Parallel Interval-Newton Using Message Passing: Dynamic Load Balancing Strategies

Chao-Yang Gau[*] and Mark A. Stadtherr[†]

Department of Chemical Engineering,
University of Notre Dame
182 Fitzpatrick Hall
Notre Dame, IN 46556 USA

Abstract

Branch-and-prune and branch-and-bound techniques are commonly used for intelligent search in finding all solutions, or the optimal solution, within a space of interest. The corresponding binary tree structure provides a natural parallelism allowing concurrent evaluation of subproblems using parallel computing technology. Of special interest here are techniques derived from interval analysis, in particular an interval-Newton/generalized-bisection procedure. In this context, we discuss issues of load balancing and work scheduling that arise in the implementation of parallel interval-Newton on a cluster of workstations using message passing, and describe and analyze techniques for this purpose. Results using an asynchronous diffusive load balancing strategy show that a consistently high efficiency can be achieved in solving nonlinear equations, providing excellent scalability, especially with the use of a two-dimensional torus virtual network. The effectiveness of the approach used, especially in connection with a novel stack management scheme, is also demonstrated in the consistent superlinear speedups observed in performing global optimization.

Keywords: Parallel Computing, Nonlinear Equations, Global Optimization, Branch-and-Prune, Branch-and-Bound, Interval Analysis

[*]Current address: LINDO Systems, Inc., 1415 North Dayton Street, Chicago, IL 60622, USA; E-mail: tgau@lindo.com
[†]E-mail: markst@nd.edu

1 Introduction

Process systems engineering involves the development and application of computer-aided, model-based techniques for the design, optimization, operation and control of chemical processing systems, as well as for the associated planning, scheduling and maintenance problems. In this field, two core numerical issues are the solution of nonlinear equation systems and of nonlinear programming (optimization) problems. However, neither of these problems can be solved with complete reliability using standard techniques. In this context, it has been shown (e.g., Balaji and Seader, 1995; Schnepper and Stadtherr, 1996; McKinnon et al., 1996; Byrne and Bogle, 2000; Han et al., 1997; Berner et al., 1999; Gau and Stadtherr, 2000; Stradi et al., 2001) that methods based on interval mathematics, such as the interval-Newton technique (e.g., Neumaier, 1990; Hansen, 1992; Kearfott, 1996), can be used to solve the nonlinear equation solving and optimization problems with complete reliability, providing a mathematical and computational *guarantee* that either *all* roots are found in the nonlinear equation solving problem, or that the *global* optimum is found in the nonlinear programming problem. Unfortunately, though not unexpectedly, this guarantee of reliability comes at the cost of a potentially high computational cost (CPU time). However, the interval-Newton method is implemented using branch-and-prune (BP) or branch-and-bound (BB) strategies that present a natural parallelism, and which therefore are particularly amenable to parallel processing. Thus, we seek to make effective use of high performance computing technology (parallel computing) to implement BP and BB strategies for reliably and efficiently solving problems in chemical process modeling and engineering.

Of particular interest here is the use of distributed parallel computing using a cluster of workstations, in which multiple workstations on a network are used as a single parallel computing resource by employing message passing. This sort of parallel computing system has advantages since it is relatively inexpensive, and is based on widely available hardware. Thus, such an approach to parallel computing has become a important trend in providing high performance computing resources in science and engineering. In this paper, we focus specifically on issues of load balancing and scheduling that arise in the implementation of parallel BB and BP using message passing on a workstation cluster, and describe and analyze techniques for this purpose, including a new work queue (stack) management scheme. Applications to problems arising in chemical process engineering are used to demonstrated the effectiveness of the approach used.

2 Background

Branch-and-prune and branch-and-bound algorithms are general-purpose intelligent search techniques for finding all solutions, or the optimal solution, within a search space of interest, and have a wide range of applications. These techniques employ successive decomposition (tessellation) of the global problem into smaller disjoint or independent subproblems that are solved recursively until all solutions, or the optimal solution, are found. BB and BP methods have many important applications in engineering and science, especially when a global solution to an optimization problem, or all solutions to a nonlinear equation solving problem are sought. In chemical engineering, these applications include process synthesis (e.g., Adjiman et al., 2000), process scheduling (e.g., Subrahmanyam et al., 1996), analysis of phase behavior (e.g., Xu et al, 2000), parameter estimation (e.g., Gau and Stadtherr, 2000), molecular modeling (e.g., Klepeis and Floudas, 2000), and many other applications of interest.

In BP, a subproblem is typically processed in some way to verify the existence of a feasible solution. The subproblem may be examined by a series of tests, and is pruned when it fails specified criteria or if a unique solution can be found inside this subdomain. If no conclusion is available, and so the subproblem cannot be pruned, the problem is bisected into two additional subproblems (nodes), generating a *binary tree* structure. One of the subproblems is then put in a stack (or work queue) and tests are continued on the other. This type of BP procedure underlies the application of the interval-Newton approach to equation-solving problems. In this case, the pruning scheme consists of a function range test and the interval-Newton existence and uniqueness test. There are three situations in which a node (here an interval or "box") can be pruned: (1) there is some component of the function range that does not contain zero; (2) a unique solution is proven to be enclosed, and (3) it is proven that no solutions exist. With these pruning criteria, a scheme can be constructed that searches the entire binary tree and finds (or, more precisely, encloses within narrow intervals) all solutions of the equation system, or determines that there are none.

In BB, the goal is typically to find a globally optimal solution to some problem. BB may be built on top of BP schemes by embedding an additional pruning test. In this test, a node is pruned when its optimal (lower bounding) solution is guaranteed to be worse (greater) than some known current best value (an upper bound on the global minimum). Thus, one avoids visiting subproblems which are known not to contain the globally optimal solution. In this context, various heuristic schemes may be of considerable importance in maintaining search efficiency. For example, when solving global minimization problems using interval analysis, the best upper bound value may be generated and updated by some heuristic combination of an interval extension of the objective function, a point objective function evaluation with interval arithmetic, and a local minimization with a verification by interval analysis. In order to enhance bounding and pruning efficiency, some approaches also apply a priority list scheme in BB. In this case, all problems in the stack are rearranged in the order of some importance index, such as a lower bound value. The idea is that the most important subproblems stored in the stack are examined with higher priority, in the hope that the global optimum be found early in the search process, thus allowing other later subproblems that do not possess the global optimum to be quickly pruned before they generate new nodes. We will present here a novel approach for management of the workload stack. This is a dual-stack scheme, in which the workload stack in each processor is split into a global stack for the global search and a local stack for the local search, thus providing the capability for balancing different search priorities in the parallel processing. Details concerning this new strategy and its experimental performance will be discussed below.

In BB or BP search, the shape and size of the search space typically changes as the search proceeds. Portions that contain a solution might be highly expanded with many nodes and branches, while portions that have no solutions might be discarded immediately, thus resulting in an irregularly structured search tree. It is only through actual program execution that it becomes apparent how much work is associated with individual subproblems and thus what the actual structure of the search tree is. Since the subproblems to be solved are independent, execution of both BP and BB on parallel computing systems can clearly provide improvements in computational efficiency; thus the use of parallel computing to implement BP and BB has attracted significant attention (e.g., McKeown et al., 1992; Rushmeier, 1993; Gendron and Crainic, 1994; Kumar et al., 1994; Correa and Ferreira, 1996; Mitra et al., 1997; Correa, 2000; Schnepper and Stadtherr, 1993, 1996; Epperly, 1995; Androulakis and Floudas, 1998; Berner et al., 1999; Smith and Pantelides, 1999; Sinha et al., 1999). However, because of the irregular structure of the binary tree, this implementation on distributed systems is often not straightforward. Details concerning the methodology for implementing BP and BB on distributed parallel systems will be discussed in later sections. Of particular interest here is to consider the effect of the virtual network (connectivity) used for message passing in reducing workload distribution irregularities.

In implementing an interval-Newton algorithm, there are opportunities for the use of parallel computing at multiple levels. On a fine-grained level, the basic interval arithmetic operations can be parallelized (e.g., Rump, 1999). On a larger-grained level, the solution of the linear interval equation system for the image can be parallelized (e.g., Frommer and Mayer, 1989; Gan et al., 1994; Hu et al., 1995; Hu, 1996). Then, as already noted, on a coarse-grained level, each independent subproblem (box) generated in the bisection process can be tested in parallel (e.g., Schnepper and Stadtherr, 1993, 1996; Berner et al., 1999; Sinha et al., 1999). It is only this coarsest level of parallelism that will be considered here.

