Parallel Physical Optimization Algorithms for Allocating Data to Multicomputer Nodes

Nashat Mansour
Geoffrey C. Fox

CRPC-TR92259
April 1992

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Parallel Physical Optimization Algorithms for Allocating Data to Multicomputer Nodes

NASHAT MANSOUR and GEOFFREY C. FOX

School of Computer and Information Science, and Syracuse Center for Computational Science,
Syracuse University, Syracuse, NY 13244

Abstract. Three parallel physical optimization algorithms for allocating irregular data to multicomputer nodes are presented in this paper. They are based on simulated annealing, neural networks and genetic algorithms. The performances of the algorithms are evaluated and compared. All three algorithms deviate from the sequential versions in order to achieve acceptable speed-ups. The parallel simulated annealing (PSA) and neural network (PNN) algorithms include adaptive communication schemes for maintaining both good solutions and reasonable execution times. PNN also includes a data reallocation step to further reduce the communication overhead. Yet, PNN has the smallest efficiency whilst the parallel genetic algorithm (PGA) shows excellent efficiency due to its embarrassing parallelism. The three parallel algorithms maintain the good solution qualities and the non-bias properties of their sequential counterparts. Also, they are scalable. The comparison of the parallel physical algorithms shows their suitability for different applications. For example, PGA yields the best solutions; but, it is the slowest of the three. PNN is the fastest; but, it yields lower quality solutions. PSA's performance lies in the middle.

Keywords. automatic parallelization, data allocation, data partitioning, genetic algorithms, load balancing, mapping, neural networks, physical optimization, simulated annealing.

1. Introduction

One of the first steps in data-parallel programming for distributed-memory MIMD computers, i.e. multicomputers, is the partitioning of the data set into disjoint subsets and the allocation of the subsets to the multicomputer nodes. This step is henceforth referred to as data allocation. The allocation of the data is a primary factor in determining the execution time of the associated parallel algorithm. Hence, the data allocation problem can be considered as an optimization problem that aims for minimizing the total execution time subject to the constraints presented by the parallel algorithm and the parallel machine.

For some regular data sets and multiprocessor topologies, for certain applications, optimal or near-optimal allocations can be easily determined, by inspection or by simple techniques. However, good allocations of general and irregular data sets to various multiprocessor topologies is very difficult to accomplish. In fact, the data allocation problem is an NP-complete resource allocation problem. Several heuristic methods have been suggested for finding good sub-optimal solutions. Important deterministic methods include recursive coordinate bisection, recursive graph bisection, recursive spectral bisection, mincut based methods, clustering techniques, geometry based mapping, block based spatial decomposition, and scattered decomposition [Berger 1987; Chrisochoides et al. 1991; Ercal 1988; Fox 1988; Fox et al. 1988; Houstis et al. 1990; Nolting 1991; Pothen

et al. 1990; Simon 1991; Walker 1990; Williams 1991]. Deterministic heuristics are fast; but, they usually favor certain problem or multicomputer configurations. Another class of methods for data allocation consists of physical optimization methods, which employ techniques from natural sciences [Fox 1990]. Simulated annealing [Kirkpatrick et al. 1983], from statistical physics, views optimization as finding the ground state of a system in a heat bath. The system temperature is gradually reduced and a Monte Carlo algorithm is used to simulate the system's behavior at each temperature. Biologically motivated neural networks [Hopfield and Tank 1986; Fox and Furmanski 1988] are based on a mean field theory derivation, from physics, to quickly find good minima in the search space. Genetic algorithms [Goldberg 1989; Holland 1975], from evolutionary biology, are also used for optimization problems since they maximize the fitness of evolving population of structures. These three paradigms have been applied to the data allocation problem [Flower et al 1987; Fox et al. 1988; Fox and Furmanski 1988; Mansour and Fox 1991a] and their performances have been evaluated and compared for a number of test cases [Mansour and Fox 1991b]. They yield good sub-optimal solutions and have more general applicability than deterministic methods, although they are slower. To speed up their operation, the use of hybrid techniques and the addition of a preprocessing graph contraction step were recently explored [Mansour and Fox 1991c]. The implementation of both suggestions led to a reduction in execution time. However, the parallelization of the physical methods is particularly attractive for fast execution, especially that the multicomputer to whose nodes data are to be allocated is available anyway. In fact, for large or dynamically varying problems, parallel implementation is indispensable to circumvent memory space as well as time constraints.

A large class of parallel algorithms for scientific and engineering problems is loosely-synchronous [Fox et al. 1988]. Loosely-synchronous algorithms consist of compute-communicate cycles. In each cycle, multicomputer nodes carry out local computations on their allocated data subsets concurrently and independently. Then, nodes communicate to exchange boundary information. In this model, the total execution time of a parallel program is determined by the slowest node, which is responsible for the maximum combination of computational work-load and communication cost. In this paper, we concentrate on allocating irregular data sets to hypercube nodes for loosely-synchronous programs; nevertheless, the methods are not restricted to these cases. This work constitutes a part of a broader automatic parallelization effort, the Fortran D programming system, which is a joint project of Rice University, ICASE and Syracuse University [Fox et al 1990]. In the Fortran D system, we are interested in including a number of data allocation schemes that suit a variety of problems and multiprocessors. High quality data allocation is needed for irregular problems, such as finite elements in complex domains, and particle dynamics with finite range forces. We also need to target SIMD and MIMD parallel machines with a variety of topologies and different communication mechanisms.

In this paper, we present three parallel physical optimization algorithms: parallel simulated annealing (PSA), parallel neural network (PNN) and parallel genetic algorithm (PGA). All three algo-

rithms deviate from their sequential counterparts, in order to achieve acceptable speed-ups. Thus, we first investigate their individual properties, in comparison with the sequential ones, using four test cases with different features. Then, their performances are compared. Two recursive bisection methods are also included in the comparison as representative deterministic heuristics. It is worth noting, however, that the three parallel algorithms can be regarded as general paradigms for parallelizing the physical optimization methods, whose application is not restricted to data allocation.

This paper is organized as follows. Section 2 includes a definition of the problem. In Sections 3, 4, and 5, the three parallel algorithms are presented and their individual properties are discussed. In Section 6, a comparison of their performances is given. In Section 7, conclusions are presented.

2. Problem Description

Given a problem and a parallel algorithm, ALGO, the problem can be represented by a computation graph with P vertices and edges. The P vertices represent the data objects in the underlying data set, and the edges represent data dependences specified by ALGO. Data allocation refers to dividing the computation graph among N hypercube nodes for minimizing the total execution time of the parallel program, associated with ALGO. Thus, the execution time represents the objective function to be minimized; it is determined by the slowest node and is typically given by

$$OF_{typ} = \max_n \{ W(n) + \sum_m C(n,m) \}, \quad (1)$$

where $W(n)$ is the computational load for node n, and $C(n,m)$ is the cost of communication with node m. OF_{typ} OF_{exact} is computationally expensive for incremental changes needed in the physical algorithms. It can be replaced with a cheaper approximate quadratic objective function, given by

$$OF_{appr} = \gamma \sum_n W^2(n) + \beta R \sum_n \sum_m C(n,m), \quad (2)$$

where R is a machine dependent ratio of communication to computation of one word; γ and β are scaling factors expressing the relative importance of the computation and the communication terms, respectively. It is assumed, in this work, that the communication cost is given by Hamming distances and that latency and link contention can be ignored. Obviously, the determination of accurate cost functions depends on the particular algorithm and the the specific architecture and software of the multicomputer. Thus, OF_{typ} and OF_{appr} and their assumptions refer to one set of algorithm and machine characteristics. However, they are not restrictive and can be modified to fit other characteristics. Further, their assumptions are considered reasonable for typical cases. For example, message latency is relatively small when message sizes are sufficiently large. Also, the conservative Hamming metric favors near-neighbor communication and, hence, would reduce the likelihood of link contention.

Although OF_{appr} is used for the physical optimization methods, the data allocations they produce

are still evaluated using OF_{typ} . The quality of a solution of the physical algorithms refers to the efficiency of the parallel program, associated with ALGO, and is defined as $\Sigma_n W(n) /(N*OF_{typ})$. The solution itself is represented as an allocation vector, ALLOC[], with length P; an element n at index p indicates that data object p is allocated to node n.

The implementation of the parallel physical algorithms and the experimental work reported below involve the execution of the physical algorithms on N_H-node NCUBE/2. The interesting case, when N_H=N, occurs when the physical data allocation algorithms are executed on the same hypercube to which data are to be allocated. The efficiency of a physical algorithm is the time of the sequential algorithm (N_H=1) divided by the product of N_H and the parallel time (N_H nodes).

