the Jacobi AUCTION algorithm. If the number of search tasks per unassigned bidder is equal to the number of processors, the resulting algorithm is an asynchronous implementation of the Gauss-Seidel AUCTION algorithm. Asynchronous hybrid variations are obtained by modifying the ratio of the number of processors used to the number of search tasks generated per unassigned bidder. In the following subsections, we discuss the results obtained from our implementations of the asynchronous Jacobi AUCTION algorithm and two asynchronous Hybrid AUCTION algorithms.

4.3 ASYNCHRONOUS JACOBI AUCTION ALGORITHM

The asynchronous Jacobi AUCTION algorithm design is aimed at reducing the overall synchronization overhead by allowing bids to be computed based on older values of the object

prices. Specifically, processors start computing new bids without waiting for other processors to complete their price updates. Some synchronization is still required to guarantee that the prices of each object are changed in an appropriate order, and to guarantee that each processor computes the bid of a different person. This synchronization is implemented using locks on each object and a lock on the queue of unassigned persons; these locks allow only one processor at a time to modify the price of a given object, and only one processor at a time to update the queue of unassigned persons. Figure 4-1 illustrates the design of the asynchronous Jacobi AUCTION algorithm. In order to reduce contention for the locks, when the number of persons in the unassigned persons queue is lower than the number of processors, excess processors are diverted to a barrier to wait for a new ε-scaling cycle.

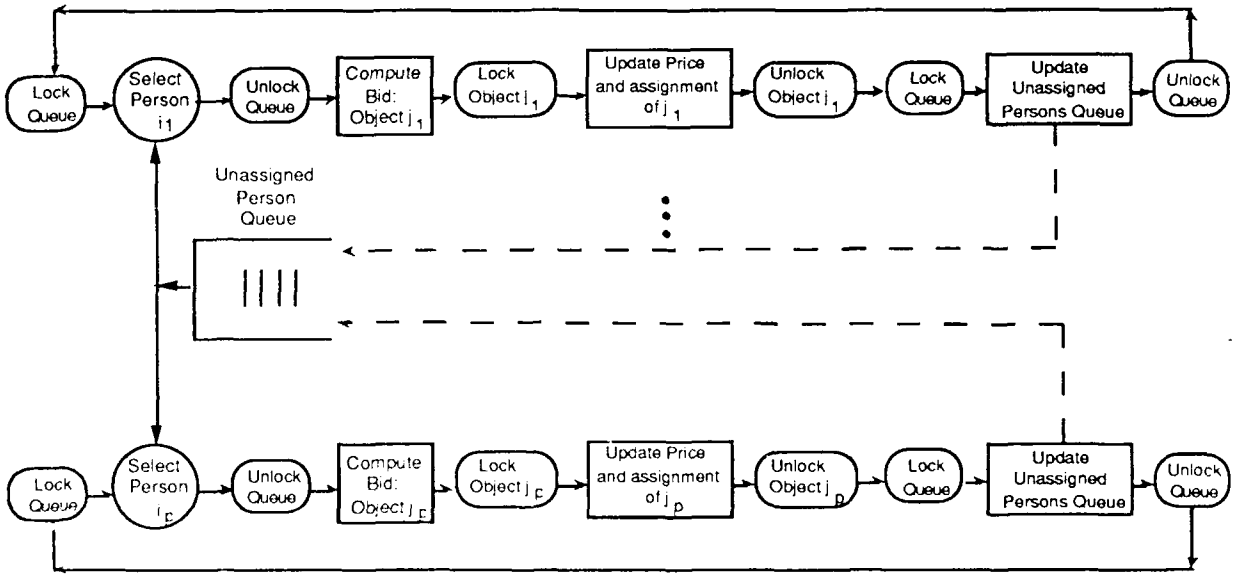

Figure 4-1. Design of Asynchronous Jacobi AUCTION algorithm. Locks on each object and on the unassigned persons queue are used to guarantee data integrity and preserve complementary slackness.

The performance of the asynchronous Jacobi AUCTION algorithm is illustrated in Fig. 4-2. The numbers shown represent an average of three runs; the actual running time of the algorithm varies from run to run because the order in which different processors complete their bids and acquire the locks affects the order in which objects are inserted into the unassigned

persons queue. A different ordering of persons produces a different auction process, which affects the total computation time. The curves in Fig. 4-2 represent the total computation time, and the number of bidding iterations performed by the parent process. Note the close correlation between these two curves, indicating a minimal amount of synchronization overhead. Note furthermore that the computation times are reduced to nearly 7.4 seconds, which represents a 28% improvement over the minimum times achieved by the synchronous Jacobi AUCTION algorithm in Section 3.2.2. This improvement is achieved because of the improved load balance among processors, as processors do not wait idly for other processors to complete their bidding process. Note that there is no apparent slowdown of the achievable processor efficiency with increased number of processors, unlike the performance of the synchronous Jacobi AUCTION algorithm.

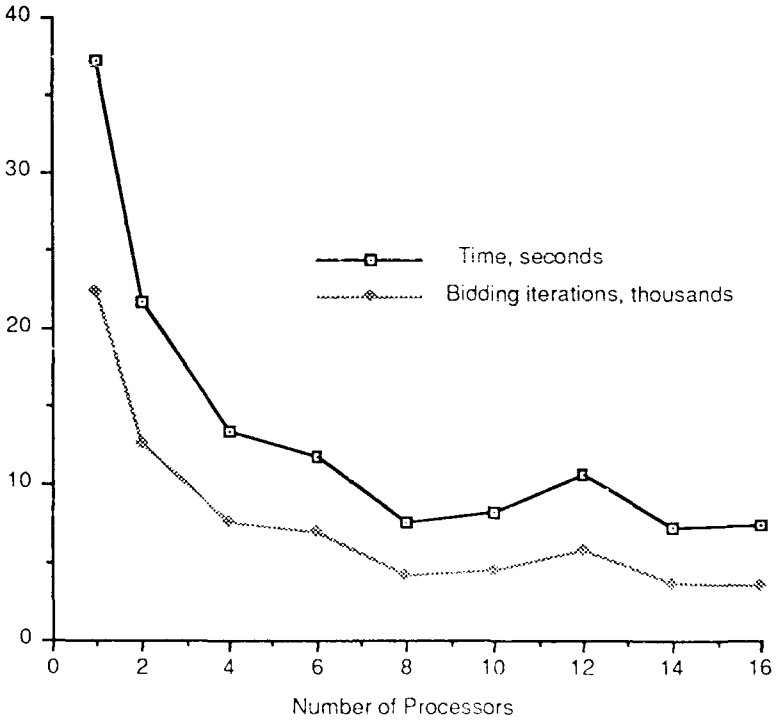

Figure 4-2. Performance of the asynchronous Jacobi AUCTION algorithm for 1000 person, 20% dense assignment problem, benefit range [1,1000]. The number of iterations by the parent processor are also indicated.

4.4 ASYNCHRONOUS HYBRID AUCTION ALGORITHMS

The results obtained using the asynchronous Jacobi AUCTION algorithm indicate that reducing the synchronization per iteration can improve the performance of the parallel AUCTION algorithms. The designs of the asynchronous Hybrid AUCTION algorithms were aimed at developing asynchronous algorithms which effectively combined the speedups of Jacobi and Gauss-Seidel parallelization. The designs of the two asynchronous algorithms differ slightly, and follow closely the theory of the asynchronous AUCTION algorithm presented in Appendix A.