3 Dynamic Load Balancing

As noted above, since the subproblems to be solved are independent, the execution of interval-Newton techniques, whether BP or BB, on distributed parallel systems can clearly provide improvements in computational efficiency. And since, for some practical problems, the binary tree that needs to be searched may be quite large, there may in fact be a strong motivation for trying to exploit the opportunity for parallel computing. However, because of the irregular structure of the binary tree, doing this may not be straightforward.

While executing a program to assign the unprocessed workload (stack boxes) to available processors, the irregularity of the tree could cause a highly uneven distribution of work among processors and result in poor utilization of computing resources. Newly generated boxes at some tree nodes, due to bisection, could cause some processors to become highly loaded while others, if processing tree nodes that can be pruned, could become idle or lightly loaded. In this context, we need an effective dynamic load balancing and work scheduling scheme to perform the parallel tree search efficiently. To manage the load balancing problem, one seeks to apply an optimal work scheduling strategy to transfer workload (boxes to be tested) automatically from heavily loaded processors to lightly loaded processors or processors approaching an idle state. The primary goal of dynamic load balancing algorithms is to schedule workload among processors during program execution, to prevent the appearance of idle processors, while minimizing interprocessor communication cost and thus maximizing the utilization of the computing resources.

A common load balancing strategy is the “manager-worker” scheme (e.g., Fraga and McKinnon, 1995; Androulakis et al, 1996; Schnepper and Stadtherr, 1996; Smith and Pantelides, 1999; Sinha et al., 1999), in which a single “manager” processor centrally conducts a group of “worker” processors to perform a task concurrently. This scheme has been popular in part because it is relatively easy to implement. It amounts to using a centralized pool to buffer workloads among processors. However, as the number of processors becomes large, such a centralized scheme could result in a significant communication overhead expense, as well as contention on the manager processor. As a result, in many cases, especially in the context of a tightly coupled cluster, this scheme does not exhibit particularly good scalability. Thus, to avoid bottlenecks and high communication overhead, we concentrate here on decentralized schemes (without a global stack manager), and consider three types of load balancing algorithms specifically designed for network-based parallel computing using message passing. It should be noted, however, that for loosely coupled systems, the manager-worker scheme can be quite effective, as demonstrated, for example, in the metaNEOS project (Anstriecher et al, 2000; Goux, et al., 2000).

The parallel algorithms considered adopt a distributed strategy that allows each processor to locally make workload placement decisions. This strategy helps a processor maintain for itself a moderate local work queue (stack), hopefully preventing itself from becoming idle, and alleviates bottleneck effects when applied on large-scale multicomputers. Distributed parallel algorithms of this type are basically composed of five phases: workload measurement, state information exchange, transfer initiation, workload placement, and global termination. Each of these phases is now discussed in more detail.

3.1 Workload Measurement

As the first stage in a dynamic load balancing operation, workload measurement involves evaluation of the current local workload using some “work index”. This is a criterion that needs to be calculated frequently, and so it must be inexpensive to determine. It also needs to be sufficiently precise for purposes of making good workload placement decisions later. In the context of interval BP and BB, a good approach is to simply use the stack length (number of boxes in the work queue) as the work index. This index is effective since: 1. A long stack suggests a heavy workload in a local processor, and vise versa; 2. Exhibiting an empty stack indicates the local processor is approaching an idle state; 3. A more precise representation of workload by work index may not be needed, since it may not be necessary to maintain an equal workload on all processors, but merely to prevent the appearance of idle states. Thus, the stack length can serve as a simple, yet effective, workload index. Note that this workload measurement scheme is not intended to count how much total workload is ultimately associated with the boxes in the work queue, but just to give a temporary (transient) measurement of the workload stored at each processor. This then gives the load-balancing scheme the information needed to make proper work transmission decision.

3.2 State Information Exchange

After all processors identify their own workload state, the parallel algorithm makes this local information available to all other cooperating processors, through interprocessor message passing, to construct a global work index vector. The cooperating processors are a group of processors participating in load balancing operations with a local processor, and define the domain of interprocessor communication, thereby determining a *virtual network* for cooperation. The range of this domain is critical in determining the cost of communication and the performance of load balancing. One possibility is that the cooperating processors could include all processors available on the network, and a global all-to-all communication scheme could then used to update global state information. This provides a very up-to-date global work index vector but might come at the expense of high communication overhead. Alternatively, the cooperating processors might include only a small subset of the available processors, with this small subset defining a local processor’s nearest “neighbors” in the virtual network. Now one needs only to employ cheap local point-to-point communication operations. However, without a good load balancing algorithm, these local schemes could more readily result in workload imbalance and idle states.

3.3 Transfer Initiation

After obtaining an overview of the workload state, at least for the group of cooperating (“neighboring”) processors, load balancing algorithms now need to decide if a workload placement is necessary to maintain balance and prevent an idle state. This is done according to an initiation policy that dictates under what conditions a workload (box) transfer is initiated, and decides which processors will trigger the load balancing operation. Generally, the migration of boxes from one processor to another processor can be initiated either periodically or on demand. When triggered on demand, the load balancing operations are event driven according to different procedures, such as a sender-initiate scheme (e.g., Vornberger, 1987; Troya and Ortega, 1989; Quinn, 1990), a receiver-initiate scheme (e.g., Vornberger, 1986; Finkel and Manber, 1987; Rao and Kumar, 1987) and a symmetric scheme (e.g., Luling and Monien, 1989; Clausen and Traff, 1991; Epperly, 1995). In the sender-initiate scheme, when the workload of any processor is too heavy and exceeds an upper threshold, the overloaded processor will offload some of its stack boxes to another processor through the network. The receiver-initiate approach works in the opposite way by having an underloaded processor request boxes from heavily loaded processors, when the underloaded processor's workload is less than a lower threshold. The symmetric scheme combines the previous two strategies and allows both underloaded and overloaded processors to initiate load balancing operations.

3.4 Workload Placement

The next step in a dynamic load balancing algorithm is to complete a workload placement. Here the donor processor splits the local work queue (stack) into two parts, sending one part to the requesting processor and retaining the other. This operation is done according to a transfer policy consisting of two rules: a work-adjusting rule and a work-selection rule. The work-adjusting rule determines how to distribute workload among processors and how many stack boxes are to be transferred. If the requesting processor receives too little work, it may quickly become idle; if the donor processor offloads too much work, it itself could also become idle. In either case, the result would eventually intensify the communication needed to perform later load balancing operations. Many approaches are available for this rule. One simple approach is to transfer a constant number of work units (boxes) upon receiving a request, such as in a *work stealing* strategy (e.g., Blumofe and Leiserson, 1994, 1999). A more sophisticated approach is to adopt a *diffusive propagation* strategy (e.g., Cybenko, 1989; Heirich and Taylor, 1995, Heirich and Arvo, 1998), which takes into account the workload states on both sides and adjusts the workload dynamically with a mechanism analogous to heat or mass diffusion.

In addition to the quantity of workload, as measured by the work index, the "quality" of transferred boxes is also an important issue. In this context, a work-selection rule is applied to select the most suitable boxes to transmit in order to supply adequate work to the requesting processor, and thus reduce the demands for further load balancing operations later. Although it is difficult to precisely estimate the size of the tree (or total work) rooted at an unexamined node (box), many heuristic rules have been proposed to select the appropriate boxes. One rule-of-thumb is to transmit boxes near the initial root of the overall binary tree, because these boxes tend to have more future work associated with the subsequent tree rooted at them (e.g., Foster, 1995). While this has been demonstrated to be a good selection rule in many tree search applications, this and other such selection rules will not necessarily have a strong influence on the performance of a parallel BP algorithm applied to solve equation-solving problems using interval analysis. However, the selection rule used can have a strong impact on a parallel BB algorithm when solving global minimization problems, since, by affecting the evaluation sequence of boxes, it in turn affects the time at which good upper bounds on the global minimum are identified. In general, the earlier a good upper bound on the global minimum can be found, the less work that needs to be done to complete the global minimization, since this means it is more likely that boxes can be pruned using an objective range test. This issue will be addressed in more detail below.

3.5 Global Termination

Parallel computation will be terminated when the globally optimal solution for BB problems, or all feasible solutions for BP problems, have been found over the entire binary tree, making all processors idle. For a synchronous parallel algorithm, global termination can be easily detected through global communication or periodic state information exchange. However, detecting the global termination stage is a more difficult task for an asynchronous distributed algorithm, not only because of the lack of global or centralized control, but also because there is a need to guarantee that upon termination no unexamined workload remains in the communication network due to message passing. One commonly used approach that provides a reliable and robust solution to this problem is Dijkstra's token termination detection algorithm (Dijkstra and Scholten, 1980; Dijkstra et al., 1983; Kumar et al., 1994).