3. Parallel Simulated Annealing Algorithm

3.1. Algorithm

Simulated Annealing (SA) for data allocation usually starts with a random allocation vector, ALLOC[]. This random configuration is associated with a high temperature and a high energy state. Energy is given by the objective function, OF_{appr}, and the goal of SA is to find a configuration that corresponds to the ground state (minimum). The configuration is cooled down from the initial temperature according to a schedule. In sequential SA [Mansour and Fox 1991b], a number of successive perturbations, or moves, to the configuration are performed at each temperature, until thermal equilibrium is reached. A perturbation is accomplished by a random reallocation of a randomly chosen data object, i.e. by a random change to an element in ALLOC[]. A perturbation that decreases the objective function value (downhill move) is always accepted; a perturbation that increases it (uphill move) is allowed only with a temperature-dependent probability. A change in OF_{appr} due to reallocating data object p from node n to m depends on the total numbers of objects allocated to n and m, in ALLOC[], and on the change in communication cost due to the reallocation [Mansour and Fox 1991a]. The numbers of objects allocated to the nodes are henceforth represented by vector ALLOC_NUM[] with length N. Thermal equilibrium at each temperature is reached when the numbers of attempted or accepted perturbations exceeds some values, MAX_ATTEMPTS and MAX_ACCEPTS, respectively. The cooling schedule determines the next temperature as a fraction of the current one. In our implementation, this fraction varies between 0.91 and 0.99 in a direction that is opposite to the slope of the number of accepted perturbations. The initial and freezing temperatures correspond to a high probability and a very small probability of accepting uphill moves, respectively.

The SA algorithm is very sequential, since it assumes one perturbation attempt at one time. A number of strategies have been suggested for Parallel Simulated Annealing (PSA) with acceptable speed-ups [Greening 1990; Eglese 1990]. The strategy adopted in this paper is based on executing

the sequential SA concurrently and loosely-synchronously in all nodes of the hypercube, where the nodes contain disjoint segments of the data allocation vector, ALLOC[], and the associated computation subgraphs. This strategy is called asynchronous in [Greening 1990] and error SA in [Eglese 1990]; we henceforth refer to it simply as PSA. It is adopted in this work because it is faster and more scalable than the other known strategies, at least for our application. This strategy was applied to dynamic load balancing in [Williams 1991]. Our design of PSA and its application to data allocation have similar features. However, our adaptive communication scheme, illustrated below, is more flexible and can lead to better speed-ups. In addition, we include explicit discussion of the design choices made and employ diverse test cases, including 3-dimensional unstructured tetrahedral meshes, for performance evaluations.

Clearly, PSA is not faithful to the sequential SA because perturbations can occur concurrently and not successively. Hence, the local node (in N_H-node cube) view of a change in objective function, ΔOF, due to a reallocation, in the local ALLOC[] segment, of a data object from node *n1* to *n2* is not always consistent with the global view. Since ΔOF depends on ALLOC_NUM[*n1*] and ALLOC_NUM[*n2*] and on the change in communication costs, three types of inconsistencies can be identified in PSA. The first type occurs if two (or more) concurrent perturbations, in different nodes in N_H-node cube, involve data objects *p1* and *p2* (or more) with ALLOC[*p1*] = ALLOC[*p2*]. The second inconsistency type is concerned with ALLOC_NUM[] in different nodes. The third inconsistency type occurs due to local use of outdated information about nonlocal elements of ALLOC[] that are involved in the communication part of ΔOF. We emphasize, again, that these inconsistencies are due to the deviation from the sequential algorithm. If they are allowed to accumulate, they lead to degeneration. Inconsistency accumulation leads to either a convergence to a bad minimum or an increase in the number of passes needed to maintain reasonable solutions, causing a decrease in speed-up in the latter case. The design of PSA discussed in the remainder of this section consists of steps, which address these potential sources of inconsistencies, for producing solutions comparable to those of sequential SA in acceptable execution times. The design strategy is based on frequently unifying the local views of the global state in order to prevent degeneration. Unifying involves inter-node communication, which reduces the speed-up of PSA. However, an adaptive scheme is devised below for reducing the communication cost.

The first question that arises for PSA is how to allocate ALLOC[] and the associated computation graph to N_H hypercube nodes, which is the same problem that PSA aims for solving in the first place. We have chosen a negligible-time naive allocation scheme, where ALLOC[] is split into contiguous segments, DS_0, DS_1, ... that are as equal as possible. The elements in DS_i and the corresponding computation subgraph are allocated to node *i*. Clearly, this allocation scheme is far from optimal and the speed-up for the PSA algorithm is sensitive to the numbering order of the input data because it determines the amount of inter-node communication.

An outline of PSA is given in Figure 1. After the naive allocation step, the algorithm includes

boundary communication and global summation steps, in addition to the local sequential SA procedure. The global summation of the number of attempted and accepted perturbations in all nodes is required to detect thermal equilibrium. At high temperatures, the number of accepted moves at one temperature is high and, thus, the number of elements that change in ALLOC[] at each attempt might be large. Therefore, the magnitudes of the three inconsistency types, described above, would grow rapidly if local SA steps proceed without correction. Corrections can be accomplished in two ways: a global sum operation to unify ALLOC_NUM[] in all nodes, correcting the second inconsistency type, and inter-node communication of boundary information, correcting the third inconsistency. The first inconsistency type occurs randomly and contributes to an inherent "erroneousness" of PSA, mainly at high temperatures.

```
Determine segment of ALLOC[] and computation subgraph allocated to my_node;
Determine inter-node communication information (nodes, boundary elements);
Generate random ALLOC[] segment;
Determine Initial temperature, T(0) (1 global comm.);
Determine Freezing temperature;
Communicate boundary information;
Global summation for ALLOC_NUM[];
while ( T(i) > Tfreeze and NOT converged ) do
   Determine ν_mvs;
   while (not equilibrium) do
      Local SA step;
      Update #attempted and #accepted perturbations at ν_mvs;
      Communicate boundary info at ν_bdry;
      Update ALLOC_NUM[] at ν_mvs (global sums);
   end_while
   T(i) = k * T(i-1)
end_while (end 1 pass)
```

Figure 1. PSA node algorithm for data allocation.

Obviously, it would be disastrous for PSA's speed-up to make the above-mentioned corrections at every attempted move or at a high frequency. On the other hand, low-frequency corrections would lead to degeneration. However, as temperature decreases the number of accepted moves decreases and, thus, the likelihood of inconsistencies also decreases; at low temperatures PSA approaches SA. This observation points to a remedy for the speed-solution dilemma, which is the use of an adaptive correction scheme. In this scheme, the frequency of global summation of the numbers of attempted and accepted moves, ν_{mvs}, can be annealed, and the frequencies of updating ALLOC_NUM[], ν_{Nsum}, and of inter-node communication, ν_{bdry}, can be made adaptive to the number of accepted moves. Specifically, ν_{mvs} is decreased linearly from one every attempt, at initial temperature, to a few times (1 to 10) per MAX_ATTEMPTS per node, at freezing temperature. ν_{Nsum}

equals one every few, say two, accepted moves per node. v_{bdry} equals one every $3*(P/N_H)*(1/$ SMALL_GRAIN) accepted moves per node, where SMALL_GRAIN is the size of a grain whose boundary elements would be about one third. This value of v_{bdry} makes use of the fact that the boundary elements that are needed in other nodes are only a fraction of the local grain size, (P/N_H), and that this fraction, generally, decreases with bigger granularity. We emphasize that the number of global summation updates of ALLOC_NUM[] and the number of inter-node communication operations are decreased with temperature due to their dependence on the decreasing number of accepted moves. We also note here that the experimental results below show that while such communication frequencies maintain reasonable speed-ups by allowing inconsistencies to occur in between corrections, these inconsistencies are corrected so frequently that the final solution quality is not degraded.

3.2. Experimental Results

In this subsection, the properties of PSA are investigated; the quality of its solutions, efficiency, and robustness are experimentally examined. We give particular attention to possible differences in performance between PSA and SA due to the inconsistencies allowed in PSA. Three structures, shown in Figure 2, are used. FEM-W is a 545-point unstructured tetrahedral finite-element coarse discretization of an aircraft wing. GRID-P is a 551-point 2-dimensional grid-based discretization of a broken plate having a large variation in the spatial density of its points. FEM-2 is a 198-point 3-dimensional structure. Four allocation test cases are considered: Test1 is allocation of FEM-W to 4-cube (N=16), Test2 is FEM-W to 3-cube (N=8), Test3 is GRID-P to 4-cube, and Test4 is FEM-2 to 4-cube. The four test cases provide different geometries, granularities, spatial dimensionality, and graph connectivities. The solution quality is the concurrent efficiency that results from the data allocation solution, as explained in Section 2. All results in the paper are averages of ten runs.