Figure 4-3 illustrates the design of the asynchronous Hybrid AUCTION algorithms. Instead of an unassigned person queue, there is a queue of unassigned search tasks and bid tasks. Each unassigned person is represented by S search tasks and one bid task in this queue, ordered consecutively in the queue, so that the bid task follows the S search tasks. Different types of asynchronous algorithms can be generated by controlling the number of search tasks generated for each unassigned person. As before, a synchronization lock is required to allow tasks to be read and generated one at a time.

Figure 4-3 illustrates the processing of a single processor. After reading a task from the task queue, the processor determines whether it is a search task or a bid task. If it is a search task for bidder i, the processor searches the appropriate segment of the objects A(i) and writes a message in shared memory with the results of its search (the two highest net profit levels and the object offering the highest net profit). The message is protected by a lock indexed by the processor index and the person index, which guarantees that the message must be read in its entirety by the bid task. After writing the message, the processor releases the lock and attempts to acquire another task.

If the task acquired is a bid task, the processor must read the messages left by the search tasks for this person. Some of these search tasks may still be in process, so the bid processor may be reading old messages. The processor locks each message, reads the

contents, releases the lock and merges the results of the individual search tasks into an overall search result. This is then used to compute a bid (from person i to object j). The processor then locks object j, updates the price and assignment of object j and releases the object. If an unassigned person results from this operation, the processor then locks the unassigned task queue, inserts S search tasks and one bid task at the end of the queue for the unassigned person, and releases the queue.

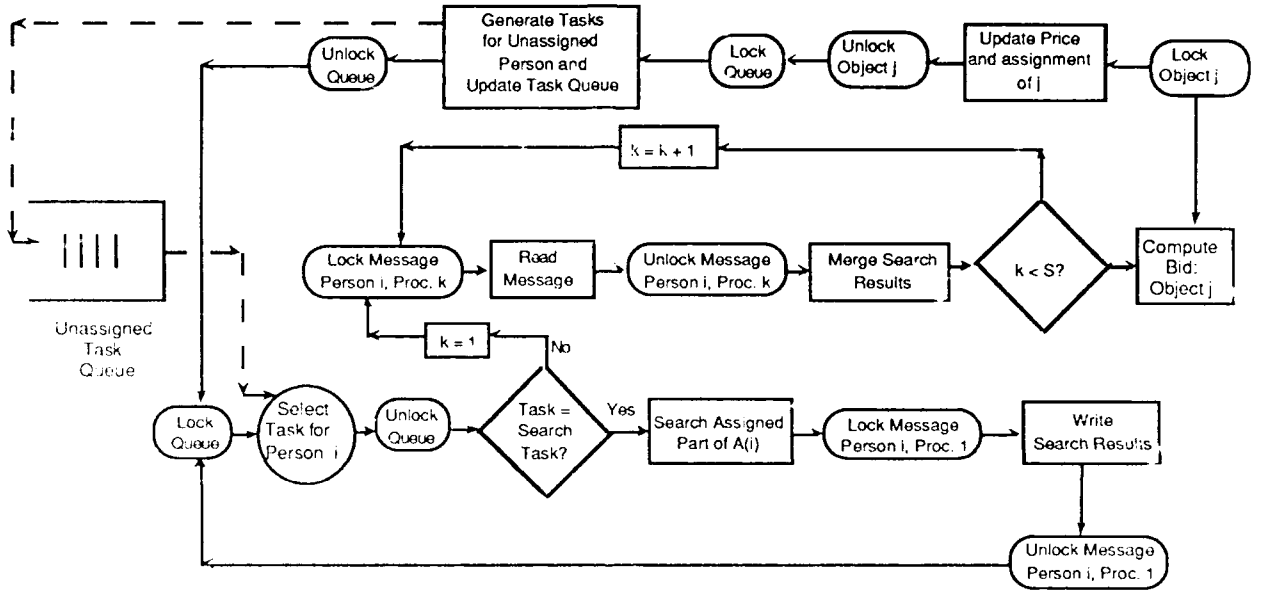

Figure 4-3. Design of the asynchronous Hybrid AUCTION algorithm.

The algorithm described above is the asynchronous Hybrid AUCTION II algorithm. The difficulty with this algorithm is that a bid is often computed based on outdated messages, leading to a large increase in the number of losing bids (and therefore the number of iterations required for convergence). Ideally, the bid task for person i would wait for the search tasks for person i to be completed; however, this requires synchronization. In the Hybrid AUCTION I algorithm, the processor that acquires the last search task corresponding to a person also acquires the bid task corresponding to that person. This processor executes the search task first, then the bid task. In this manner, the likelihood that the other search tasks corresponding to that person are complete by the time the bid task is executed is substantially increased.

Figure 4-4 illustrates the performance of one variation of the asynchronous Hybrid AUCTION I algorithm for the same 1000 person, 20% dense assignment problem. In this variation, the number of search tasks generated per person is equal to half the total number of processors used. In this manner, the results are comparable to the results obtained using the synchronous Hybrid AUCTION algorithm with two computed bids simultaneously. As before, the number of iterations required for convergence depends on the order in which the processors complete their tasks, and varies between different executions of the algorithm. The times shown are the average times of three runs. Contrasting the results of Fig. 4-4 with those of Fig. 3-12, we see that the asynchronous Hybrid AUCTION I algorithm achieves nearly a 30% reduction in computation time over the corresponding synchronous Hybrid AUCTION algorithm. Notice that the minimal times of both curves occur around 10 processors; adding additional processors increases the overhead for merging the results of additional searches, thereby detracting from overall performance in both the synchronous and asynchronous cases. The computation reduction of the asynchronous algorithm is due to improvements in load-balancing and reduced synchronization overhead. Load balance among processors is improved by having search tasks conducted in parallel with bid tasks, thereby keeping the majority of the processors performing useful computations. Reduced synchronization is accomplished by removing global synchronization primitives such as barriers and monitors, instead replacing these by locks on the specific data items (such as messages) for which integrity must be maintained.