4 Implementation of Dynamic Load Balancing Algorithms

In this section, a sequence of three algorithms (SWS, SDLB and ADLB) is described for load balancing in a binary tree, with each algorithm in the sequence representing an improvement in principle over the previous one. The last method (ADLB) represents a synthesis of the most attractive and effective strategies adapted from previous research studies, and also incorporates some novel strategies in this context. Though ADLB is expected to be superior, we describe all three approaches, as this provides a basis for comparison, as well as a context within which to better understand the ADLB approach. Interprocessor communication is performed using the MPI protocol (Gropp et al., 1994, 1999), a very powerful and popular technique for message passing operations that provides various communication functions as discussed below. The performance of the three algorithms described will be compared using both a nonlinear equation solving test problem and a global optimization test problem.

4.1 Synchronous Work Stealing (SWS)

This first workload balancing algorithm applies a global strategy, and is illustrated in Figure 1. All processors are synchronized in the interleaving computation and communication phases. Synchronous blocking all-to-all communication is used to periodically (after some number of tests on boxes) update the global workload state information. Then, every idle processor, if there are any, “steals” one unit of workload (one box) from the processor with the heaviest workload (the largest number of stack boxes), applying a receiver-initiate scheme. As the responsibility for the workload placement decision is given to each individual processor, rather than in a centrally controlling manager processor, but global communication is maintained, SWS can be regarded as a type of *distributed* manager/worker scheme.

The global, all-to-all communication used in this approach provides for an easy determination of workload dynamics, and may lead to a good global load balancing. However, like the centralized manager/worker scheme, this convenience also comes at the expense of increased communication cost when using many processors. Such costs may result in intolerable communication overhead and degradation of overall performance (speedup). It should also be noted that the synchronous and blocking properties of the communication scheme may cause idle states in addition to those that might arise due to an out-of-work condition. When using the synchronous scheme, a processor (sender) that has reached the synchronization point and is ready for communication needs to stay idle and wait for another processor (receiver) to reach the same status, and then initiate the communication together. Additional waiting states may occur due to the use of blocking communication, since a message-passing operation may not complete and return control to the sending processor until the data has been moved to the receiving processor and a receive posted. Thus, the main difficulties with the SWS approach are the communication overhead and the likely occurrence of idle states, which together may result in poor scalability. However, one advantage to this approach is that the global communication makes it easy to detect global termination.

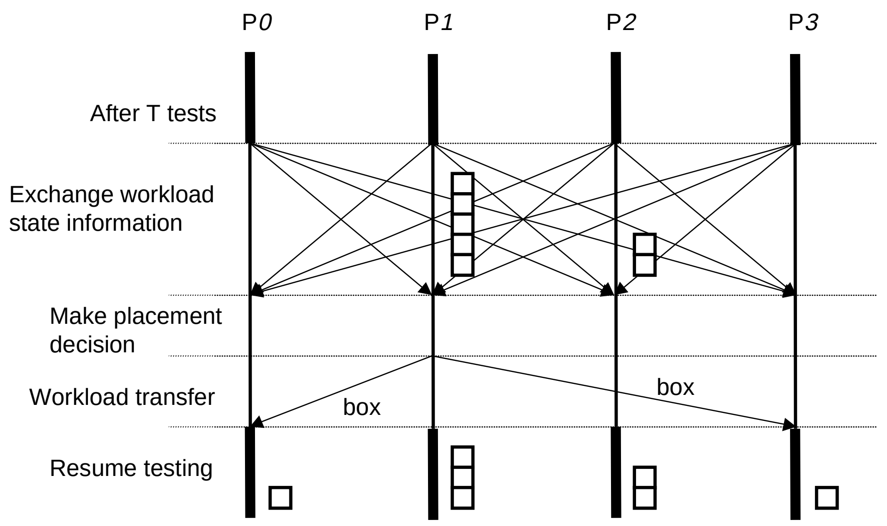

Figure 1: The SWS algorithm uses global all-to-all communication to synchronize computation and communication phases.

4.2 Synchronous Diffusive Load Balancing (SDLB)

This second approach for workload balancing follows a localized strategy, by using local, point-to-point communication and a local cooperation strategy in which load balancing operations are limited to a local domain of cooperating processors, i.e., a group of "nearest neighbors" on some predefined *virtual* network. A diffusive work-adjusting rule is also applied here to dynamically coordinate workload transmission between processors, thereby achieving a workload balance with a mechanism analogous to heat or mass diffusion, as illustrated in Figure 2.

Instead of using global communication, point-to-point synchronous blocking communication is used to exchange workload state information among cooperating (neighbor) processors. The gathered information allows a given processor to construct its own work index vector indicating the workload distribution in its neighborhood. Then, the algorithm uses a symmetric initiation scheme to cause the workload (boxes) to "diffuse" from processors with relatively heavy workloads to processors with relatively light workloads, in order to maintain a roughly equivalent workload over all processors. The virtual network used initially here is simply a ring, which gives each processor two nearest neighbors. Each local processor, i, adjusts its local workload with a neighbor, j, according to the rule

$$u(j) = C[W(i) - W(j)],$$

where $u(j)$ is the workload-adjusting index, C is a "diffusion coefficient" and W is the work index vector. If $u(j)$ is positive and/or greater than a threshold, the local processor sends out workload (boxes); if $u(j)$ is negative and/or less than a threshold, the local processor requests workload (boxes). The diffusion coefficient, C, is a heuristic parameter determining what fraction of local work to offload, and is set at 0.5 in our applications. This diffusive scheme has two advantages. First, when applied at an appropriate frequency, it provides some certainty in preventing the appearance of out-of-work idle states. Also, compacting multiple units of workload (boxes) together for transmission enlarges the *virtual grain* of the transmitted messages. The use of coarse-grained messages to reduce communication frequency tends to minimize the effect of high latency in network transmission, especially on Ethernet. For example, less total time is wasted in startup time of transmission, thus lowering the average transmission cost of a work unit (box), as well as the ratio

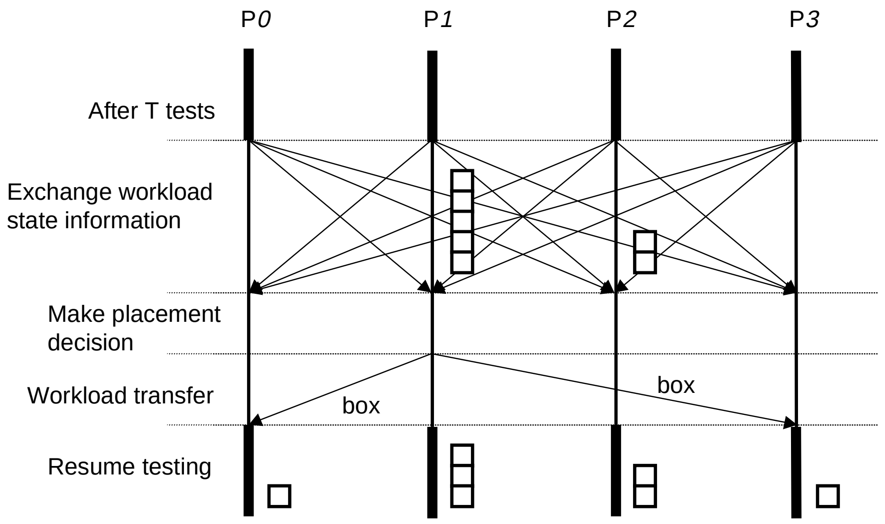

Figure 2: SDLB uses a diffusive work-adjusting scheme to share workload among neighbors in the virtual network. It is synchronous like SWS.

of communication time to computation time. It should be noted that in considering message grain there may also be maximum message size considerations.

Though the use of a local communication scheme will reduce communication costs to some extent, the use again of synchronous and blocking communication operations are still difficulties in achieving good scalability. On the other hand, while using local rather than global communication makes the detection of global termination less efficient, the synchronous and blocking properties make this relatively straightforward. Since the problem of detecting global termination becomes more difficult as the number of processors grows, this is another important issue in scalability.

4.3 Asynchronous Diffusive Load Balancing (ADLB)

In this third load balancing approach, a local communication strategy and diffusive work-adjusting scheme are used, as in SDLB. However, a major difference here is the use of an asynchronous nonblocking communication scheme, one of the key capabilities of MPI. The combination of asynchronous communication functionality and nonblocking, persistent communication functionality not only provides for cheaper communication operations by eliminating communication idle states, but also, by breaking process synchronization, makes the sequence of events in the load balancing scheme flexible by allowing *overlap* of communication and computation. As illustrated in Figure 3, when each processor can perform communication arbitrarily at any time, and independently of a cooperating processor, all communication operations can be scattered among computation, with less time consumed in message passing.