Figures 3, 4, 7 and 8 show the efficiency and the number of passes (temperatures) of PSA for different number of nodes, N_H, for the four test cases. The notion of efficiency, or speed-up, is not precise here since the parallel algorithm deviates from the sequential one. However, it still serves as a measure of the parallelizability of the annealing algorithm. This comment also applies to PNN and PGA in the following sections. All figures show a decrease in efficiency when N_H increases and granularity decreases. Efficiency drops due to an increase in the relative cost of global summation operations and inter-node communication. It decreases more rapidly for FEM-2 because it has the smallest granularity. The curves of the number of passes do not show a uniform and consistent behavior. However, it can be seen in the four cases that, with the adaptive communication frequencies, there is no significant increase in the number of passes for larger N_H. Hence, the inconsistencies allowed in PSA do not lead to significant delays in the progress towards the final solution. But, the communication cost per pass increases with N_H.

PSA's solutions are given in Figures 5, 6, 9 and 10. Clearly, PSA's solutions are very close to those

of sequential SA (N_{II}=1) in all cases except for the small granularity cases of FEM-2, which is still within a small fraction. Thus, the deviation of PSA from its sequential counterpart does not result in premature convergence and degradation in solution quality, as long as the grain size is not too small. Consequently, the solutions and the efficiency figures show that our scheme of annealed v_{mvs} and adaptive v_{Nsum} and v_{bdry} leads to both, preservation of SA's solution quality and acceptable efficiency values.

Sequential SA is fairly robust. Robustness, in this paper, refers to the insensitivity to problem and design parameters. PSA includes additional parameters, namely the communication frequencies. Although the adaptive communication scheme described above is adequate, these parameters make PSA somewhat less robust than sequential SA.

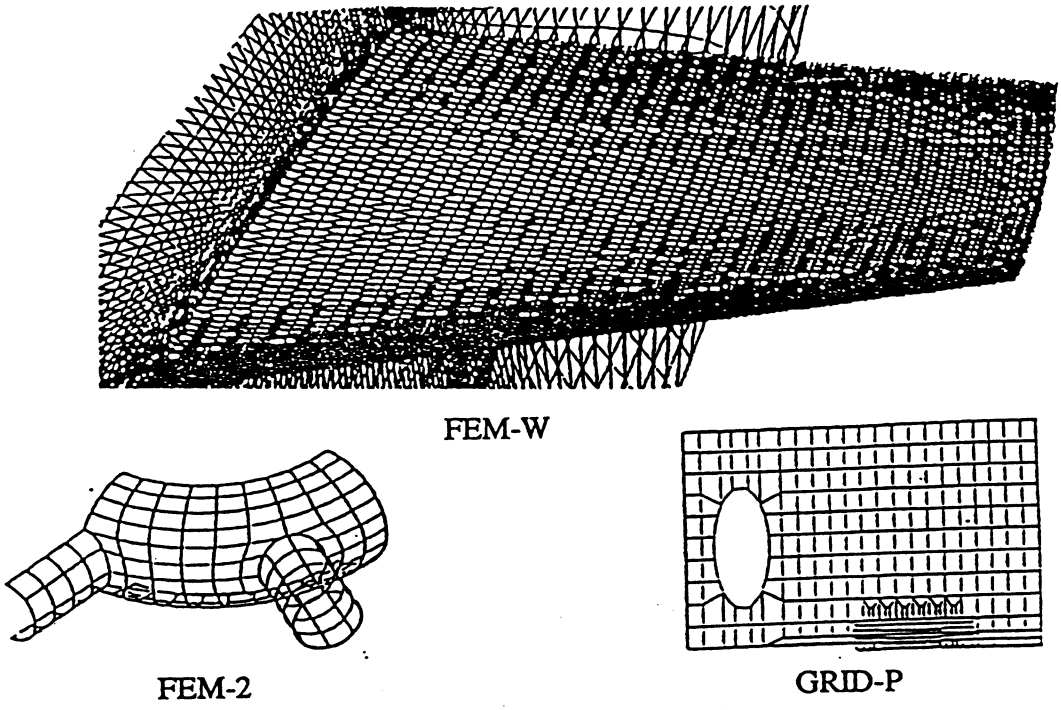

Figure 2. Three examples.

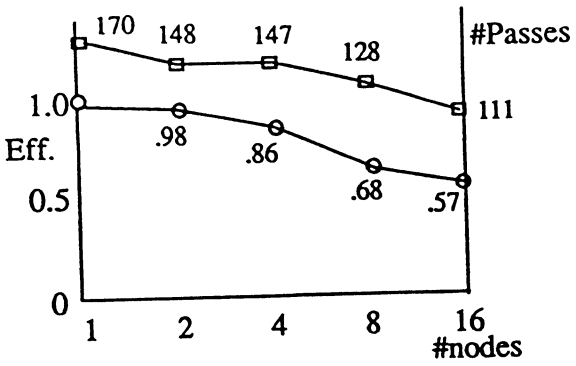

Figure 3. PSA efficiency and #passes for Test1.

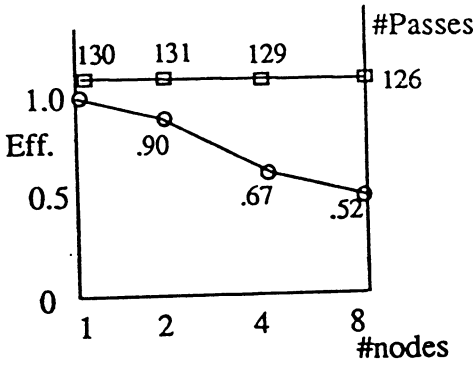

Figure 4. PSA efficiency and #passes for Test2.

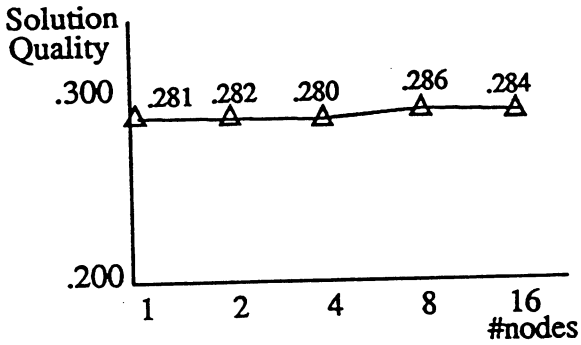

Figure 5. PSA solution for Test1.

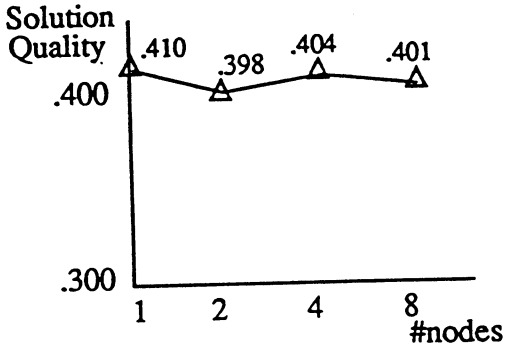

Figure 6. PSA solution for Test2.

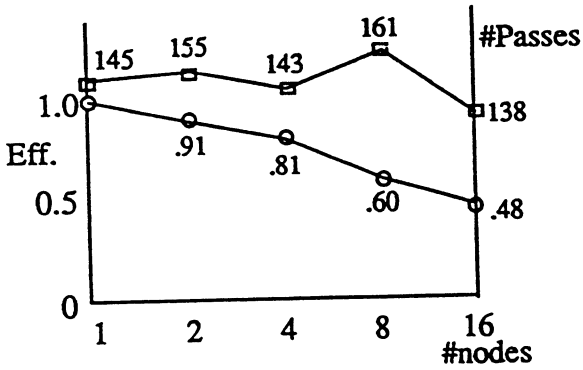

Figure 7. PSA efficiency and #passes for Test3.

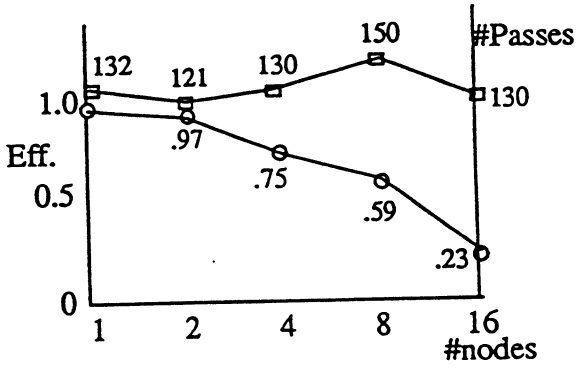

Figure 8. PSA efficiency and #passes for Test4.