Figure 4-5 illustrates the performance of the asynchronous Hybrid AUCTION I algorithm using 16 total processors as the numbers of bid and search tasks are varied. The goal of the Hybrid AUCTION I algorithm is to obtain a multiplicative combination of Jacobi and Gauss-Seidel speedups; the results of Fig. 4-5 indicate that the asynchronous Hybrid AUCTION I algorithm has approached close to a multiplicative combination for the optimal choices of numbers of processors and search tasks. There is a noticeable dropoff in the

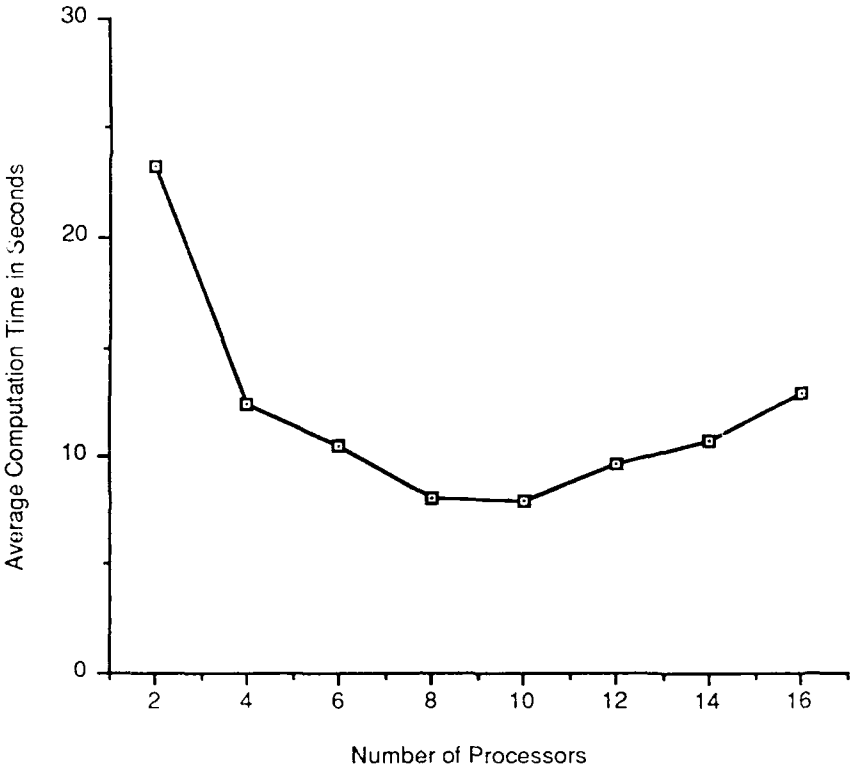

Figure 4-4. Average computation time of asynchronous Hybrid AUCTION I for 1000 person, 20% dense assignment problem, benefit range [1,1000]. The times shown are the average of three different runs on the Encore Multimax. In these problems, the number of search tasks per bid was equal to 1/2 the number of processors.

combined effectiveness when large numbers of search tasks are generated for each bid. The reason for this dropoff is that the total synchronization overhead associated with each iteration increases because the overall length of the task queue increases; this length is equal to the number of search tasks per bid times the number of bids required for the algorithm to converge, and thus grows linearly with the number of search tasks. Since synchronization (using locks) is used to maintain the integrity of the task queue, the synchronization overhead increases as the number of search tasks per bid increases for a fixed number of processors.

As the results of Fig. 4-5 indicate, the asynchronous Hybrid AUCTION I algorithm approached a successful combination of the speedups possible from Jacobi and Gauss-Seidel parallelism through a careful management of the order in which tasks are selected for processing. In order to illustrate the effects of more general asynchronous implementations,

we designed the asynchronous Hybrid AUCTION II algorithm, which was identical to the Hybrid AUCTION I algorithm except that the bid tasks would be assigned to the first available processor after all the corresponding search tasks for that bid had been assigned (as opposed to assignment to the same processor which selected the last search task). Figure 4-6 illustrates the relative performance (averaged across three runs) of the asynchronous Hybrid AUCTION I and II algorithms for the same 1000 person assignment problem. In these experiments, the two search tasks per bid are generated. Clearly, the asynchronous Hybrid AUCTION II algorithm is nearly twice as slow as the asynchronous Hybrid AUCTION I algorithm. The reason for this behavior is illustrated in Fig. 4-7, which describes the number of bids generated by each algorithm for convergence to an optimal solution. In essence, the number of bids required more than doubles for the asynchronous Hybrid AUCTION II algorithm! This is because the bid task is generating the bids before the search tasks have completed their scans;

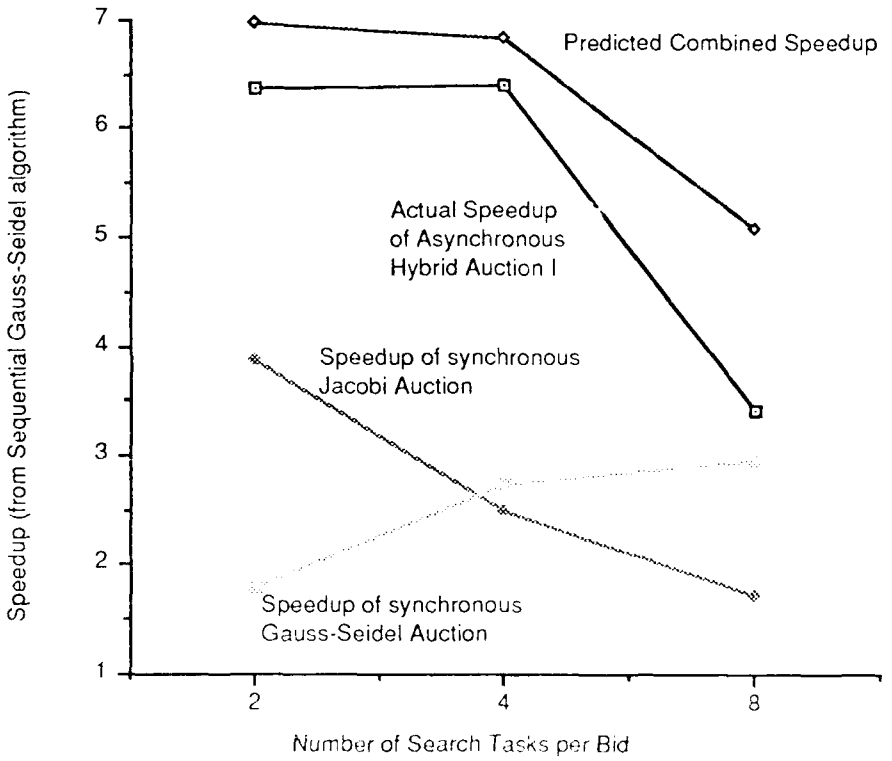

Figure 4-5. Comparison of predicted and actual speedups achieved by the asynchronous Hybrid AUCTION I algorithm for 1000 person, 20% dense assignment problem, benefit range [1,1000].

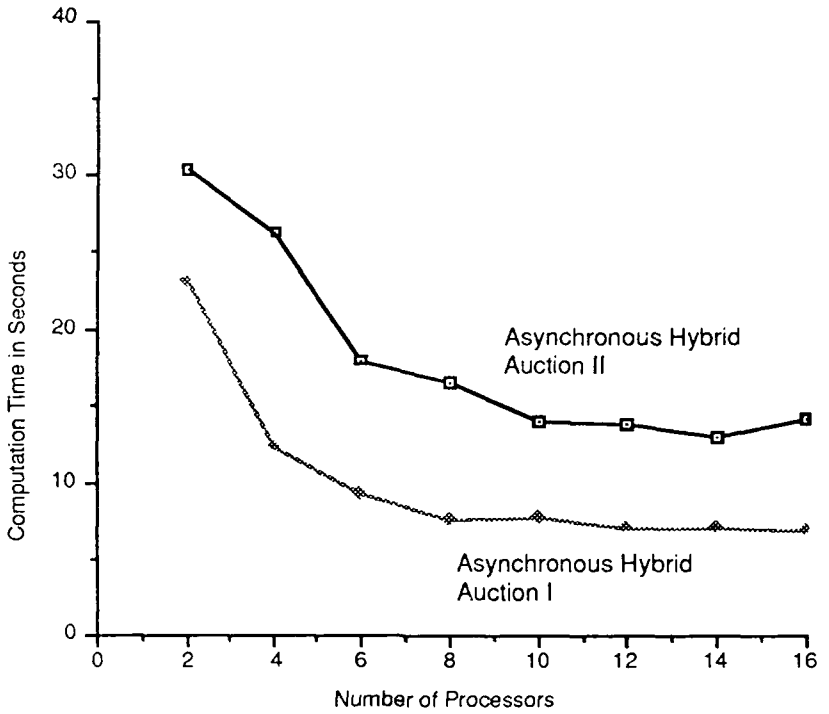

Figure 4-6. Performance of different asynchronous Hybrid AUCTION algorithms for 1000 person, 20% dense assignment problems, benefit range [1,1000].