In addition to the cheaper and more flexible communication scheme, we incorporate into the ADLB approach two new strategies to try to reduce the demand for communication and thereby try to achieve a higher overall performance. First, as noted above, in BP and BB methods, it is not really necessary to maintain a completely balanced workload across processors. The actual goal is to prevent the occurrence of idle states by simply maintaining a workload for each processor sufficiently large to keep it busy with computation. To achieve balanced workloads may require a very large number of workload transmissions,

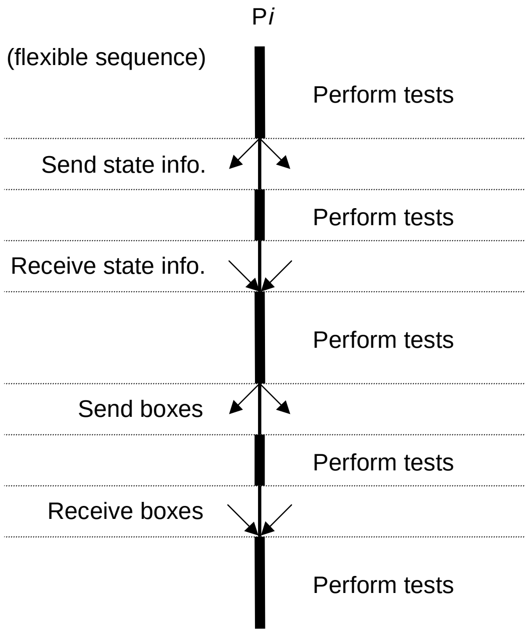

Figure 3: ADLB uses an asynchronous, nonblocking communication scheme, providing more flexibility to each processor and overlapping communication and computation phases.

resulting in a heavy communication burden. However, in this case, many of the workload transmissions may be unnecessary, since in BP and BB each processor deals with its stack one work unit (box) at a time sequentially, leaving all other workload simply standing by. For a processor to avoid an idle state, and thus have a high efficiency in computation, it is not necessary that its workload be balanced with other processors, but only that it be able to obtain additional workload from another processor through communication as it is approaching an out-of-work state. Thus, we use here a receiver-initiate scheme to initiate work transfer only when the number of boxes in a processor's stack is lower than some threshold, which should be set high enough that the processor is not likely to complete the work and become idle during the processing of the workload request to its neighboring processors.

As a consequence, we can also implement a second strategy, which eliminates the periodic state information exchange and combines the load state information of the requesting processor with the workload request message to the donor processor. Upon receiving the request, the donor follows a diffusive work-adjusting scheme as described above for the SDLB approach, but with a modification in the response to the workload adjusting index. Here, if $u(j)$ is positive and/or greater than a threshold, the donor sends out workload (boxes) to the requesting processor; otherwise, it responds that there is no extra workload available. Thus, when approaching idle, a processor sends out a request for work to all its cooperating neighbors, and waits for any processor's donation of work. In case of no work being transferred, it means that the neighbor processors are also starved for work and are making work requests to other neighbors. In this case, the processor will keep requesting work from the same neighbors until they eventually obtain extra work from remote processors and are able to donate parts of it. Through such a diffusive mechanism, heavily loaded processors can propagate workload to lightly loaded processors with a small communication expense. A variation on this approach, which we have not tested, would be to have each neighbor processor update its state to record the demanding processor's request for more work, thus preventing the need for repetitive identical calls for work from the same processor and reducing communication overhead even further. Then, when the demanding processor no longer needs work, it can notify neighbor processors again to modify their states accordingly.

In the solution of global minimization problems by BB, it is also necessary to communicate information about the upper bound on the global minimum. This is done in two ways. First, whenever a processor conducts any kind of send operation (e.g., to send workload status, to send workload, etc.), its current bound value is packed with the communication. Second, whenever the processor obtains a better (smaller) bound, whether by local computation or by communication from another processor, that new bound is sent to neighboring processors. In this way, bound values are propagated across the entire cluster so that all processors quickly receive the current best bound value.

The last step of the load balancing procedure is to detect global termination. Because the ADLB scheme is asynchronous, the detection of global termination is a more complex issue than in the synchronous case. As noted above, a popular and effective technique for dealing with this issue is Dijkstra's token algorithm; this is the technique used in the ADLB scheme.

In the next section, we describe tests of the three approaches outlined above for load balancing in parallel BP and BB.

4.4 Computational Experiments and Results

4.4.1 Test Environment

The performance of an algorithm on a parallel computing system is not only dependent on the problem characteristics and the number of processors but also on how processors interact with each other, as determined both by a physical architecture in hardware and a virtual architecture in software. The physical architecture used in the tests described in this section is a network-based system, comprising 16 Sun Ultra 1/140e workstations, physically connected by switched 100 MBit Ethernet. (Note that this architecture is used to carry out only the tests described in this section, namely the comparison of the three load balancing schemes discussed above; in the later sections, involving tests on the effects of virtual network and stack management, another experimental platform with 32 Sun Ultra 2/2200 machines is used instead.) As noted above, in comparison to mainframe systems, such a cluster of workstations (COW) has advantages in its relatively low expense and easy availability of hardware. However, depending on the communication bandwidth and on the communication demands of the algorithm being executed, network contention can have a serious impact on the performance of such a system, particularly if the number of processors is large.

Two types of virtual network are used: an all-to-all network in the case of SWS, and a one-dimensional torus (ring) network in the cases of SDLB and ADLB. In the SWS algorithm, the all-to-all network is implemented by the use of global, all-to-all communication. However, in the SDLB and ADLB algorithms, in order to reduce communication demands and alleviate potential network contention, we only use point-to-point local communication functions and implement the ring network. The load balancing algorithms and test problems were implemented in FORTRAN-77 using the MPI protocol for interprocessor communication. In particular, we used the popular LAM (Local Area Multicomputer) implementation (version 6.4) of MPI, developed and distributed by the Laboratory for Scientific Computing at the University of Notre Dame (http://www.lsc.nd.edu or http://www.lam-mpi.org).

4.4.2 Test Problems

The test problems used are based on a global nonlinear parameter estimation problem involving a vapor-liquid equilibrium (VLE) model (Wilson's equation). Such models, and the estimation of parameters in them, are important in chemical process engineering, since they are the basis for the design, simulation and optimization of widely-used separation processes such as distillation. In this particular problem, we use as the objective function the maximum likelihood estimator, with two unknown standard deviations, to determine two model parameters giving the globally optimal fit of the data to the model (Gau and Stadtherr, 1998). In addition to the difficult nonlinear objective function, the problem data and characteristics were chosen to make this a particularly difficult problem, requiring a few hours of computation time on a single processor. Interval analysis, as described above, is used to guarantee the correct global solution. The problem can be solved in either of two ways. One approach is to treat it as a nonlinear equation solving problem, and use the parallel interval BP algorithm to solve for all stationary points of the objective function (there are five stationary points in this problem). The alternative approach is to treat it directly as a global optimization problem and use the parallel interval BB algorithm. The major difference between the two approaches is the use of the objective range test in the BB algorithm.

4.4.3 Computational Results

This parameter estimation problem was solved using the cluster of workstations described above. During the computational experiments, the cluster was dedicated exclusively to solving this problem; that is, there were no other users either on the workstations or on the network. Both the BP scheme solving for all stationary points and the BB scheme merely searching for the global optimum were executed on up to 16 processors using each of the three load balancing schemes described above. Both sequential and parallel execution times were measured in terms of the MPI wall time function, and the performance of each approach evaluated in terms of parallel speedup (ratio of the sequential execution time to the parallel execution time) and parallel efficiency (ratio of the parallel speedup to the number of processors used).

For the interval BP problem of finding all stationary points, the speedups obtained using the three load balancing algorithms, i.e. SWS, SDLB and ADLB, on various number of processors are shown in Figure 4. All five stationary points were found in every experiment. All points in Figure 4 are based on an average over several runs. Since both the sequential runs and all parallel BP runs explored the same binary tree and treated an equivalent amount of total work, the computational results are repeatable and consistent with negligible deviations. As expected, the ADLB approach clearly outperforms SWS and SDLB, exhibiting only slightly sublinear speedup. This can also be seen in the parallel efficiency curves, as shown in Figure 5. While efficiency curves tend to decrease as the number of processors increases, as a consequence of Amdahl's law, the ADLB procedure maintains a high efficiency of around 95%. Thus, with the only slightly sublinear speedup and the very high efficiency on up to 16 processors, it seems likely that the ADLB algorithm will be highly scalable to larger numbers of processors.