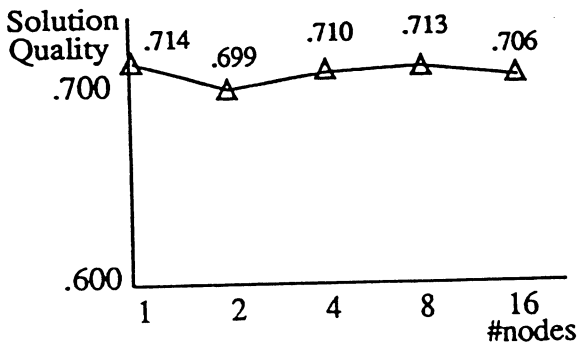

Figure 9. PSA solution for Test3.

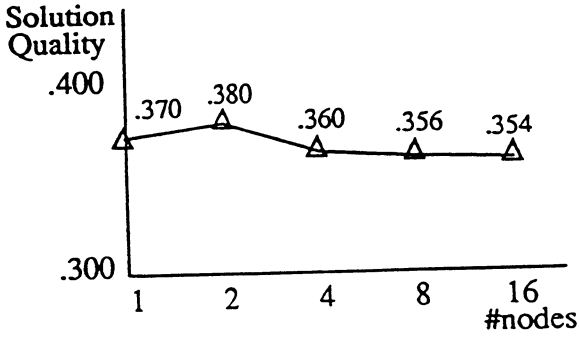

Figure 10. PSA solution for Test4.

4. Parallel Neural Network Algorithm

4.1. Algorithm

The sequential Bold Neural Network (NN) algorithm [Fox and Furmanski 1988; Mansour and Fox 1991b] consists of d iterations, where $d = logN$ is the dimension of the hypercube to which the data objects are to be allocated. In each iteration, i, the current 2^{i-1} subsets of objects are bisected and allocated to 2^i subcubes. After the d-th iteration, N subsets become allocated to the nodes and, thus, the allocation vector, ALLOC[], becomes fully specified. In each iteration, random spin values are generated in the -1 to 1 range. Each spin $s(e,i,t)$ is associated with data object e, where i corresponds to the i-th bit in a node label and t stands for time. A number of sweeps over the entire problem are then performed until convergence. At convergence, new spin subdomains, SPN_SD_k with $k = 1$ to 2^i, are formed. The spin direction, 1 (up) or -1 (down), determines the subdomain in which the spin (or data object) lies. In each sweep, the spins are sampled at random and their values are updated as follows:

$$s(e,i,t+1) = tanh\ \{ -\alpha\ s(e,i,t) + \beta \sum_{s'} G(s,s') - (\frac{\gamma}{D_{i-1}})\ SPN\_SUM[SPN\_SD_k] \}, \qquad (3)$$

where α, β and γ are scaling factors; D_{i-1} is the size of the current subdomain (to be bisected) to which data element e belongs; SPN_SUM[] is a vector of (up to N) net sums of spin values in current spin subdomains; G is the coupling matrix determined from the problem's computation graph. The derivation of equation (3) uses OF_{appr} as the energy function to be minimized. The second term in (3) represents local interaction which aligns neighboring spins. The third term represents a long-range force that spans the entire current subdomain, to be bisected, and is responsible for the up/down spin balance in this subdomain. The first term is a noise term.

The Parallel Neural Network (PNN) algorithm takes a similar approach to that of PSA. It is based on executing the sequential NN concurrently and loosely-synchronously in the N_H hypercube nodes, where the local memories of the nodes contain disjoint subsets of spins, i.e. data objects, and their associated computation subgraphs. This gives rise to parallelization issues similar to those encountered in PSA. However, the NN algorithm suggests different ways to address these issues. For example, the initial allocation of spins to N_H nodes follows the naive scheme used in PSA for the first bisection step of PNN. But, the results of the bisection steps themselves allow a reallocation of spins which subsequently reduces the communication overhead. A suitable reallocation scheme is discussed below.

PNN also deviates from the sequential operation of NN. According to equation (3), an update of a spin value $s(e)$ of spin subdomain SPN_SD_k at any time in a sweep depends upon the most recent values of the neighboring spins (second term) and the most recent value of the sum, $SPN\_SUM[SPN\_SD_k]$, of the values of all spins that lie in SPN_SD_k. Concurrent spin updating in PNN, therefore, involves two possible sources of inconsistencies. The first source is the local use of outdated nonlocal spin values. The second source of inconsistency is the concurrent updating of

spins that belong to the same subdomain, which leads to what we henceforth refer to as inherent "erroneousness" of PNN. The design of PNN consists of steps that deal with these inconsistencies by updating boundary information and global spin summation. Similarly to PSA, an important component of PNN's design is a communication scheme which exploits the characteristics of PNN and is guided by the requirement to maintain both, acceptable solution quality and reasonable execution time.

Similarly to PSA, the question related to outdated neighboring spin values can be stated as follows: How often should inter-node communication take place for correcting local information about neighboring spin values in other nodes and yet keeping the communication frequency as small as possible? Before answering the question, we make three observations. The first observation is that the boundary spins, to be communicated, form only a fraction of the spins allocated to a node. Hence, their values are neither updated nor needed every spin update. The second observation is experimental; after a number of sweeps, #SWP1, spin domains are formed, although not in their final configuration. The spins in the middle of a domain become permanently aligned, and only the spins near the boundaries of the domain might change value/direction. Figure 11 depicts an example of spin alignment after a number of update sweeps for the first bisection and their naive allocation to a 4-node cube. It shows that a large proportion of spins that keep changing direction after the formation of domains could lie entirely within a node and that spin values at the node boundaries might not change for many updates in a sweep and, therefore, need not be communicated between nodes frequently. The third observation is that some inconsistencies resulting from the use of outdated neighboring spin values can be tolerated since PNN is inherently "erroneous" anyway. These observations lead to an internode communication scheme, whereby the frequency of boundary communication, ν_{bdry}, is attenuated with the number of sweeps. ν_{bdry} starts with a high value, 3 to 6 times per sweep, until #SWP1 and is then gradually decreased to once per sweep after #SWP2. #SWP1 and #SWP2 are application dependent. They depend on the dimensionality of the problem and the degree of connectivity of the computation graph. For example, for 3-dimensional graphs with high vertex degree, such as that for FEM-W, suitable #SWP1 and #SWP2 would be $2P^{1/3}$ and $4P^{1/3}$, respectively.

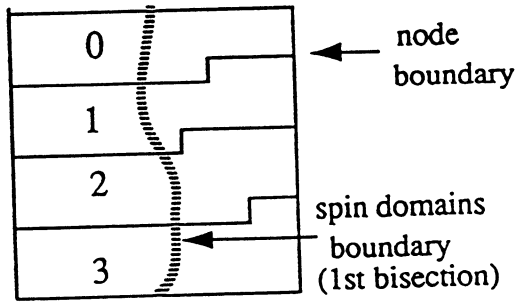

Figure 11. Boundaries of nodes and spin domains.

The question of updating spin sums, SPN_SUM[], by global operations is similar to that of

ALLOC_NUM[] in PSA. Here also, the frequency of global summation, ν_{Ssum}, should be kept to a minimum, whilst not allowing the inconsistency in the local values to grow beyond an acceptable level. This question can be addressed in two ways. However, we first note that we are interested in net sums of spin values, not in individual values, where inconsistencies contributing to a sum might cancel each other. Further, due to the inherent erroneousness of PNN, some magnitude of inconsistency can be tolerated and, hence, the global sums need not be evaluated every spin update. One way to address the question of updating SPN_SUM[] is simply to perform global summation with a frequency that decreases with the number of sweeps. The reason for decreasing the summation frequency is, as mentioned above, that after the formation of spin domains, a smaller number of spins change values/directions until convergence. A suitable scheme has been empirically found to be as follows: ν_{Ssum} starts with a high value, 3 to 6 spin updates in a sweep, until #SWP1 and is then decreased gradually to a minimum of once per sweep.

Another way to decrease the cost of updating SPN_SUM[] is based on spin reallocation. Given the small amount of computations required for a spin update, the relative cost of global summation, at ν_{Ssum}, for updating SPN_SUM[] rises rapidly and PNN's speed-up starts to vanish for smaller granularity and large hypercubes. To decrease the summation cost, we reallocate spins to nodes after each bisection pass so that summation will subsequently be needed within smaller subcubes instead of the entire cube. The initial allocation remains as described above. After the first bisection pass, two spin domains are generated. The spins in domain 0 can be reallocated to nodes in subcube xx...x0 and those in domain 1 to subcube xx...x1, where x is a don't-care symbol, as shown in Figure 12. That is, spins are reallocated so that boundaries of spin domains coincide with node boundaries. In the second bisection pass, each of SPN_SUM[0] and SPN_SUM[1] is needed in only one subcube. Thus, updating the two values can be carried out within the two subcubes concurrently, which reduces the cost of the global operation to a half of what it is in the first bisection pass. Similarly, after the *i*-th bisection, spins in subdomain *j* are reallocated to the subcube whose node numbers agree in the (*i-1*)-th least-significant bits with those of *j*. The cost of updating SPN_SUM[] is therefore halved with each successive bisection pass, which yields significant overhead reduction for large hypercubes and improves PNN's scalability. The overhead due to reallocation has been, experimentally, found to be relatively small and is, anyway, acceptable since it places the data objects where they should be allocated for the parallel program, ALGO. However, the cost of internode communication might increase for some problems because of possible increase in the number communicating nodes, with smaller messages. For example, in Figure 12, node 2 has three neighboring nodes instead of two, as in Figure 11. The PNN algorithm is summarized in Figure 13.