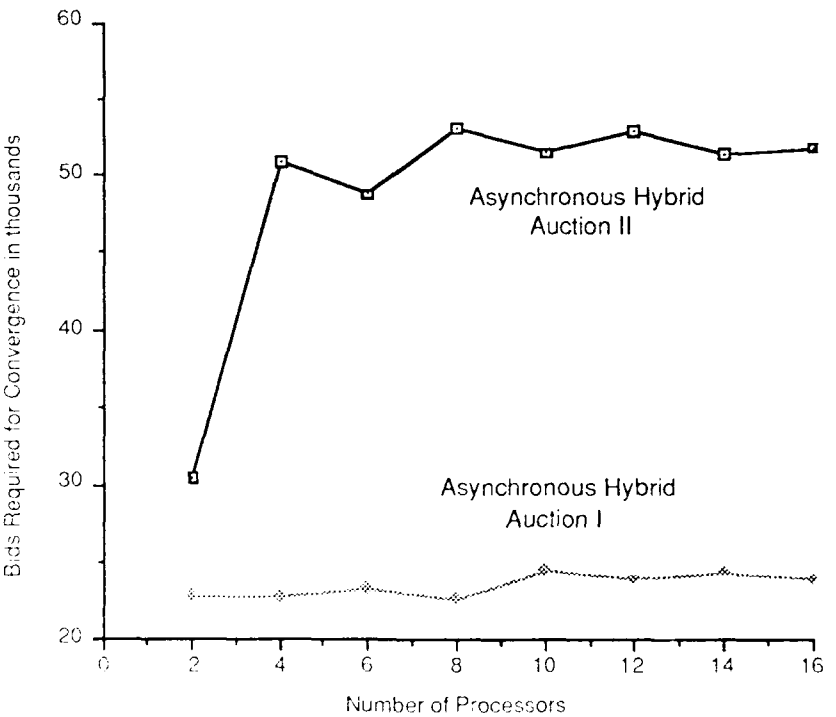

Figure 4-7. Number of bids required for convergence to optimal assignments by different asynchronous Hybrid AUCTION algorithms for 1000 person, 20% dense assignment problems, benefit range [1,1000].

these bids based on old information are often rejected, so that additional bids are required. The results illustrate the importance of careful management of asynchronous tasks in order to guarantee that the processors are doing useful work (i.e. work that will not become irrelevant when new information is acquired.)

Figure 4-8 compares the performance of the synchronous Hybrid AUCTION algorithm with the performance of the asynchronous Hybrid AUCTION I algorithm for 1000 person assignment problems with varying feasible assignment density. In these experiments, the number of search tasks generated per unassigned bidder was equal to half the number of processors selected; on the average, only two simultaneous bids were computed by the asynchronous algorithm, making it comparable to the synchronous Hybrid AUCTION algorithm. The computation times of the asynchronous algorithm are averaged across 3 runs. Note the significant reduction in computation time achieved by the asynchronous algorithm; this improvement reflects the improvement in load balancing across the multiple processors used.

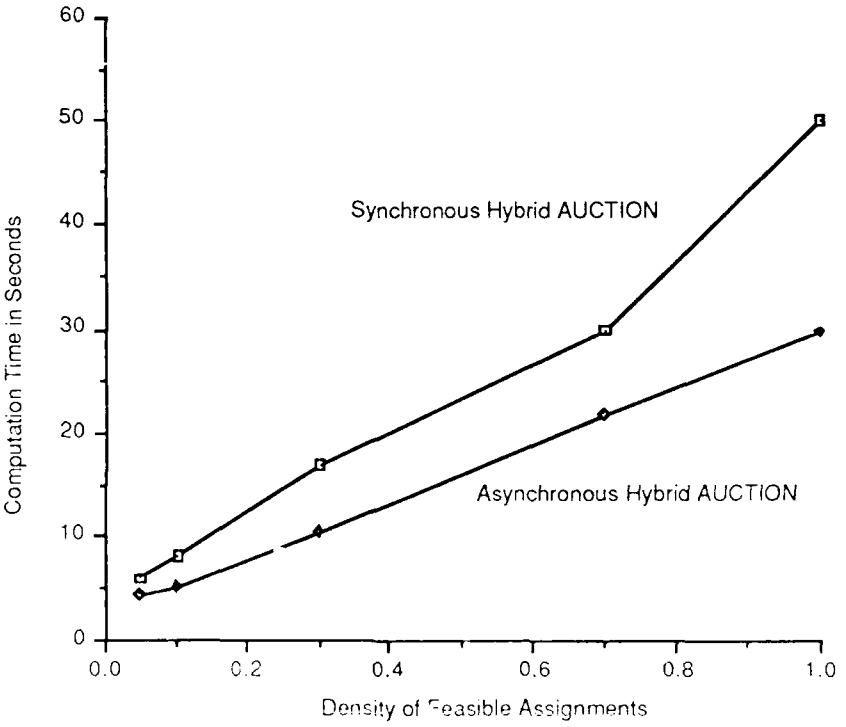

Figure 4-8. Performance of synchronous Hybrid AUCTION and asynchronous Hybrid AUCTION I algorithms on 1000 person assignment problems of varying density, benefit range [1,1000].

REFERENCES

1. Castañon, D., N. R. Sandell, W. M. Stonestreet, M. Athans, J. G. Eoll and K. A. Hatch, Advanced Weapon Target Assignment Algorithms Program: Systems Analysis Report, ALPHATECH report TR-427 on Army Strategic Defense Command Contract DASG60-86-C-0050, March 1989.

2. Castañon, D. et. al., Advanced Weapon Target Assignment Algorithms Program: Final Report, ALPHATECH report on Army Strategic Defense Command Contract DASG60-86-C-0050, July, 1989.

3. Garey, M.R., and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.

4. Lloyd, S.P., and H.S. Witsenhausen, "Weapons Allocation is NP-Complete," Proc. of 1986 Summer Conference on Simulation, Reno, Nevada, July 1986.

5. Dantzig, G.B., Linear programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

6. denBroder, G. G., R. E. Ellison and L. Emerling, "On Optimum Target Assignment," Operations Research, Vol. 7, 1959.