SWS exhibits the poorest performance of the three load balancing methods. This is partly due to a poor global workload distribution, resulting in a relatively large number of out-of-work idle states, and also partly due to the communication overhead from using the global synchronous blocking communication scheme. In SDLB, the symmetric diffusive work-adjusting scheme using the local communication scheme substantially reduces out-of-work idle states by achieving an even load balance and thus improving the speedup and efficiency. However, while a local communication scheme is employed, the synchronous blocking communication functions used retain a high communication cost and represent a scaling bottleneck. This issue is

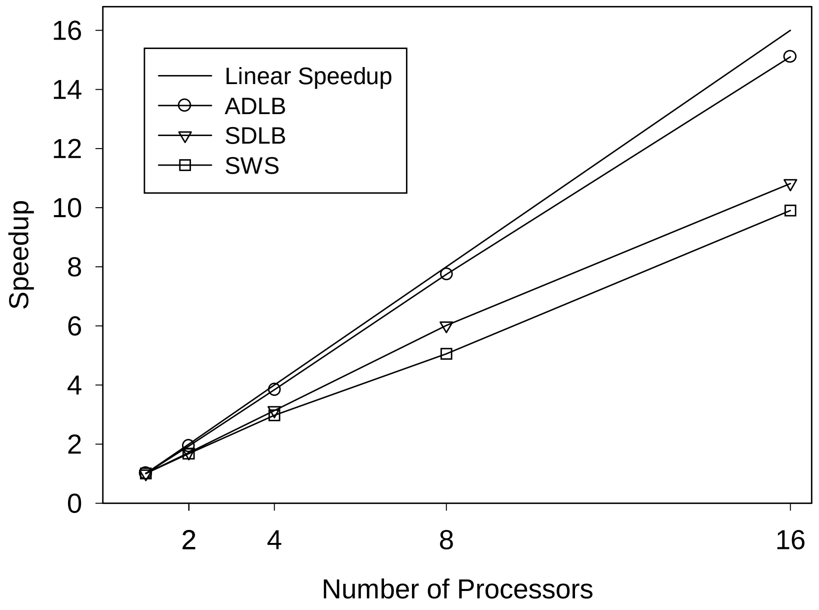

Figure 4: Comparison of load balancing algorithms on equation solving problem: Speedup vs. number of processors.

addressed in ADLB by using asynchronous nonblocking communication functions, allowing the overlap of communication and computation. In addition, by working towards a goal of maintaining non-empty local work stacks instead of an evenly balanced global workload distribution, ADLB provides a large reduction in network communication requirements, thus greatly reducing communication bottlenecks. The reduction of such bottlenecks in ADLB allows it to achieve a consistently high, nearly linear speedup.

For solving the parameter estimation problem as a global optimization problem with parallel interval BB, only the best load balancing scheme, ADLB, was employed. Multiple runs solving the same problem were made at two, four, eight and 16 processors. The resulting speedups are shown in Figure 6. We first observe that all speedups are above the linear speedup line, with a speedup over 50 on 16 processors in one case. Superlinear speedup is possible because of the broadcast of least upper bounds, which may cause tree nodes (boxes) to be discarded earlier than in the sequential case, i.e. there is less work to do in the parallel case than in the sequential case. Also, the speedups are not exactly repeatable and may vary significantly from run to run. This occurs because of slightly different timing in finding and broadcasting improved upper bounds in each run. Speedup anomalies, such as the superlinear speedups seen here, are not uncommon in parallel BB search, provided the reduction in the work required in the parallel case (which often happens but not always) is not outweighed by communication expenses or other overhead in the parallel computation.

The excellent performance of ADLB on the tests described above provides motivation for further improving the ADLB approach for execution on even larger numbers of processors and applied to different sizes of problems. Along these lines, one factor we have investigated is the effect of the underlying virtual network. Another factor of interest is the stack management scheme used to prioritize work. Both of these factors are considered in the sections below, along with additional test problems.

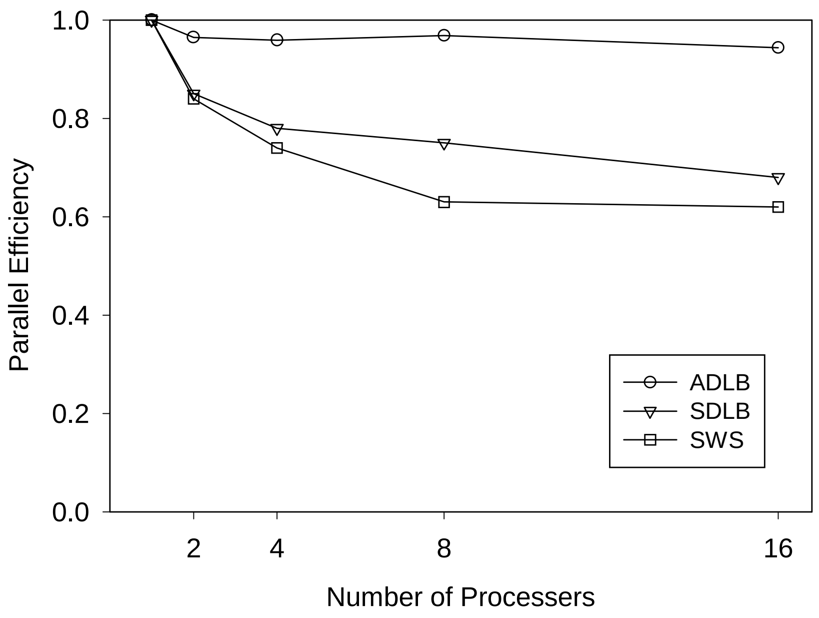

Figure 5: Comparison of load balancing algorithms on equation solving Problem: Efficiency vs. number of processors.

5 Effect of Virtual Network

The virtual network used in the local coordination of neighbor processors for workload distribution and message propagation has an important effect on network communication overhead. Since more connections mean higher communication costs, the number of neighbors on the virtual network affects the message propagation speed across the set of processors. On one hand, having more connections among processors can reduce the message diffusion distance and help balance overloaded or underloaded processors. However, on the other hand, increasing the amount of communication on the network may degrade its performance, delaying message transmissions and reducing load balancing performance. Thus, an appropriate virtual network is necessary. In fact, the optimal virtual network depends on various factors, including the hardware environment, the numerical algorithm being applied, and the size and difficulty of the problems being solved.

Here, with the goal of improving the performance of the ADLB approach, we consider a two-dimensional torus (or 2-D wraparound mesh) virtual network, and compare its performance to that of the 1-D torus used for the previous test problems. Relative to a 1-D torus, the 2-D torus has a higher communication overhead due to more neighbors, but results in a smaller network diameter, $\lceil\sqrt(P)/2\rceil$ vs. $\lfloor P/2\rfloor$, that decreases message diffusion distance. It is expected that the trade-off between communication overhead and message diffusion distance may favor the 2-D torus for larger numbers of processors.

To compare the performance of ADLB on the 1-D and 2-D torus virtual networks, some reasonable and systematic measurement is needed. In the performance analysis done above, the parallel speedup and efficiency were used as simple performance measures, as is very commonly done. However, it is also well known that both these measures tend to depend on the size of the problem being solved, with higher speedups and efficiencies seen on larger problems. To consider the effect of problem size in the comparison here, we will use an *isoefficiency* analysis (Kumar et al., 1994). The isoefficiency function describes how much problem size must be increased as a function of the number of processors used in order to maintain

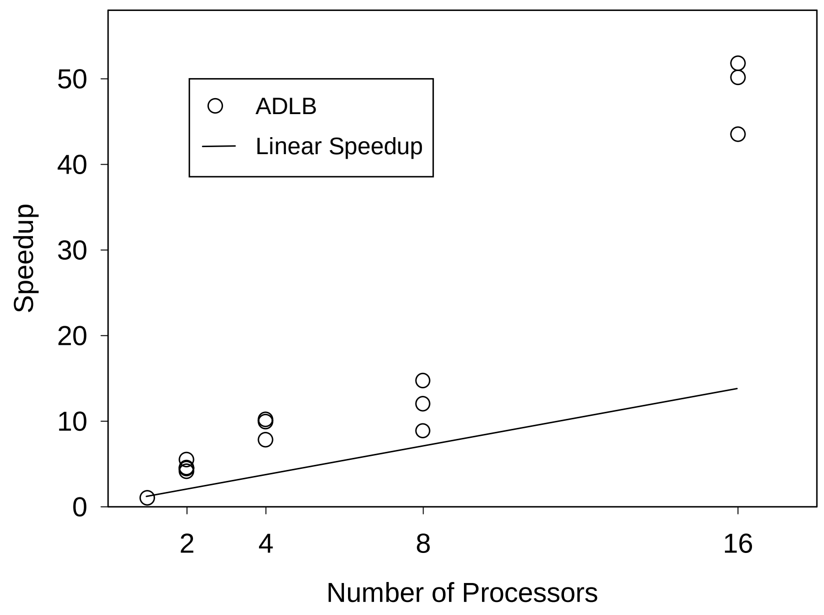

Figure 6: Superlinear speedups are observed in solving the global optimization problem using the parallel BB algorithm based on ADLB.

a constant parallel efficiency. For a given machine architecture or virtual network, this function can be determined experimentally by running the parallel algorithm with different numbers of processors over a wide range of problem sizes. The smaller the increase in the isoefficiency function with the number of processors, the better the scalability.