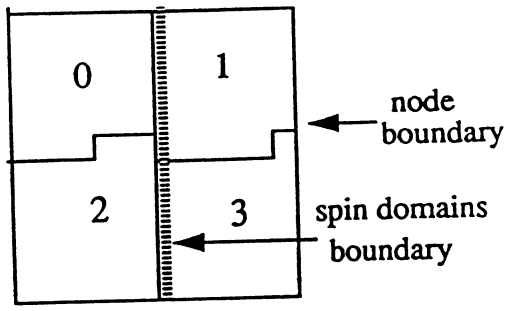

Figure 12. Spin reallocation after the first bisection.

```
Determine spin subset and computation subgraph allocated to my_node;
Find inter-node communication info (nodes,boundary spins);
for i = 0 to (lg2N -1) do
   if (i > 0) then /* after 1st bisection */
      Reallocate_spins() and determine my_subcube;
      Find inter-node communication info (nodes,boundary spins);
   endif
   random spin values, s(e,i,t), e=0 to P-1;
   repeat
      for all spins 0 to P-1 do
         Global_add(SPN_SSUM[],my_subcube) at vSsum;
         Communicate_boundary(spin values) at vbdry;
         Pick a spin, e, randomly;
         Compute s(e,i,t+1); /* equation (3) */
      end-for
   until (convergence)
   Set bit i in the neurons (0 or 1);
end-for
```

Figure 13. PNN node algorithm for data allocation.

4.2 Experimental Results

Figures 14, 15, 18 and 19 show the efficiency of PNN for the four test cases for different N_H. All four figures show a rapid decrease in efficiency with increasing N_H and decreasing granularity. The sharp fall in efficiency reflects the small amount of computation performed by PNN per spin and, thus, the rapid increase in the relative cost of global and semi-global summation operations for SPN_SUM[] and in the cost of inter-node communication. However, it is clear that for reasonable grain sizes the efficiency is acceptable.

Figures 16, 17, 20 and 21 show the quality of the solutions produced by PNN. It seems that a

decrease in granularity leads to a decrease in quality. However, the decrease is small in some cases and negligible in others. The decrease can be attributed to considerable increase in the relative magnitudes of the inconsistencies, in between communication operations, for small grain sizes and can be ignored for reasonable granularities.

Sequential NN is quite robust; the quality of its solutions is not sensitive to design and problem parameters. In PNN, ν_{Ssum} and ν_{bdry} are two additional parameters whose values affect solutions, needles to mention efficiency. Although the empirically derived frequencies are adequate, experimental experience has shown that the inclusion of these parameters results in reduced robustness.

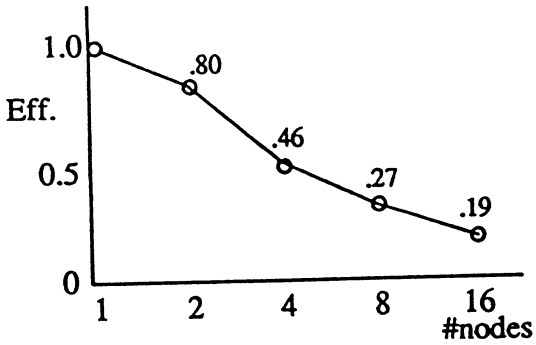

Figure 14. PNN efficiency for Test1.

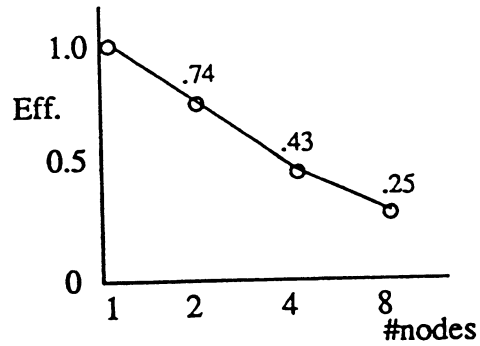

Figure 15. PNN efficiency for Test2.

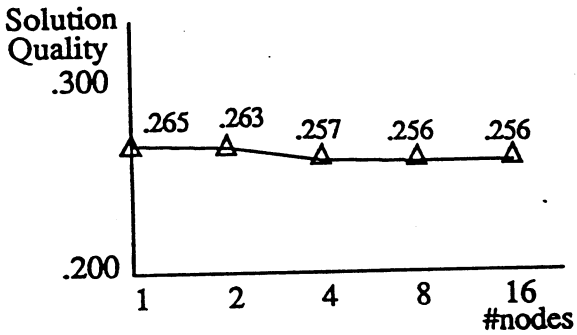

Figure 16. PNN solution for Test1.

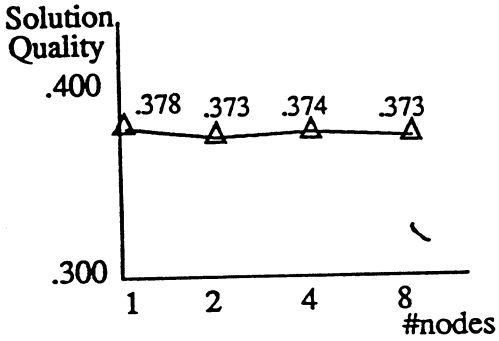

Figure 17. PNN solution for Test2.

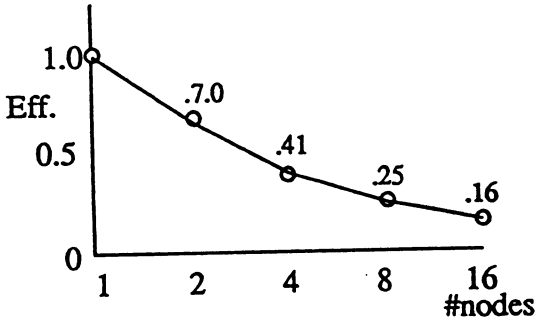

Figure 18. PNN efficiency for Test3.

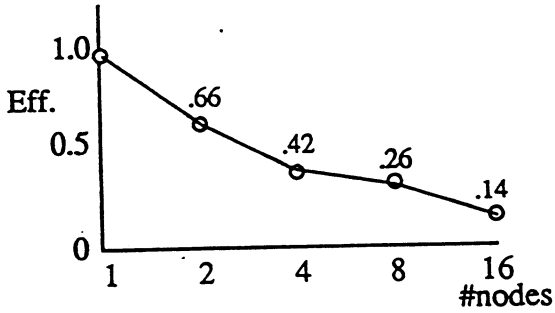

Figure 19. PNN efficiency for Test4.

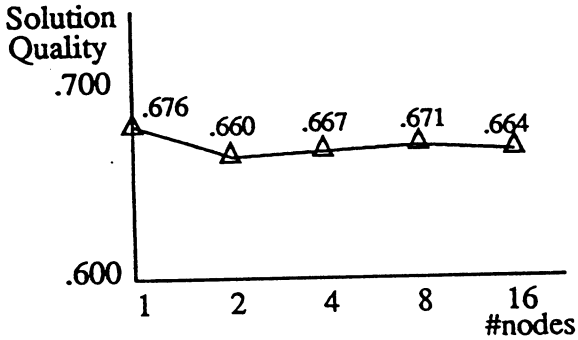

Figure 20. PNN solution for Test3.

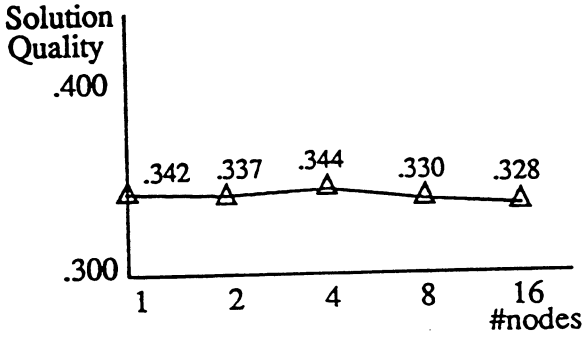

Figure 21. PNN solution for Test4.