7. Kattar, J.D. (1986). Solution of the KKV Multi-Weapon, Multi-Target Assignment Problem, working paper WP-26597, the MITRE Corporation (Bedford Operations). Bedford, Massachusetts.

8. Chang, S.C., R. M. James and J. J. Shaw, "Assignment Algorithms for Kinetic Energy Weapons in Boost Phase Defense," Proceedings of the 26th IEEE Conf. on Decision and Control, Los Angeles, California, Dec. 1987.

9. Shaw, J.J., et al., "Battle Management Structures, Kinetic Energy Weapon, Weapon Target Assignment Algorithms - Final Report,", Volume II, TR-360, ALPHATECH, Inc., Burlington, MA, January 1988.

10. "Sixth Month Demo Weapon Target Pairing (U)," Contract F19628-88-C-0007, Sparta, Inc., Laguna Hills, California, June 1988, SECRET.

11. James, R.M., S. C. Chang and J. J. Shaw, "Weapon-Target Assignment for Boost Phase Defense," Proceedings of the Joint Director of Laboratories Command and Control Research Symposium, Washington, D. C. June 1987.

12. "Weapons Allocation and RF Spectrum Management: Algorithm Specification," AT&T Technologies, submitted to Naval Research Laboratory, contract N00014-87-C-2035, Greensboro, North Carolina August 1987.

13. Preston, F. L. and W. A. Metler, "An Algorithmic Solution to the Weapons Allocation Problem," AT&T Bell Laboratories, submitted to Naval Research Laboratory, contract N00014-87-C-2035, Whippany, New Jersey, December 1988.

14. "EV-88 Integration and Implementation, Experiment Program, Software Detailed Design Document, Levels 1 & 2 Battle Management CSCI," contract DASG60-87-C-0068, TRW Defense Systems Group, Huntsville, Alabama, June 1988.

15. Papadimitriou, C.H., and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

16. Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem", Lab. for Information and Decision Systems Working Paper, M.I.T., March 1979.

17. Bertsekas, D.P., "The Auction Algorithm: a Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, 1988.

18. Bertsekas, D. P., and D. A. Castañon, "The Auction Algorithm for Transportation Problems", LIDS Report P-1850, M.I.T., Feb. 1989, to appear in Annals of Operations Research.

19. Sanderson, J., private communication.

20. Geyer, H. K., "Parallelization of ALPHATECH's Auction Algorithm," Argonne National Laboratory, 1987.

21. Payne, D. G. and J. C. Horvath, "Battle Management on the Hypercube: Concurrent Engagement Management," March, 1988.

22. Kempa, D. N., J. L. Kennington, and H. A. Zaki, "Performance Characteristics of the Jacobi and Gauss-Seidel Versions of the Auction Algorithm on the Alliant FX/8", Report OR-89-008, Dept. of Mech. & Ind. Eng., Univ. of Illinois, Champaign-Urbana, 1989.

23. Balas, E., D. Miller, J. Pekny and P. Toth, "A Parallel Shortest Path Algorithm for the Assignment Problem," Management Science Research Report MSRR 552, Carnegie Mellon University, Pittsburgh, PA, April, 1989.

24. Kennington, J. and Z. Wang, "Solving Dense Assignment Problems on a Shared Memory Multiprocessor," Technical Report 88-OR-16, Dept. of Operations Research & App. Science, Southern Methodist University, October 1988.

25. Phillips, C., and S. A. Zenios, "Experiences with Large Scale Network Optimization on the Connection Machine", Report 88-11-05, Dept.of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Phil., Penn., Nov. 1988.

26. Bertsekas, D. P. and D. A. Castañon, "Parallel Synchronous and Asynchronous Implementations of the AUCTION Algorithm," in preparation.

27. Ortega, J. M., and Rheinboldt, W. C., "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press, N. Y., 1970.

28. Bertsekas, D. P., and Tsitsiklis, J. N., Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J., 1989.

29. Boyle, J., Butler, R., Disz, T., Glickfield, B., Lusk, E., Overbeek, R., Patterson, J., and Stevens, R., Portable Programs for Parallel Processors, Holt, Rinehart & Winston, New York, 1987.

30. Dritz, K. W., and Boyle, J. M., "Beyond "Speedup": Performance Analysis of Parallel Programs", Argonne National Lab. Report ANL-87-7, Feb.1987.

APPENDIX A

THE AUCTION ALGORITHM

In this Appendix, we overview the theory of the AUCTION algorithm, describe a model for an asynchronous variation of the algorithm, and establish that this asynchronous variation obtains an optimal solution to the assignment problem.

A.1 THE AUCTION ALGORITHM FOR ASSIGNMENT PROBLEMS

In the assignment problem, n persons wish to allocate among themselves n objects, on a one-to-one basis. Each person i must select his object from a given subset A(i). There is a given benefit a_{ij} that i associates with each object j in A(i). An *assignment* is a set of k person-object pairs $(i_1,j_1), \ldots, (i_k,j_k)$, such that $0 \le k \le n$, $j_m \in A(i_m)$ for all k, and the persons $i_1, \ldots, i_k$ and objects $j_1, \ldots, j_k$ are all distinct. The total benefit B of the assignment is the sum of the benefits of the assigned pairs.

$$B = \sum_{m=1}^{k} a_{i_m j_m}$$

An assignment with is called *complete* (or *incomplete*) if it contains $k = n$ (or $k<n$, respectively) person-object pairs. We want to find a complete assignment with maximum total benefit, assuming that there exists at least one complete assignment. This is the classical assignment problem, studied algorithmically by many authors [A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, A.9, A.10, A.11, A.12, A.13, A.14, A.15], beginning with Kuhn's Hungarian method [A.16].

In the AUCTION algorithm, each object j has a price p_j with the initial prices being arbitrary. Prices are adjusted upwards as persons "bid" for their "best" object, that is, the object for which the corresponding benefit minus the price is maximal. Only persons without an object submit a bid, and objects are awarded to their highest bidder. In particular, the prices p_j are adjusted at the end of "bidding" iterations. At the beginning of each iteration, we have a

set of object prices and an incomplete assignment, and the algorithm terminates when a complete assignment is obtained. Each iteration involves of subset I of the persons that are unassigned at the beginning of the iteration. It has two phases:

> Bidding Phase: Each person $i \in I$ determines an object $j_i \in A(i)$ for which $a_{ij} - p_j$ is maximized over j, i.e.
>
> $$j_i = \arg\max_{j \in A(i)} (a_{ij} - p_j)$$
>
> and submits a bid $p_{j_i} + g_i$ for this object, where g_i is a positive bidding increment to be specified shortly.
>
> Assignment Phase: Each object j that receives one or more bids, determines the highest of these bids, increases p_j to the highest bid, and gets assigned to the person who submitted the highest bid. The person that was assigned to j at the beginning of the iteration (if any) is now left without an object (and becomes eligible to bid at the next iteration). If an object does not receive any bid during an iteration, its price and assignment status are left unchanged.