In choosing a test problem for the isoefficiency analysis, one first must be sure that the amount of work required to solve the problem will not change with number of processors. Thus, it is not appropriate to use a global optimization problem solved using BB, since the amount of work needed will depend on the timing with which improved bounds are found, which in turn depends on the number of processors. Thus, the test problem used is a nonlinear equation solving problem, solved using the interval-Newton BP approach described above. It is also desirable to design the test problem so that the problem size can be easily and precisely varied. One conceivable approach for doing that in this case would be to vary the size of the initial interval used. However, this is not a precise way to vary the problem size, since, for instance, doubling the size of the initial interval will not necessarily double the amount of work that is needed to solve the problem. A more precise way to vary the problem size is to give the algorithm multiple and independent instances of the same initial interval. If the number of instances of the the initial interval is doubled, then this precisely doubles the amount of work required.

The test problem chosen involves the computation of mixture critical points from cubic equation-of-state models. Stradi et al. (2001) have recently described how this can be done reliably using an interval-Newton methodology to solve the criticality conditions. The specific problem used here is to compute the critical point of a mixture of methane, carbon dioxide, and hydrogen sulfide using the Peng-Robinson equation-of-state. Details of the problem formulation, as well as model parameters and initial intervals for all variables, are given by Stradi et al. (2001). To solve this problem on a single processor requires 48.5 CPU seconds. This represents one "unit" of problem size, with multiple units of problem size created by using multiple instances of the same initial interval, as discussed above. The physical hardware used is a cluster of Sun Ultra 2/2200 workstations connected using switched 100 MBit Ethernet. These machines each have two

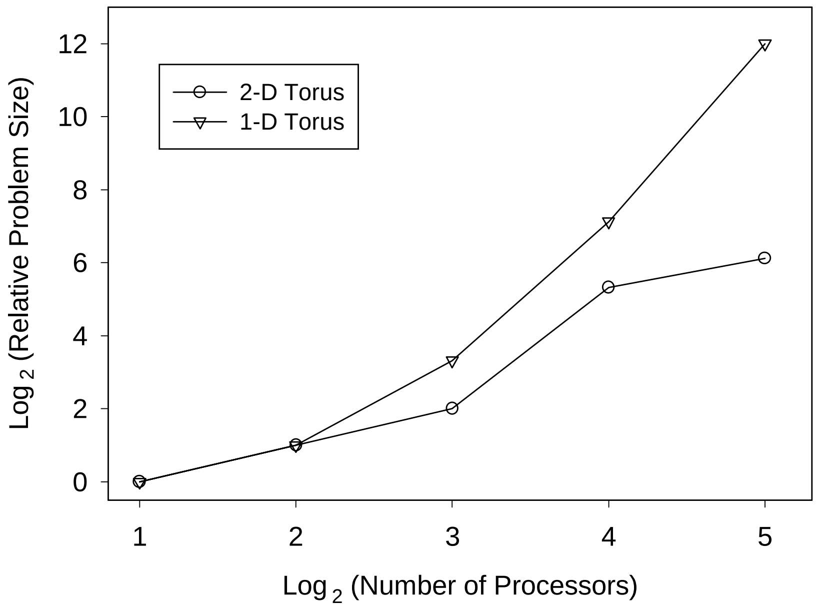

Figure 7: Isoefficiency curves for 1-D and 2-D virtual networks using ADLB for an equation solving problem.

processors (200 MHz); however, each processor used in this study was from a different machine, thus providing a homogeneous cluster.

Isoefficiency curves were determined experimentally by making runs with two, four, eight, 16 and 32 processors, increasing the problem size in each case to maintain a parallel efficiency of 92%. Results are shown in Figure 7. The lower rate of increase for the 2-D torus curve, especially for larger number of processors, indicates the superior scalability of the 2-D torus virtual network for this type of problem. For instance, it can be seen that for 32 processors the problem size required to achieve a 92% parallel efficiency in the 1-D torus virtual network is approximately 64 (2^6) times the problem size needed to achieve this level of efficiency using ADLB in the 2-D torus virtual network. ADLB can also easily be implemented for other types of virtual networks, for instance a hypercube or a higher dimension torus. Some extensive experiments using this test problem were also done on a hypercube virtual network, with the resulting isoefficiency curve being very similar to that of the 2-D torus. The selection of an optimal virtual network is dependent on the hardware and software environment and in practice would be determined experimentally using scalability analysis of the type demonstrated here (Kumar et al., 1994).

6 Stack Management

Another issue of interest is how to improve the parallel search efficiency of interval BB for global optimization problems. As noted above, when a parallel BB approach is applied to solve global minimization problems, the amount of work required will typically differ depending on the number of processors used. This is because the number of processors used affects the timing which with best (least) upper bounds on the global minimum are found and propagated across the network to the entire cluster. In fact, because the propagation of bounds on the network is not a deterministic operation, the amount of work required may vary significantly from run to run in solving the same problem on the same number of processors, as observed in Figure 6. It is possible that the amount of work needed in the parallel case will be greater than that in the serial case. This is referred to as a deceleration anomaly and will result in sublinear speedups. On the other hand, it is also possible that the work needed in the parallel case may be less than that in the serial case. This is referred to as an acceleration anomaly, and may result in a superlinear speedup, provided that the decrease in the amount of work is not offset by communication overhead and poor load balancing performance.

Clearly, from either a serial or parallel standpoint, it is desirable to find a tight upper bound on the global minimum, or the global minimum itself, as early as possible in the search process. Towards this goal, various stack management schemes, or work selection rules, have been proposed (e.g., Kanal and Kumar, 1988; Teng, 1990; Pardalos and Rodgers, 1990; McKeown et al., 1992; Kumar et al., 1994; Correa and Ferreira, 1996; Correa, 2000). Generally these prioritize the stack in some way so that the work (tree nodes) that are considered most promising from some standpoint are selected for early processing. Such best-first schemes then replace or supplement the usual depth-first search strategy. For example, Correa (2000) and Correa and Ferreira (1996) use a heuristic function to generate a priority list based on objective function lower bound values. In a variety of contexts, prioritization schemes such as these have been demonstrated to be useful. However, in our experience, at least in the context of interval-based BB, best-first strategies have performed inconsistently. The difficulty with using lower bound values is that these may not be sufficiently tight to provide any useful heuristic ordering for the evaluation of stack boxes. This is particularly true if the lower bound is obtained by simple interval arithmetic, which often provides only loose bounds when applied to a complicated function. As an alternative approach to obtain more consistent performance, we have developed a novel, dual-stack management scheme.

In this dual-stack strategy, each processor maintains two stacks (work queues), a global stack and a local stack. The local stack is unprioritized; that is, with workload appearing in the same sequence as it is generated in the IN/GB algorithm. The local processor draws its work from the local stack as long as it is not empty. This contributes a depth-first pattern to the overall tree search process. If a new least upper bound is found locally by a processor in a feasible node or box, then this box and its contiguous boxes are locked into the local stack to ensure immediate further exploration of this potentially promising region. The global stack is also unprioritized, and is created by randomly removing unlocked boxes from the local stack. The global stack provides boxes for workload transmission to other processors. This contributes breadth to the tree search process. By dynamically incorporating both depth and breadth into the stack management scheme, we are able to carry out a consistently well-balanced tree search process. However, it should be noted that the dual-stack strategy is a general one, and is not tied necessarily to using depth-first search for the local stack. In general, the local stack can be managed using whatever approach (e.g. depth-first, best-first) is considered most appropriate for sequential local search for the problem at hand, while the global stack will continue to provide another mechanism for box circulation among processors. Thus, the dual-stack approach should also be effective if strategies other than depth-first search are used for the local stack.