5. Parallel Genetic Algorithm

5.1. Algorithm

A sequential hybrid Genetic Algorithm (GA) starts with a population of randomly-generated possible solutions, ALLOC[], and evolves over a number of generations [Mansour and Fox 1991a]. Each ALLOC[] in the population is called an individual. In each generation, individuals reproduce and genetic operators are then applied to the survivors. Individuals are assigned reproduction trials according to their fitnesses, the reciprocal of OF_{typ}. Then, pairs of mates are randomly selected from the reproduction trials list. Crossover is applied to the selected parents by randomly choosing a contiguously numbered segment of data objects and swapping the respective node allocations. Mutation, at a low rate, represents a random reallocation of a data object. The algorithm is hybridized by the inclusion of a hill-climbing procedure for the individuals. The resultant offsprings form the next generation and the evolution process is continued until convergence.

The Parallel Genetic Algorithm (PGA) described in this section represents a different approach to that of PSA and PNN. It is based on discontinuous distributed population structures and the shifting balance theory of evolution. In PGA, the population is divided into subpopulations, i.e. demes, to which reproduction is restricted. This is a deviation from the classic GA where the whole population is a single mating unit. Such a distributed population structure is a better simulation of natural population and represents a natural way for preserving genetic diversity and, thus, for circumventing the problem of premature convergence often encountered in the single mating GA model.

PGA's model of evolution is based on Wright's shifting balance theory [Wright 1977], in which evolution consists of three phases. The first and second phases are for random genetic drift and local intrademe selection. In these phases, the isolated demes explore different parts of the solution space. Also, favorable gene combinations which correspond to different fitness peaks in the space are preserved. In the third phase, interdeme selection takes place; demes with higher fitness expand in size and shift the genetic structure of adjacent demes until they come under the control of the higher fitness demes. The fitter genetic building blocks spread throughout the population in widening concentric circles. This evolution model possesses the property of intrinsic parallelism, which refers to the concurrent and independent exploration by the demes of different regions in the solution space. It has been chosen, in contrast with other distributed population models, because it is coarse-grain and promises faster convergence.

The shifting balance model lends itself to an embarrassingly parallel implementation. The population is divided into equal N_H demes assigned to the nodes of the hypercube and interdeme selection occurs as migration between neighboring nodes with direct interconnections. Clearly, PGA is perfectly load balanced. Further, the time required for the drift and local selection phases, which constitute local computations, is much greater than that for the interdeme selection phase, which is

associated with inter-node communication. Therefore, the communication overhead is very small. An outline of a shifting balance theory based PGA is given in Figure 22. The drift and local selection phases are simulated by a simplified version of the sequential hybrid GA described above. The simplifications cover features that were included in GA to alleviate premature convergence. Drift and local selection are performed within each deme, i.e. in each node, for D generations. Interdeme selection is simulated as a one-way migration of the $M\%$ best individuals from the fitter demes to the less fit neighboring demes, allowing the better structures to spread throughout the population. Neighborhood, in this context, is associated with the physical interconnection network. Fitter demes send copies of their $M\%$ best individuals. Receiving demes replace their worst $M\%$ individuals with the incoming migrants, assuming limited resources for a constant population size. The parameters D and M depend on the deme size and the required convergence rate. Suitable values have been empirically found to be about half the deme size for D and 20% to 40% of the deme size for M [Mansour and Fox 1991d].

```
Random generation of initial deme.
Evaluate fitness of this deme.
repeat
    /* Drift and local selection phases */
    for (D drift generations) do
        Perform Sequential GA
    endfor
    /* 1-way migration phase (interdeme selection) */
    Find the highest fitness peak in the immediate
                neighborhood (including this deme)
    if (mynode contains highest peak) then
        Send copies of M migrants to less fit neighbors
    else
        Receive M migrants from the fittest neighbor
        Replace M weakest individuals with migrants
    end-if-else
until convergence
Solution = Fittest individual
```

Figure 22. PGA node algorithm for data allocation.

5.2 Experimental Results

Figures 23, 24, 27 and 28 show the efficiency and the number of generations. All figures show superlinear efficiencies, which increase with N_H. This indicates that, in contrast with the sequential classic GA, intrinsic parallelism tends to evolve good solutions in a shorter time, which is evident in the number of generations taken. In this sense, superlinear efficiency is a property of PGA. As

expected, inter-node communication was found responsible for less than 3% of the time in the worst case attempted.

The quality of the solutions are shown in Figures 25, 26, 29 and 30. They are close to the sequential solutions, although they show a small decrease in quality for the largest N_H. Previous work [Mansour and Fox 1991d] showed that PGA would find better solutions than GA if the design parameters were appropriately tuned and the population size is sufficiently large. In this work, we favor general setting of parameters and faster execution to small improvements in solution quality.

Sequential GA is more sensitive to design and problem parameters than SA and NN. PGA has reduced sensitivity with respect to some parameters, such as operator frequencies, since it embodies another mechanism for controlling premature convergence. But D, M and the global convergence detection parameter are additional parameters that affect PGA's performance for different problems. The overall result is some decrease in PGA's robustness.

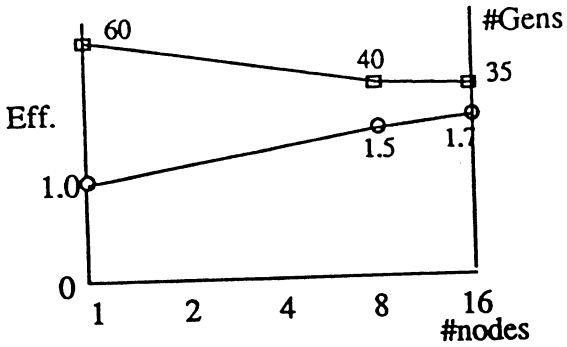

Figure 23. PGA efficiency and #generations for Test1.

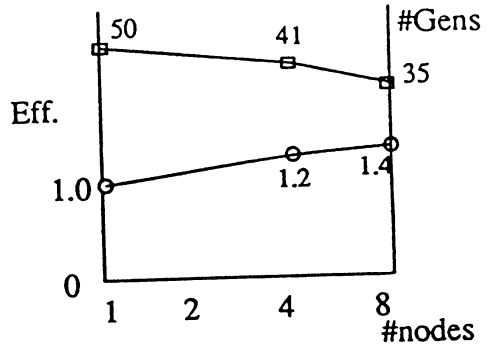

Figure 24. PGA efficiency and #generations for Test2.

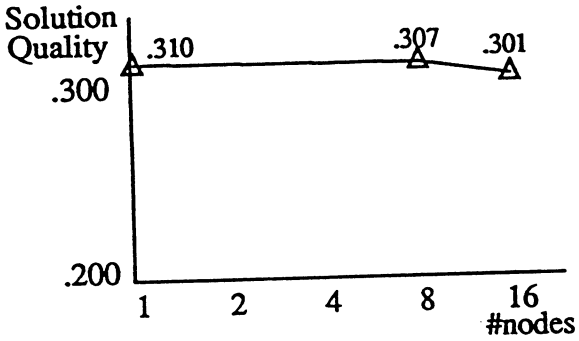

Figure 25. PGA solution for Test1.

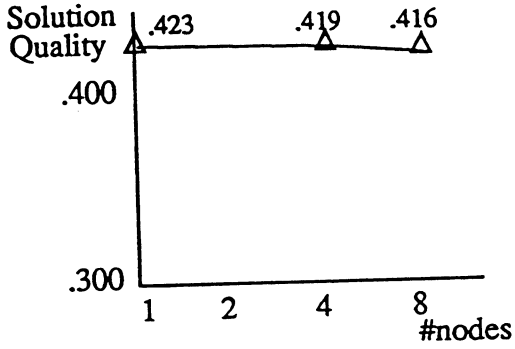

Figure 26. PGA solution for Test2.

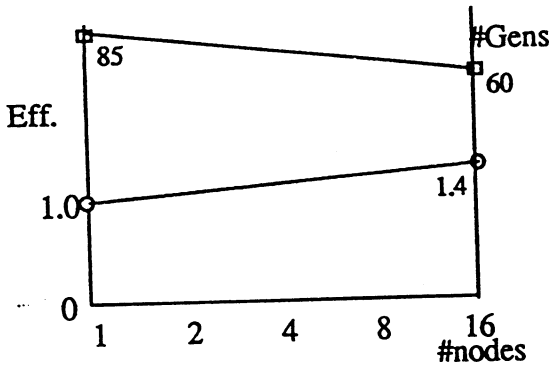

Figure 27. PGA efficiency and #generations for Test3.

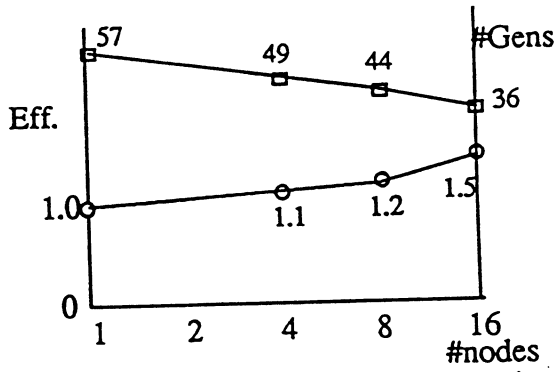

Figure 28. PGA efficiency and #generations for Test4.