It can be shown that if the bidding increments g_i are bounded from below by some $\epsilon > 0$, this auction process terminates in a finite number of iterations with all persons having an object. To get a sense of this, note that if an object receives a bid in m iterations, its price must exceed its initial price by at least $m\epsilon$, while if an object is unassigned, its price has not yet changed from its initial value. Thus, for sufficiently large m, the object will become "expensive" enough to be judged "inferior" to some unassigned object by each person. It follows that there is a bounded number of iterations at which an object can be considered best and thus be preferred to all unassigned objects by some person. (This argument as stated, assumes that it is feasible to assign any person to any object but it can be generalized for the case where the set of feasible person-object pairs is limited, as long as there exists at least one feasible assignment; see e.g.[A.17, A.18].)

Whether the complete assignment obtained upon termination of the auction process is optimal depends strongly on the method for choosing the bidding increments g_i. In a real auction, a prudent bidder would not place an excessively high bid for fear the object might be won at an unnecessarily high price. Consistent with this intuition, one can show that if the

bidding increment g_i is small enough to ensure that even after the bid is accepted, the object will be "almost best" for the bidder, then the final assignment will be "almost optimal". In particular, we can show that if upon termination, we have

$$\max_j (a_{ij} - p_j - \varepsilon) \leq a_{ij_i} - p_{j_i} \quad \text{for all assigned pairs } (i,j_i) \tag{A-1}$$

(a property known as *ε-complementary slackness* or ε-CS for short), then the total benefit of the final assignment is within nε of being optimal. For a first principles derivation of this, note that the total benefit of *any* complete assignment $\{(i,j_i), i = 1, ..., n\}$ satisfies

$$\sum_{i=1}^{n} a_{ij_i} \leq \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_{j \in A(i)} (a_{ij} - p_j)$$

for any set of prices p_j, $j = 1, ..., n$, since

$$\sum_{i=1}^{n} \max_{j \in A(i)} (a_{ij} - p_j) \geq \sum_{i=1}^{n} (a_{ij_i} - p_{j_i})$$

$$\sum_{i=1}^{n} p_{j_i} = \sum_{j=1}^{n} p_j$$

Therefore, the optimal total assignment benefit cannot exceed the quantity

$$A^* = \min_{p_j,\, j=1,\ldots,n} \left\{ \sum_{i=1}^{n} p_j + \sum_{i=1}^{n} \max_{j \in A(i)} (a_{ij} - p_j) \right\} \tag{A-2}$$

On the other hand, if the ε-CS property (A-1) holds upon termination of the auction process, then by adding Eq. (A-1) over all i, we see that

$$\sum_{i=1}^{n} p_{j_i} + \sum_{i=1}^{n} \max_{j \in A(i)} (a_{ij} - p_j) \leq \sum_{i=1}^{n} a_{ij_i} + n\varepsilon \tag{A-3}$$

Since the left side above cannot be less than A*, which as argued earlier, cannot be less than the optimal total assignment benefit, we see that the final total assignment benefit is within nε

of being optimal. We note parenthetically that the preceding derivation is guided by duality theory; the assignment problem can be formulated as a linear programming problem, and the minimization problem in the right side of Eq. (A-2) is a dual problem (see e.g. [A.18, A.19]).

Suppose now that the benefits a_{ij} are all integer, which is the typical practical case (if a_{ij} are rational, they can be scaled up to integer by multiplication with a suitable common positive integer). Then, the total benefit of any assignment is integer, so if $n\varepsilon < 1$, a complete assignment that is within $n\varepsilon$ of being optimal must be optimal. It follows, that if $\varepsilon < 1/n$, the benefits a_{ij} are all integer, and the ε-CS condition (A-1) is satisfied upon termination, then the assignment obtained is optimal.

There is a standard method for choosing the bidding increments g_i so as to maintain the ε-CS condition (A-1) throughout the auction process, assuming this condition is satisfied by the initial prices and the initial assignment (as is trivially the case when no objects are assigned initially). In this method, ε is a fixed positive number, and the bidding increment g_i is given by

$$g_i = \varepsilon + v_i - w_i \tag{A-4}$$

where v_i is the best object value,

$$v_i = \max_{j \in A(i)} (a_{ij} - p_j) \tag{A-5}$$

and w_i is the "second best" object value

$$w_i = \max_{j \in A(i),\, j \neq j_i} (a_{ij} - p_j) \tag{A-6}$$

where j_i is a best object for which the maximum in Eq. (A-5) is attained. We will assume for convenience throughout that A(i) contains at least two objects, so the maximum in Eq. (A-6) is well defined.

A.2 COMPUTATIONAL ASPECTS -- ε-SCALING

The AUCTION algorithm exhibits interesting computational behavior and it is essential to understand this behavior in order to implement the algorithm efficiently. We first note that

the amount of work to solve the problem can depend strongly on the value of ε and on the maximum absolute object value

$$C = \max_{i,j} a_{ij} \tag{A-7}$$

Basically, for many types of problems, the number of bidding iterations up to termination tends to be proportional to C/ε. We note also that there is a dependence on the initial prices; if these prices are "near optimal", it can be expected that the number of iterations to solve the problem will be relatively small. This suggests the idea of *ε-scaling*, which consists of applying the algorithm several times, starting with a large value of ε and successively reducing ε up to an ultimate value which is less than the critical value $1/n$. Each application of the algorithm provides good initial prices for the next application.

In practice, it is a good idea to at least consider scaling. For sparse assignment problems, that is, problems where the set of feasible assignment pairs is severely restricted, scaling seems almost universally helpful. This was established experimentally at the time of the original proposal of the AUCTION algorithm [A.20]. There is also a related polynomial complexity analysis [A.18] that uses some of the earlier ideas of an ε-scaling analysis [A.9], for the ε-relaxation method of [A.21].

Our implementation of ε-scaling is as follows: the integer benefits a_{ij} are first multiplied by n+1 and the AUCTION algorithm is applied with progressively lower value of ε, up to the point where ε becomes 1 or smaller (because a_{ij} have been scaled by n+1, it is sufficient for optimality of the final assignment to have $\varepsilon < 1$). The sequence of ε values used is

$$\varepsilon(k) = \max(1, \Delta/\theta^k), \quad k = 0, 1, \ldots$$

where $\Delta > 0$ and $\theta > 1$ are parameters set by the user. Typical values that we used for sparse problems are $\Delta = C/4$ or $\Delta = C/2$, and $4 \leq \theta \leq 8$.

A.3 THE TOTALLY ASYNCHRONOUS VERSION OF THE AUCTION ALGORITHM

One may view a synchronous parallel algorithm as a sequence of consecutive computation segments called *phases*. The computations within each phase are divided in some way among the processors of a parallel computing system. The computations of any two processors within each phase are independent, so the algorithm is mathematically equivalent to some serial algorithm. Phases are separated by *synchronization points*, which are times at which all processors have completed the computations of a given phase but no processor has yet started the computations of the next phase. In asynchronous parallel algorithms, the coordination of the computations of the processors is less strict. Processors are allowed to proceed with computations of a phase with data which may be out-of-date because the computations of the previous phase are incomplete. An asynchronous algorithm may contain some synchronization points but these are generally fewer than the ones of the corresponding synchronous version.

To get a first idea of the totally asynchronous implementation of the AUCTION algorithm, it is useful to think of a person as an autonomous decision maker that obtains at unpredictable times information about the prices of the objects. Each unassigned person makes a bid at arbitrary times on the basis of its current object price information (that may be outdated because of communication delays). Furthermore, assignment of objects may be decided even if some potential bidders have not been heard from. There are basically two conditions that must be observed in order for this process to terminate properly. We state roughly these conditions below and we will give a more precise formulation shortly.