To test the dual-stack management strategy in comparison to the standard single-stack (depth-first) scheme, we use a global optimization problem that arises in a parameter estimation problem in which the error-in-variables approach is used. The specific problem is described by Gau and Stadtherr (2000), and involves estimation of parameters in the Van Laar equation for activity coefficient for a mixture of methanol and 1,2-dichloroethane. Details of the problem formulation, together with data and initial intervals used are given by Gau and Stadtherr (2000). The single- and dual-stack cases were compared by measuring speedups on up to 32 processors. The ADLB approach with 2-D torus virtual network was used, and the physical hardware used is again a cluster of Sun Ultra 2/2200 machines connected using switched 100 MBit Ethernet, with each processor from a different machine. Since, as explained previously, execution times can be expected to vary from run to run on the same number of processors, a range of speedup values was

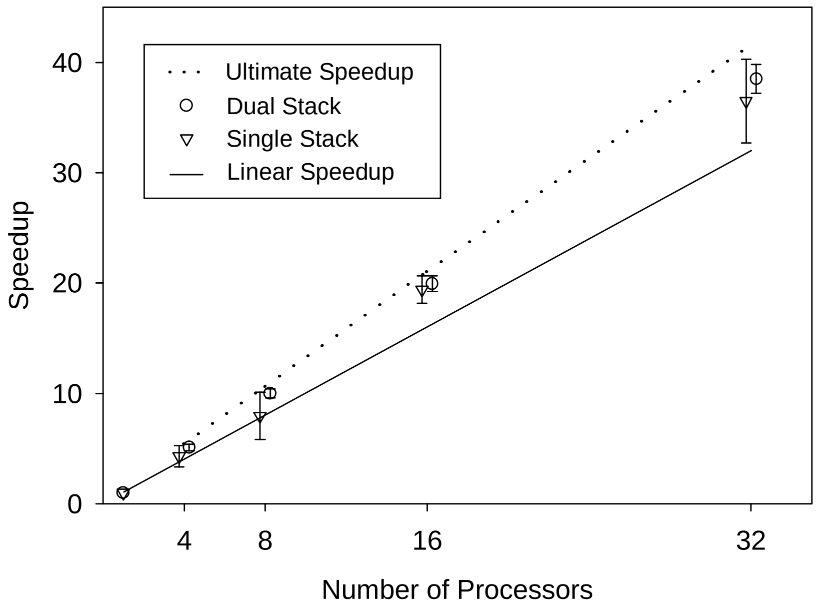

Figure 8: Comparison of stack management schemes in ADLB on a global optimization problem. See text for discussion of ultimate speedup. For clarity, the data symbols are slightly offset horizontally for each value of number of processors.

determined based on 20 runs for each case.

Results are shown in Figure 8, which shows the mean speedup for each case as well as the range of observed values. Also shown here are "ultimate" speedup values. This refers to the case in parallel execution in which all processors are initially given the global minimum as the best upper bound. Obviously these results represent an idealized case, since the global minimum will not normally be known *a priori*. However, the ultimate speedup results are useful since they serve to show an upper bound on the parallel speedups. The results indicate that, on the average, the dual-stack strategy provides for consistently higher speedup compared to the single-stack case. Furthermore, the range of speedup values observed is significantly tighter in the dual-stack case, indicating the improved balancing of the search tree. Note that for four and eight processors, sublinear speedups were observed in some runs when the single-stack strategy was used, but that all runs for the dual-stack case yielded superlinear speedups. When 32 processors were used, the problem was solved in an average of 56.7 seconds, compared to a serial time of 2164 seconds. It should be noted that the this type of performance is not limited to this particular problem or to this scale of serial time requirement. In many other applications with a wide range of serial time requirements, we have also observed acceleration, not deceleration, anomalies in all runs on different numbers of processors, when the dual-stack scheme was applied.

Because of the acceleration anomaly, one cannot tell from the results of Figure 8 the intrinsic performance of the load balancing algorithm on this problem. To see this it is necessary to make comparisons to a simulated best sequential time, obtained from the idealized case in which the global minimum is used initially as the best upper bound. Speedups calculated on this basis are shown in Figure 9. Now the sequential case and the ultimate speedup cases reflect the same work requirement, and so a comparison of the ultimate speedup results to linear speedup can be used to judge load balancing performance. With ADLB achieving a roughly 96% parallel efficiency on this basis, the effectiveness of this load balancing strategy can be seen.

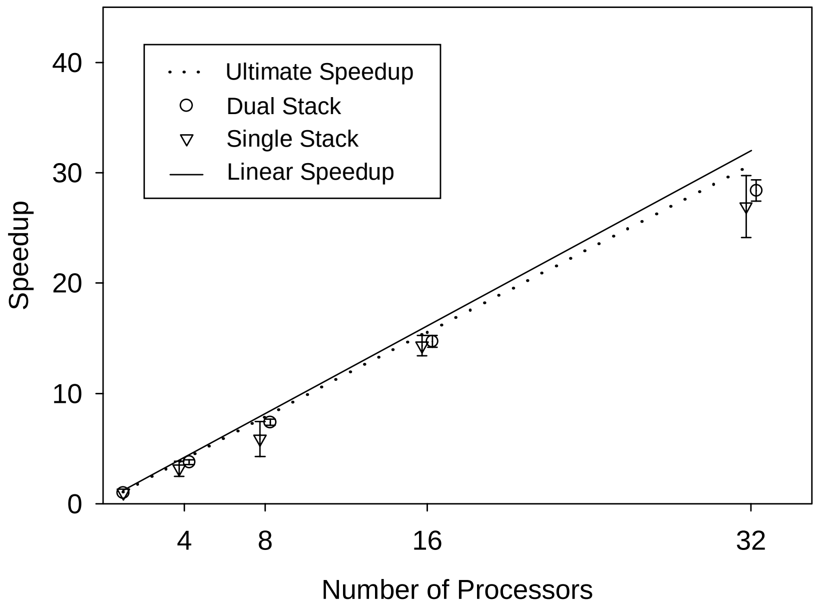

Figure 9: Performance of ADLB on a global optimization problem. Speedups are calculated with respect to a simulated optimal serial case. For clarity, the data symbols are slightly offset horizontally for each value of number of processors.

7 Concluding Remarks

We have described here how load management strategies can be used for effectively solving interval BB and BP problems in parallel on a network-based cluster of workstations. Of the dynamic load balancing algorithms considered, the best performance was achieved by the asynchronous diffusive load balancing (ADLB) approach. This overlaps communication and computation by the use of the asynchronous non-blocking communication functions provided by MPI, and uses a type of diffusive load-adjusting scheme to prevents out-of-work idle states while keeping communication needs small.

The ADLB algorithm was applied in connection with interval analysis, in particular with an interval-Newton/generalized bisection (IN/GB) procedure for reliable nonlinear equation solving and deterministic global optimization. IN/GB provides the capability to find (enclose) all solutions in a nonlinear equation solving problem with mathematical and computational certainty, or the capability to solve global optimization problems with complete certainty. The results of applying ADLB in the equation solving context have shown that the parallel BP algorithm is highly scalable up to 32 processors, especially when using a two-dimensional torus virtual network. In the context of global optimization, the parallel BB algorithm is capable of achieving significantly superlinear speedups. By implementing a new dual stack management scheme in connection with ADLB it appears that consistently high superlinear speedups on optimization problems can be obtained.

Though the results here are based on specific test problems, it should be emphasized that the parallel IN/GB method is general-purpose and can be used in connection with a wide variety of global optimization problems and nonlinear equation solving problems. Also, the load management schemes described here can be applied to a wide variety of other parallel tree search problems in chemical process engineering, such as in process synthesis and process scheduling.

Acknowledgments

This work has been supported in part by the donors of The Petroleum Research Fund, administered by the ACS, under Grant 35979-AC9, by the National Science Foundation Grant EEC97-00537-CRCD, by the Environmental Protection Agency Grant R826-734-01-0. Computational hardware was provided in part through U.S. Army Research Office Grant DAAG55-98-1-0091.

References

Adjiman, C. S., Androulakis, I. P. & Floudas, C. A. (2000) Global optimization of mixed-integer nonlinear problems. *AIChE J.*, **46**, 1769.

Androulakis, I.P. & Floudas, C. A. (1998) Distributed branch and bound algorithms for global optimization. In *Parallel Processing of Discrete Problems*, ed. P. M. Pardalos, Springer-Verlag, Berlin.

Anstreicher, K. M., Brixius, N., Goux, J.-P. & Linderoth, J. (2000) Solving large quadratic assignment problems on computational grids. Presented at 17th International Symposium on Mathematical Programming, Atlanta, GA, August.

Balaji, G. V. & Seader, J. D. (1995) Application of interval-Newton method to chemical engineering problems. *AIChE Symp. Ser.*, **91**(304), 364.

Blumofe, R. D. & Leiserson, C. E. (1994) Scheduling multithreaded computations by work stealing. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, ed. S. Goldwasser, IEEE Computer Society Press, Los Alamitos, CA.

Blumofe, R. D. & Leiserson, C. E. (1999) Scheduling multithreaded computations by work stealing. *J. ACM*, **46**, 720.

Byrne, R. P. & Bogle, I. D. L. (2000) Global optimization of modular process flowsheets. *Ind. Eng. Chem. Res.*, **39**, 4296.