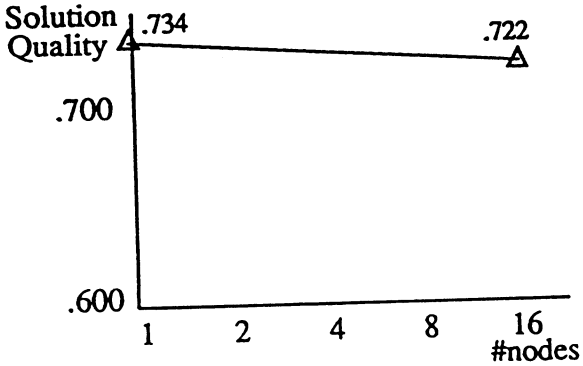

Figure 29. PGA solution for Test3.

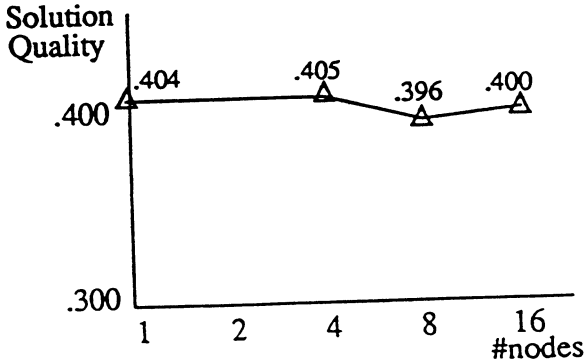

Figure 30. PGA solution for Test4.

6. Comparison of the Algorithms

The measures considered, in this section, for comparing the performances of the three parallel data allocation algorithms are solution quality, bias, execution time, scalability, robustness, and memory space requirements.

Figures 31-34 provide compact pictures of the solution quality of the parallel physical algorithms for the four test cases. In addition, results of two fast bisection heuristics are included to give an indication of how good the solutions of the physical methods are. The two methods are recursive coordinate bisection (RCB) and recursive spectral bisection (RSB) [Simon 1991], which partition a computation graph into subgraphs (data subsets). RSB is regarded as a representative of good quality heuristics [Pothen et al. 1990]. For a consistent comparison, we added a second step to both algorithms. A simulated annealing algorithm is added to allocate the subgraphs to hypercube nodes such that OF_{typ} is minimized. It is clear from the figures that the physical algorithms produce good sub-optimal solutions which outperform the bisection methods. The only exception is the two-dimensional GRID-P case where RSB and PNN seem comparable. It should be noted, however, that the discrepancy between the uses of OF_{appr} to guide the operation of the three algorithms and OF_{typ} to evaluate their solutions presents an impediment to the full realization of the capabilities of the physical optimization methods. Comparing the three algorithms, it is clear that PGA consistently produces the best solutions, PSA produces the second best, and PNN's solutions come last. For example, for Test1 and $N_H = 16$, PSA's solution was 11% better than PNN's, and PGA's solution was 6% better than PSA's and 18% better than PNN's. This finding is consistent with what is observed for the sequential algorithms ($N_H = 1$). It is interesting to note that the differences in the solution qualities do not undergo any significant changes with the grain size, i.e. with different N_H values. Further, since the solutions of the parallel algorithms are consistent with those of their sequential counterparts, our earlier conclusion about the applicability of this class of algorithms can be reiterated: the parallel physical algorithms do not exhibit a bias towards particular problem topologies [Mansour and Fox 1991b].

The execution times, in seconds, of the three parallel algorithms are summarized in Figures 35-38. It is clear that PGA is the slowest and PNN is the fastest. For example, for Test1 and $N_H = 16$, PSA was 2.4 times slower than PNN, and PGA was 2.8 times slower than PSA. This result holds for different degrees of parallelism, including the sequential case. However, the gaps separating the time curves shrink with higher degrees of parallelism; PGA's execution time decreases the fastest as N_H increases, followed by PSA. For example, for Test3, the sequential GA time is 29 times that of the sequential NN. But, for 16 nodes, the ratio decreases to only 5. This follows from the result that PGA yields the best efficiencies and PNN the smallest.

As discussed above, all three parallel algorithms are somewhat less robust than their sequential counterparts due to additional design parameters. But, the parallel algorithms no longer exhibit the

difference in robustness observed for the sequential algorithms [Mansour and Fox 1991c]; the levels of insensitivity to problem and design parameters for the three parallel algorithms are comparable.

There is a significant difference in the memory space requirements between the algorithms. In PGA, a population of structures evolve, and, thus, information is needed in every node (subpopulation) about the whole problem, whereas in PSA and PNN only the local subproblem is considered in a node. For large problems, PGA requires large memory space, which might not be possible depending on available technology. One way to alleviate this restriction is to add a preprocessing graph contraction step to PGA, where the problem, and consequently the individuals in the population, are reduced in size by a certain factor [Mansour and Fox 1991c]. Generally, this could lead to some decrease in solution quality as the contraction factor increases. But, it will also lead to significant decrease in execution time.

PSA and PNN are scalable. The quality of their solutions remains almost constant provided that the grain size does not become too small. Efficiency of both, PSA and PNN, decreases with larger hypercubes, more quickly for PNN. Decreasing efficiency implies smaller decreases in execution time as the size of the hypercube increases. If the memory space restriction is circumvented, as suggested above, PGA would enjoy better scalability than PSA and PNN. With larger hypercubes, its execution time decreases faster. It yields good solutions, even for the smallest deme size (PGA's grain size) of two. Furthermore, larger hypercubes offer the opportunity to increase the total population size and the number of demes for PGA, which is likely to produce yet better solutions.

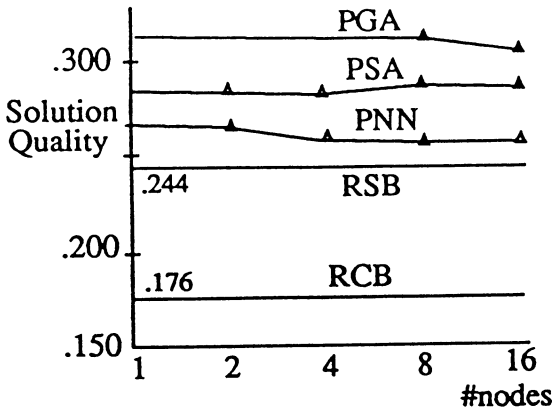

Figure 31. Comparison of solutions for Test1.

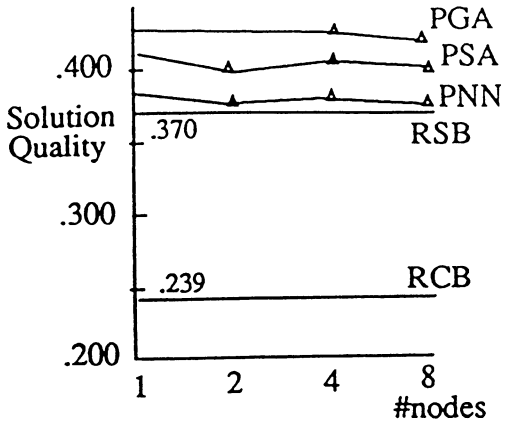

Figure 32. Comparison of solutions for Test2.

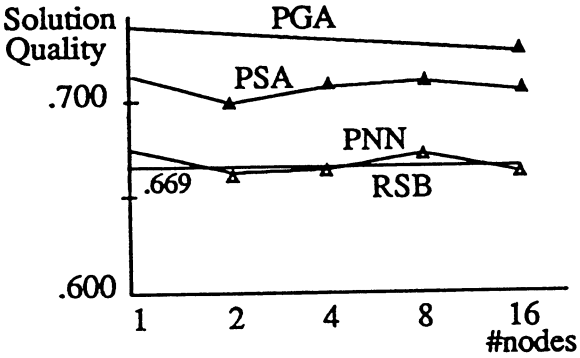

Figure 33. Comparison of solutions for Test3.

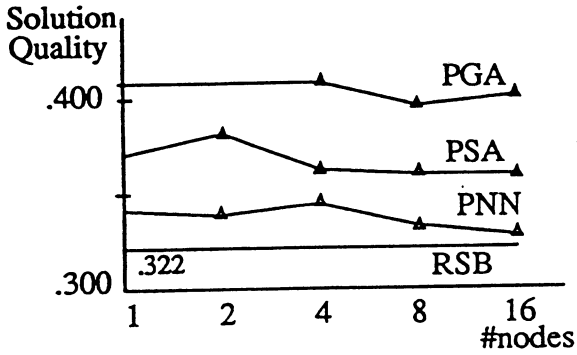

Figure 34. Comparison of solutions for Test4.