1. An unassigned person will bid for some object within finite time, and cannot bid twice (i.e., cannot bid for a second object while waiting for a reply regarding the disposition of an earlier bid for another object).

2. Whenever one or more bids are received that raise the price of an object, then, within finite time, that price must be updated, and its value must be communicated (not necessarily simultaneously) to all persons. Furthermore, a person that has lost his assigned object must be informed within finite time of the change in assignment

We now formulate the totally asynchronous model of the AUCTION algorithm, and we prove its validity. We denote

$p_j(t)$ = Price of object j at time t

$r_j(t)$ = Person assigned to object j at time t [$r_j(t) = 0$ if object j is unassigned]

U(t) = Set of unassigned persons at time t[$i \in U(t)$ if $r_j(t) \neq i$ for all objects j]

We assume that U(t), $p_j(t)$ and $r_j(t)$ can change only at integer times t; this involves no loss of generality, since t may be viewed as the index of a sequence of physical times. In addition to U(t), $p_j(t)$ and $r_j(t)$, the algorithm maintains at each time t, a subset $R(t) \subset U(t)$ of unassigned persons that may be viewed as having a "ready bid" at time t. We assume that by time t, a person $i \in R(t)$ has used prices $p_j(t_{ij}(t))$ from some earlier times $t_{ij}(t) \leq t$ to compute the best value

$$v_i(t) = \max_{j \in A(i)} (a_{ij} - p_j(t_{ij}(t))) \tag{A-8}$$

a best object $j_i(t)$ attaining the above maximum,

$$j_i(t) = \arg\max_{j \in A(i)} (a_{ij} - p_j(t_{ij}(t))) \tag{A-9}$$

the second best value

$$w_i(t) = \max_{j \in A(i),\, j \neq j_i(t)} (a_{ij} - p_j(t_{ij}(t))) \tag{A-10}$$

and has determined a bid

$$b_i(t) = p_{j_i}(t_{ij_i}(t)) + v_i(t) - w_i(t) + \varepsilon \tag{A-11}$$

The implication here is that unassigned persons i will enter the set R(t) and become eligible to bid after some computations, which update $j_i(t)$ and $b_i(t)$. However, to maximize the generality and flexibility of our model, the precise mechanism by which these computations are done is left unspecified subject to the following two assumptions:

Assumption 1: $U(t) \neq \Phi$ implies $R(t) \neq \Phi$ for some $t' > t$.

Assumption 2: For all i, j, and t, $\lim_{t \to \infty} t_{ij}(t) = \infty$.

Clearly an asynchronous AUCTION algorithm cannot solve the problem if unassigned persons stop submitting bids and if old information is not eventually discarded. This is the motivation for the preceding two assumptions. Initially, each person is assigned to at most one object, that is, $r_j(0) \neq r_{j'}(0)$ for all assigned objects j and j', and it will be seen that the algorithm preserves this property throughout its course. Furthermore, initially ε-CS holds, that is,

$$\max_k (a_{ik} - p_k(0) - \varepsilon) \leq a_{ij} - p_j(0) \quad \text{if } i = r_j(0)$$

It will be shown shortly that this property is also preserved during the algorithm.

At each time t, if all persons are assigned [$U(t)=\Phi$], the algorithm terminates. Otherwise, if $R(t) = \Phi$, nothing happens. If R(t) is nonempty the following occur:

1. A nonempty subset $I(t) \subset R(t)$ of persons that have a bid ready is selected.
2. Each object j for which the corresponding bidder set

$$B_j(t) = \{i \in I(t) \mid j = j_i(t)\} \tag{A-12}$$

is nonempty, determines the highest bid

$$b_j(t) = \max_{i \in B_j(t)} b_i(t) \tag{A-13}$$

and a person $i_j(t)$ for which the above maximum is attained:

$$i_j(t) = \arg\max_{i \in B_j(t)} b_i(t) \tag{A-14}$$

Then, the pair $[p_j(t), r_j(t)]$ is changed according to

$$[p_j(t+1), r_j(t+1)] = [b_j(t), i_j(t)] \text{ if } b_j(t) \geq p_j(t) + \varepsilon$$

$$= [p_j(t), r_j(t)] \text{ otherwise} \tag{A-15}$$

Notice that if $i_j(t) = i$ and $U(t) = R(t)$ for all t, then the asynchronous algorithm is equivalent to the synchronous version discussed in Section A.1.

The asynchronous model becomes relevant in a parallel computation context where some processors compute bids for some unassigned persons, while other processors simultaneously update some of the object prices and corresponding assigned persons. Suppose that a single processor calculates a bid of person i by using the values $a_{ij} - p_j(t_{ij}(t))$ prevailing at times $t_{ij}(t)$ and then calculates the maximum value at time t. Then, if the price of an object $j \in A(i)$ is updated between times $t_{ij}(t)$ and t by some other processor, the maximum value will be based on out-of-date information. The asynchronous algorithm models this possibility by allowing $t_{ij}(t) < t$. A similar situation arises when the bid of person i is calculated cooperatively by several processors rather than by a single processor.

The following proposition establishes the validity of the asynchronous AUCTION algorithm of this section.

Proposition 1: Let Assumptions 1 and 2 hold and assume that there exists at least one complete assignment. Then for all t and all j for which $r_j(t) \neq 0$, the pair $[p_j(t),r_j(t)]$ satisfies the ε-CS condition

$$\max_k (a_{ik} - p_k(t) - \varepsilon) \leq a_{ij} - p_j(t) \quad \text{if } i = r_j(t) \tag{A-16}$$

Furthermore, there is a finite time at which the algorithm terminates. The complete assignment obtained upon termination is within $n\varepsilon$ of being optimal, and is optimal if $\varepsilon < 1/n$ and the benefits a_{ij} are integer.

proof: *Let $[p_j(t),r_j(t)]$ be a pair with $r_j(t) \neq 0$. To simplify notation, let $i = r_j(t)$. We first* consider times t at which p_j was just updated, i.e., $p_j(t) > p_j(t-1)$ and $i \neq r_j(t-1)$, and person i submitted a highest bid for object j at time t-1. Then we have by construction

$$a_{ij} - p_j(t) = a_{ij} - b_i(t-1) = a_{ij} - p_j(t_{ij}(t-1)) - v_i(t-1) + v_i(t-1) - \varepsilon$$

$$= w_i(t-1) - \varepsilon \geq \max_{k \in A(i), k \neq j(i)} (a_{ik} - p_k(t_{ik}(t))) - \varepsilon$$

where the last inequality follows using the fact $p_k(t) \geq p_k(t')$ for all k and t, t' with $t \geq t'$. Therefore, the ε-CS condition (A-16) holds for all t at which p_j was just updated.

Next we consider times t for which p_j was not just updated. Let t' be the largest time which is less than t and for which $p_j(t') > p_j(t'-1)$; this is the largest time prior to t that object j was assigned to person i. By the preceding argument, we have

$$a_{ij} - p_j(t') \geq \max_{k \in A(i)} (a_{ik} - p_k(t')) - \varepsilon$$

and since $p_j(t')=p_j(t)$, and $p_k(t) \geq p_k(t')$ for all k, the ε-CS condition (A-16) again follows.

