9-1991

A Comparison of Load Balancing Algorithms for Parallel Computations

N. Mansouri
Syracuse University, Department of Engineering and Computer Science, namansou@ecs.syr.edu

Geoffrey C. Fox
Syracuse University

SU-CIS-91-47

A Comparison of Load Balancing Algorithms for Parallel Computations

Nashat Mansour and Geoffrey C. Fox

September, 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

A Comparison of Load Balancing Algorithms for Parallel Computations

Nashat Mansour
Computer & Information Sc.
Syracuse University

Geoffrey C. Fox
Northeast Parallel Architectures Ctr.
Syracuse University

Abstract

Three physical optimization methods are considered in this paper for load balancing parallel computations. These are simulated annealing, genetic algorithms, and neural networks. Some design choices and the inclusion of additional steps lead to new versions of the algorithms with different solution qualities and execution times. The performances of these versions are critically evaluated and compared for test cases with different topologies and sizes. Orthogonal recursive coordinate bisection is also included in the comparison as a typical simple deterministic method. Simulation results show that the algorithms have diverse properties. Hence, different algorithms can be applied to different problems and requirements. For example, the annealing and genetic algorithms produce better solutions and do not show a bias towards particular problem structures. But, they are slower than the neural network and recursive bisection. Preprocessing graph contraction is one of the additional steps suggested for the physical methods. It produces a significant reduction in execution time, which is necessary for their applicability to large-scale problems.

Keywords: Data allocation, data partitioning, genetic algorithms, load balancing, loosely synchronous algorithms, neural networks, physical optimization methods, recursive bisection, simulated annealing, task allocation.

1. Introduction

Efficient utilization of the computational power of distributed-memory parallel computers is highly dependent on how the composite calculation-communication workload is distributed among the processors. An optimal load distribution depends on the application, the algorithm and the machine. In this work we concentrate on loosely synchronous algorithms [5] for MIMD distributed-memory parallel computers, henceforth referred to as multicomputers, such as hypercubes. It should be emphasized, however, that the methods described in this paper are general and not restricted to loosely synchronous computations. The programming model is assumed to be single-program-multiple-data, where parallelism is achieved by partitioning the underlying data set of the problem and allocating the disjoint subproblems to the processors. A subproblem determines the calculation load of a processor for a given algorithm. Further, the distribution of data usually necessitates interprocessor communication to exchange information. Optimal data distribution corresponds to balancing the calculation load among the processors and minimizing/balancing communication. Both requirements about calculation and communication constitute what is henceforth referred to as the load balancing problem, which is defined more clearly in Section 2.

The load balancing problem is NP-complete, and several heuristic methods have been proposed for finding good sub-optimal solutions. Deterministic heuristics have been popular because of their speed and predictable execution time. Examples are recursive bisection, mincut-based heuristics, labeling algorithms, and scattered decomposition [1, 2, 3, 8, 15, 16, 17]. Such methods seem to favor certain problem configurations. On the other hand, physical computation has been advocated for describing, simulating, and solving intractable problems [7]. It uses methods borrowed or derived from physical and natural sciences. These methods usually include probabilistic components. They yield near-optimal or good sub-optimal solutions and have more general applicability than deterministic heuristics. However, they are usually slower. Three physical optimization methods have been adapted to the load balancing problem. Simulated annealing [12], from statistical physics, has been applied to several cases [4, 5], and continues to be of interest because of the design choices it involves. Neural networks [11], from neurobiology and using physics, have been proposed and applied to illustrative load balancing examples [6]. Recently, a genetic algorithm [9, 10], from population biology, has been adapted to load balancing and has been applied to a limited number of test cases [13].

In this work, the three physical optimization algorithms are reviewed and new versions are described. The new versions are based on the following modifications: making some parameters adaptive, modifying some steps to reduce execution time or improve robustness, employing different objective functions, adding postprocessing tuning steps, using hybrid techniques, and including preprocessing graph contraction. The goal of these modifications is to explore the applicability of the resultant versions to realistic examples and their suitability for problems with different requirements.