Clausen, J. & Traff, J. L. (1991) Implementations of parallel branch-and-bound algorithms–experience with the graph partitioning problem. *Anns. Opns. Res.*, **33**, 331.

Correa, R. C. (2000) A parallel approximation scheme for the multiprocessor scheduling problem. *Parallel Computing*, **26**, 47.

Correa, R. & Ferreira, A. (1996) Parallel best-first branch-and-bound in discrete optimization: A framework. In *Solving Combinatorial Optimization Problems in Parallel*, eds. A. Ferreira, & P. Pardalos, Springer-Verlag, Berlin.

Cybenko, G. (1989) Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distr. Comp.*, **7**, 278.

Dijkstra, E. W., Feijen, W. H. & van Gasteren, A. J. M. (1983) Derivation of a termination detection algorithm for distributed computations. *Inf. Process. Lett.*, **16**, 217.

Dijkstra, E. W. & Scholten, C. S. (1980) Termination detection for diffusing computations. *Inf. Process. Lett.*, **11**, 1.

Epperly, T. G. W. (1995) *Global Optimization of Nonconvex Nonlinear Programs Using Parallel Branch and Bound.* Ph.D. Thesis, University of Wisconsin, Madison, WI.

Finkel, R. & Manber, U. (1987) DIB–A distributed implementation of backtracking. *ACM Tran. Prog. Lang. and Syst.*, **9**, 235.

Foster, I. (1995) *Designing and Building Parallel Programs - Concepts and Tools for Parallel Software Engineering.* Addison-Wesley, Reading, MA.

Frommer, A. & Mayer, G. (1989) Parallel interval multisplittings. *Numer. Math.*, **56**, 255.

Gan, Q., Yang, Q. & Hu, C. (1994) Parallel all-row preconditioned interval linear solver for nonlinear equations on multiprocessors. *Parallel Computing*, **20**, 1249.

Gau, C.-Y. & Stadtherr, M. A. (1998) Global nonlinear parameter estimation using interval analysis: Parallel computing strategies. Presented at AIChE Annual Meeting, Miami Beach, FL, November.

Gau, C.-Y. & Stadtherr, M. A. (2000) Reliable nonlinear parameter estimation using interval analysis: Error-in-variable approach. *Comput. Chem. Eng.*, **24**, 631.

Gendron, B. & Crainic, T. G. (1994) Parallel branch-and-bound algorithms–survey and synthesis. *Oper. Res.*, **42**, 1042.

Goux, J.-P., Kulkani, S., Linderoth, J. & Yoder, M. (2000) An enabling framework for master-worker computing applications on the computational grid. Technical Report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

Gropp, W., Lusk, E. & Skjellum, A. (1994) *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA.

Gropp, W., Lusk, E. & Thakur, R. (1999) *Using MPI-2: Advanced Features of the Message-Passing Interface.* MIT Press, Cambridge, MA.

Han, J. R., Manousiousthakis, V. & Choi, S. H. (1997) Global optimization of chemical processes using the interval analysis. *Korean J. Chem. Eng.*, **14**, 270.

Hansen, E. R. (1992). *Global Optimization Using Interval Analysis*, Marcel Dekkar, New York, NY.

Harding, S. T., Maranas, C. D., McDonald, C. M. & Floudas, C. A. (1997) Locating all homogeneous azeotropes in multicomponent mixtures. *Ind. Eng. Chem. Res.*, **36**, 160.

Harding, S. T. & Floudas, C. A. (2000) Locating all heterogeneous and reactive azeotropes in multicomponent mixtures. *Ind. Eng. Chem. Res.*, **39**, 1576.

Heirich, A. & Arvo, J. (1998) A competitive analysis of load balancing strategies for parallel ray tracing. *J. Supercomputing*, **12**, 57.

Heirich, A. & Taylor, S. (1995) Load balancing by diffusion. In *Proceedings 24th International Conference on Parallel Programming*, Vol. 3, CRC Press, Boca Raton, FL.

Hu, C. (1996) Parallel solutions for large-scale general sparse nonlinear systems of equations. *J. Comput. Sci. Tech.*, **11**, 257.

Hu, C., Frolov, A., Kearfott, R. B. & Yang, Q. (1995) A general iterative sparse linear solver and its parallelization for interval Newton methods. *Reliable Computing*, **1**, 251.

Kanal, L. N. & Kumar, V. (1988) *Search in Artificial Intelligence*. Springer-Verlag, New York, NY.

Kearfott, R. B. (1996). *Rigorous Global Search: Continuous Problems*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Klepeis, J. L. & Floudas, C. A. (2000) Deterministic global optimization and torsion angle dynamics for molecular structure prediction. *Comput. Chem. Eng.*, **24**, 1761.

Kumar. V., Grama, A., Gupta, A. & Karypis, G. (1994) *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*. Benjamin-Cummings, Redwood City, CA.

Luling, R. & Monien, B. (1989) Two strategies for solving the vertex cover problem on a transputer network. *Lect. Notes in Comp. Sci.*, **392**, 160.

McDonald, C.M. & Floudas, C. A. (1997) GLOPEQ: A new computational tool for the phase and chemical equilibrium problem. *Comput. Chem. Eng.* **21**, 1.

McKeown, G. P., Rayward-Smith, V. J. & Rush, S. A. (1992) Parallel branch-and-bound. In *Advances in Parallel Algorithms*, eds. L. Kronsj & D. Shumsheruddin, Halsted Press, New York.

McKinnon, K. I. M., Millar, C. G. & Mongeau M. (1996) Global optimization for the chemical and phase equilibrium problem using interval analysis. In *State of the Art in Global Optimization: Computational Methods and Applications*, eds. C. A. Floudas & P. M. Pardalos, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Mitra, G., Hai, I. & Hajian, M. T. (1997) A distributed processing algorithm for solving integer problems using a cluster of workstations. *Parallel Computing*, **23**, 733.

Neumaier, A. (1990). *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, UK.

Pardalos, P. M. & Rodgers, G. P. (1990) Parallel branch-and-bound algorithms for quadratic zero-one programming on a hypercube architecture. *Anns. Opns. Res.*, **22**, 271.

Quinn, M. J.: Analysis and implementation of branch-and-bound algorithms on a hypercube multicomputer. *IEEE Trans. Comp.*, **39**, 384.

Rao, V. N. & Kumar, V. (1987) Parallel depth first search. 1. implementation. *Int. J. Paral. Prog.*, **16**, 479.

Rump, S. M. (1999) Fast and parallel interval arithmetic. *BIT*, **39**, 534.

Rushmeier, R. A. (1993) Experiments with parallel branch-and-bound algorithms for the set covering problems. *Oper. Res. Lett.*, **13**, 277.

Schnepper, C. A. & Stadtherr, M. A. (1993) Application of a parallel interval Newton/generalized bisection algorithm to equation-based chemical process flowsheeting. *Interval Computing*, **1993**(4), 40.

Schnepper, C. A. & Stadtherr, M. A. (1996) Robust process simulation using interval methods. *Comput. Chem. Eng.*, **20**, 187.

Sinha, M., Achenie, L. E. K. & Ostrovsky, G. M. (1999) Parallel branch and bound global optimizer for product design. Presented at AIChE Annual Meeting, Dallas, TX, November.

Smith, E. M. B. & Pantelides, C. C. (1999) A symbolic reformulation/spatial branch-and-bound algorithm for the global optimization of nonconvex MINLPs. *Comput. Chem. Eng.*, **23**, 457.

Stradi, B. A., Brennecke, J. F., Kohn, J. P. & Stadtherr, M. A. (2001) Reliable computation of mixture critical points. *AIChE J*, **47**, 212.

Subrahmanyam, S., Kudva, G. K., Bassett, M. H. & Pekny, J. F. (1996) Application of distributed computing to batch plant design and scheduling. *AIChE J.*, **42**, 1648.

Teng, S. (1990) Adaptive parallel algorithms for integral knapsack problems. *J. Parallel Distr. Comp.*, **8**, 400.

Troya, J. & Ortega, M. (1989) A study of parallel branch-and-bound algorithms with best-bound-first search. *Parallel Computing*, **11**, 121.

Vornberger, O. (1986) Implementing branch-and-bound in a ring of processors. *Lect. Notes in Comp. Sci.*, **237**, 157. 157–164

Vornberger, O. (1987) Load balancing in a network of transputers. *Lect. Notes in Comp. Sci.*, **312**, 116.

Xu, G., Scurto, A. M., Castier, M., Brennecke, J. F. & Stadtherr, M. A. (2000) Reliable computation of high pressure solid-fluid equilibrium. *Ind. Eng. Chem. Res.*, **39**, 1624.