We next show that the algorithm terminates in finite time. We first note the following:

a. Once an object is assigned, it remains assigned for the remainder of the algorithm. Furthermore, an unassigned object has a price equal to its initial price. Using Eqs. (A-8) and (A-10), we have $w_i(t) \leq v_i(t)$, so from Eq. (A-11) we see that $b_j(t) \geq p_{j_i}(t_{ij_i}(t)) + \varepsilon$. It follows from Eq.(A-13) that if person i bids for object j at time t, we must have

$$b_j(t) \geq p_j(t_{ij}(t)) + \varepsilon \qquad \text{(A-17)}$$

b. Each time an object j receives a bid $b_j(t)$ at time t, there are two possibilities: either $b_j(t) < p_j(t)+\varepsilon$, in which case $p_j(t+1)=p_j(t)$, or else $b_j(t) \geq p_j(t)+\varepsilon$, in which case $p_j(t+1) \geq p_j(t)+\varepsilon$ and $p_j(t)$ increases by at least ε[cf. Eq. (A-15)]. In the later case we call the bid *substantive*. Suppose that an object receives an infinite number of bids during the algorithm. Then, an infinite subset of these bids must be substantive; otherwise $p_j(t)$would stay constant for t sufficiently large, we would have $p_j(t_{ij}(t)) = p_j(t)$ for t sufficiently large because old price information is eventually purged from the system (cf. Assumption 2), and in view of Eqs.(A-15) and (A-17) we would have $p_j(t+1) \geq p_j(t) + \varepsilon$ for all times t at which j receives a bid, arriving at a contradiction.

Assume now, in order to obtain a contradiction, that the algorithm does not terminate finitely. Then, because of Assumption 1, there is an infinite number of times t at which R(t) is nonempty and at each of these times, at least one object receives a bid. Thus, there is a nonempty subset of objects J^* which receive an infinite number of bids, and a nonempty subset of persons I^* which submit an infinite number of bids. In view of (c) above, the prices of all objects in J^* increase to ∞, and in view of (a) above all objects in J^* are assigned for t

sufficiently large. Furthermore, the prices of all objects $j \notin J^\infty$ stay constant for t sufficiently large and since old information is purged from the system (cf. Assumption 2), we also have $p_j(t_{ij}(t)) = p_j(t)$ for all $i,j \notin J^\infty$, and t sufficiently large. These facts imply that for sufficiently large t, every object $j \in A(i)$ which is not in J^∞ would be preferable for person i to every object $j \in A(i) \cap J^\infty$. Since the ε-CS condition (1) holds throughout the algorithm, we see that for each person $i \in I^\infty$ we must have $A(i) \subset J^\infty$; otherwise such a person would bid for an object not in J^∞ for sufficiently large t.

We now note that for sufficiently large t, the only bids taking place will be by persons in I^∞ bidding for objects in J^∞, so each object in J^∞ will be assigned to some person from I^∞, while at least one person in I^∞ will be unassigned (otherwise the algorithm would terminate). We conclude that the number of persons in I^∞ is larger than the number of objects in J^∞. This, together with the earlier shown fact $A(i) \subset J^\infty$, for all $i \in I^\infty$, implies that there is no complete assignment, contradicting our assumptions.

The optimality properties of the assignment obtained upon termination follow from the ε-CS property shown and our earlier discussion on the synchronous version of the algorithm. q.e.d.

A.4 REFERENCES

A.1 Balinski, M. L., "Signature Methods for the Assignment Problem", Operations Research J., Vol. 33, 1985, pp. 527-537.

A.2 Balinski. M. L., "A Competitive (Dual) Simplex Method for the Assignment Problem", Math. Programming, Vol. 34, 1986, pp. 125-141.

A.3 Barr, R., Glover, F., and Klingman, D., "The Alternating Basis Algorithm for Assignment Problems", Math. Programming, Vol. 13, 1977, pp.1-13.

A.4 Bertsekas, D. P., "A New Algorithm for the Assignment Problem", Math. Programming, Vol. 21, 1981, pp. 152-171.

A.5 Carpaneto, G., Martello, S., and Toth, P., "Algorithms and Codes for the Assignment Problem", Annals of Operations Research, Vol. 13, 1988,pp. 193-223.

A.6 Derigs, U., "The Shortest Augmenting Path Method for Solving Assignment Problems -- Motivation and Computational Experience", Annals of Operations Research, Vol. 4, 1985, pp. 57-102.

A.7 Engquist, M., "A Successive Shortest Path algorithm for the Assignment Problem", INFOR,Vol. 20, 1982, pp. 370-384.

A.8 Glover, F., R. Glover, and D. Klingman, "Threshold Assignment Algorithm", Center for Business Decision Analysis Report CBDA 107, Graduate School of Business, Univ. of Texas at Austin, Sept. 1982.

A.9 Goldberg, A. V., "Efficient Graph Algorithms for Sequential and Parallel Computers", Tech. Report TR-374, Laboratory for Computer Science, M.I.T., Feb. 1987.

A.10 Hall, M., Jr., "An Algorithm for Distinct Representatives", Amer. Math. Monthly, Vol. 51, 1956, pp. 716-717.

A.11 Hung, M., "A Polynomial Simplex Method for the Assignment Problem", Operations Research, Vol. 31, 1983, pp.595-600.

A.12 Jonker, R., and A. Volgenant, "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", Computing, Vol. 38, 1987, pp. 325-340.

A.13 McGinnis, L. F.,"Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem", Operations Research J., Vol. 31, 1983, pp. 277-291.

A.14 Munkres, J., "Algorithms for the Assignment and Transportation Problems", SIAM J., 1956.

A.15 Thompson, G. L., "A Recursive Method for Solving Assignment Problems", in Studies on Graphs and Discrete Programming, P. Hansen (ed),North-Holland Publ. Co., 1981, pp. 319-343.

A.16 Kuhn, H. W., "The Hungarian Method for the Assignment Problem", Naval Research Logistics Quarterly, Vol.2, 1955, pp. 83-97.

A.17 Bertsekas, D.P., "The Auction Algorithm: a Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, 1988.

A.18 Bertsekas, D. P., and J. N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J., 1989.

A.19 Papadimitriou, C. H., and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall,Englewood Cliffs, N. J., 1982.

A.20 Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem", Lab. for Information and Decision Systems Working Paper, M.I.T., March 1979.

A.21 Bertsekas, D. P., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems", LIDS Report P-1606, M.I.T., revision of Nov. 1986.

A.22 Bertsekas, D. P., and D. A. Castañon, "The Auction Algorithm for the Minimum Cost Network Flow Problem", May 1989, submitted for publication.

A.23 Bertsekas, D. P., and J. Eckstein, "Dual Coordinate Step Methods for Linear Network Flow Problems", Laboratory for Information and Decision Systems Report LIDS-P-1768, M.I.T., Cambridge, MA, 1988, also in Math. Progr.,Series B, Vol. 42, 1988, pp. 203-243.