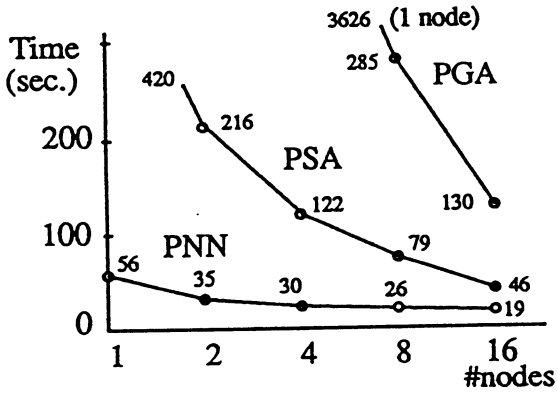

Figure 35. Execution time for Test1.

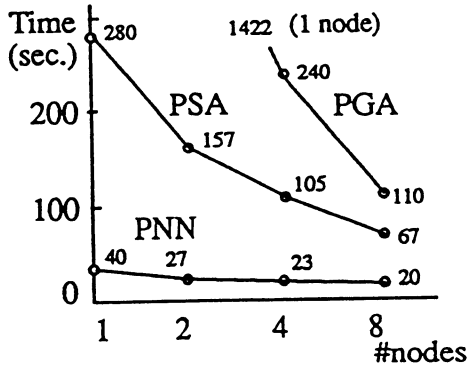

Figure 36. Execution time for Test2.

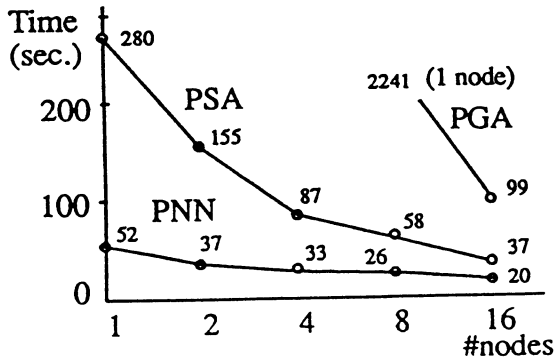

Figure 37. Execution time for Test3.

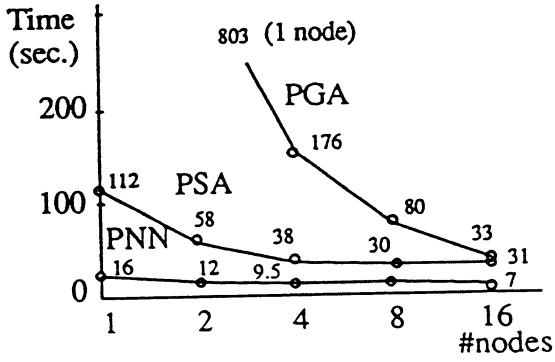

Figure 38. Execution time for Test4.

7. Conclusions and Further Research

Three parallel algorithms based on simulated annealing, neural networks and genetic algorithms have been presented and applied to the problem of allocating data to hypercube nodes. The individual properties of the parallel algorithms have been investigated and their performances have been compared for four test cases with different features.

PSA and PNN deviate from their sequential counterparts. They are based on concurrent execution of the sequential algorithms in all nodes. Communication schemes adapted to the characteristics of the algorithms constitute salient components of the algorithms and are important for limiting the communication overhead. In addition, PNN includes spin reallocation after bisection steps for reducing the cost of global communication. PGA, on the other hand, is based on distributed population models and the shifting balance theory of evolution, which lends itself to embarrassingly parallel implementation. The three parallel algorithms exhibited diverse properties which make them suitable for different applications.

The data allocations produced by the three parallel algorithms are good sub-optimal solutions and do not show a bias towards particular problem configurations. The inconsistencies allowed in PSA do not lead to degradation in the quality of the solutions. The same conclusion holds for PNN, as long as the granularity is not too small. PGA's solutions, also, are consistent with those of the sequential GA. PGA produces the best results, followed by PSA and then PNN. The adaptive communication frequencies in PSA and PNN provide adequate schemes for limiting the decrease in efficiency, for larger hypercubes, whilst maintaining good solution qualities. PGA enjoys superlinear efficiencies, with respect to classic sequential GAs. However, PNN remains the fastest and PGA the slowest. Interestingly, the gaps between the execution times of the three algorithms decrease with larger hypercubes. All three parallel physical algorithms are somewhat less robust than the sequential algorithms. However, the three share a reasonable and comparable level of robustness. Further, the three algorithms are scalable. However, PGA requires additional preprocessing to circumvent memory-space constraints.

The test cases which have been considered are rather small. However, we are confident that the results, reported in this paper, extend to larger problems. For such problems, a preprocessing graph contraction step has been found useful for reducing execution time and memory requirements [Mansour and Fox 1991c]. Based on this finding, further work is underway to apply the physical algorithms to large problems.

Acknowledgment

This work was supported by the ASAS agency of the U.S. army and the Center for Research on Parallel Computation. It was carried out using the nCUBEs at Sandia Labs and Caltech. We wish to thank Horst Simon for providing his RSB code, Joel Saltz for providing the FEM-W data, and Yeh-Ching Chung for FEM-2.

References

Berger, M., and Bokhari, S. 1987. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, C-36, 5 (May), 570-580.

Chrisochoides, N.P., Houstis, E.N., and Houstis C.E. 1991. Geometry based mapping strategies for PDE computations. In *Int. Conf. on Supercomputing*, 115-127, ACM Press.

Eglese, R. W. 1990. Simulated annealing: a tool for operational research. *Euro. J. Operational Research*, 46, 271-281.

Ercal, F. 1988. Heuristic Approaches to task allocation for parallel computing. Ph.D. thesis, Ohio State University.

Flower, J., Otto, S., and Salama M. 1987. A preprocessor for finite element problems. In *Proc. Symp. Parallel Computations and their Impact on Mechanics,* ASME Winter Meeting (Dec.).

Fox, G.C. 1988. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In *Numerical Algorithms for Modern Parallel Computers,* ed. M. Schultz, Springer-Verlag, Berlin.

Fox, G.C., and Furmanski, W. 1988. Load balancing loosely synchronous problems with a neural network. In *Proc 3rd Conf. Hypercube Concurrent Computers, and Applications*, 241-278.

Fox, G.C., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. 1988. *Solving Problems on Concurrent Processors*. Prentice Hall.

Fox, G.C. 1990. Physical computation. In *Proc. Int. Conf. Parallel Computing: Achievements, Problems and Prospects*, Italy (June).

Fox, G.C., Hiranandani S., Kennedy K., Koelbel, C., Kremer, U., Tseng, C., and Wu, M-Y. 1990. Fortran D language specification. Syracuse University, NPAC, SCCS-42.

Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Greening, D.R. 1990. Parallel simulated annealing techniques. *Physica D* 42, 293-306.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press.

Hopfield, J.J., and Tank D.W. 1986. Computing with neural circuits: a model. *Science* 233, 625-639.

Houstis, E.N., Rice, J.R., Chrisochoides, N.P., Karathonases, H.C., Papachiou, P.N., Samartzis, M.K., Vavalis, E.A. , Wang, K.Y., and Weerawarana, S. 1990. //ELLPACK: A numerical sim-

ulation programming environment for parallel MIMD machines. In *Int. Conf. on Supercomputing*, 3-23, ACM Press.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. 1983. Optimization by simulated annealing. *Science* 220, 671-680.

Mansour, N., and Fox, G.C. 1991a. A hybrid genetic algorithm for task allocation. *Proc. Int. Conf. Genetic Algorithms* (July), 466-473.

Mansour, N., and Fox, G.C. 1991b. Physical optimization methods for allocating data to multicomputer nodes. *Proc. System Design Synthesis Technology Workshop* (NSWC, Sept.).

Mansour, N., and Fox, G.C. 1991c. A comparison of load balancing methods for parallel computations. Syracuse University, SU-CIS-91-47, submitted for publication.

Mansour, N., and Fox, G.C. 1991d. Parallel genetic algorithms with application to load balancing parallel computations. Syracuse University, SU-CIS-91-48, submitted for publication.

Nolting, S. 1991. Nonlinear adaptive finite element systems on distributed memory computers. *European Distributed Memory Computing Conference* (April), 283-293.

Pothen, A., Simon, H., and Liou, K-P. 1990. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11, 3 (July), 430-452.

Simon, H. 1991. Partitioning of unstructured mesh problems for parallel processing. *Proc. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Permagon Press.

Walker, D. 1990. Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code. *Concurrency Practice and Experience* (Dec.), 257-288.

Williams, R.D. 1991. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency Practice and Experience*, 3(5), 457-481.

Wright, S. 1977. *Evolution and the Genetics of Populations*. Vol. 3, Univ. of Chicago Press.