In this paper, the performances of the different versions of the physical optimization algorithms are evaluated and compared for test cases of diverse properties. To broaden the scope of evaluation, the results of a recursive bisection method for the most realistic test case are included. The paper is organized as follows. The next section defines the model of computation and the relevant objective functions. Sections 3 through 7 describe the four methods and their modifications. Section 8 presents experimental results. Section 9 includes evaluations and comparisons. Section 10 presents conclusions.

2. Objective Functions

The computational model considered in this paper is that of loosely synchronous parallel algorithms, which are applicable to many problems in science and engineering [5]. In this model, the processors of the multicomputer iterate through a sequence of computation-communication steps. They independently execute the same code on different subproblems. But in every iteration, the processors communicate boundary information before proceeding with further computations. The total parallel execution time is, therefore, determined by the slowest processor and is given by

$$OF\_EXACT = \max_p \{ W(p) + \Sigma_q C(p,q) \}, \quad (1)$$

where $W(p)$ is the calculation load of processor p and $C(p,q)$ is its communication requirements with processor q. Equation (1) is the exact objective function that is required to be minimized for load balancing loosely synchronous computations. This function is not smooth and is computationally expensive. To bypass these drawbacks, an approximate objective function can be used, which includes the sum of the mean square deviation of the calculation loads of the processors and the total amount of communication costs. It is given by

$$OF\_APPR = \gamma \sum_p W^2(p) + \beta R \sum_p \sum_q C(p,q), \quad (2)$$

where R is a machine-dependent time ratio of one-word of communication to one floating-point calculation; $C(p,q)$ is the Hamming distance between processors p and q multiplied by the number of data elements to be exchanged; γ and β are scaling factors expressing the importance of the calculation term and the communication term, respectively. *OF_APPR* is smoother than *OF_EXACT* and is, therefore, more suitable for optimization methods. More importantly, the change in *OF_APPR* due to a change in data distribution is much less expensive to compute than the corresponding change in *OF_EXACT* [13]. In this work, the minimization of *OF_APPR* is the goal of most of the versions of the physical optimization methods. However, all solutions produced, including those of the recursive bisection method, are evaluated according to *OF_EXACT*. The discrepancy between one expression guiding the operation of the methods and another evaluating their results has motivated the introduction of versions of the algorithms that utilize both expressions, as discussed below.

3. Genetic Algorithms

In genetic algorithms a population of possible solutions (individuals) evolve over successive generations, starting from random data distributions. In every generation, individuals are selected for reproduction according to their fitness, genetic operators are applied to the selected mates, and offsprings replace their parents. In this process, fitness is gradually increased and near-optimal solutions evolve by the propagation and the combination of high-performance fit building blocks [9, 10].

An outline of a genetic algorithm (GA) for load balancing [13] is given in Figure 1. The encoding of a data distribution instance (individual) is given by a string of integers, where an integer refers to a processor and its position in the string represents the allocated data element. The fitness of an individual is the reciprocal of the value of the objective function. The reproduction scheme is based on ranking, where the individuals are sorted by their fitness values and are assigned a number of reproduction trials according to a predetermined scale of values. After ranking, pairs of parents are randomly chosen from the list of reproduction trials. The genetic operators employed in the GA are two-point crossover, two-phase mutation, and inversion. The operator rates are made adaptive to the variation in the average degree of clustering of the data elements in the individuals, in a way to counteract premature convergence. The genetic algorithm is hybridized by including a problem-specific hill-climbing procedure that directs the genetic search to the more profitable peaks in the adaptive topography. The procedure allows the transfer of boundary data elements from overloaded to underloaded processors in a data distribution instance. That is, hill-climbing only allows the incremental changes that increase the fitness of an individual. In the later phase of the evolution, the population loses diversity and approaches convergence. In this phase, standard random mutation is replaced with boundary mutation, inversion is abandoned, and, more importantly, the weights in *OF_APPR* are gradually varied for fine-tuning the converging structures. Further, some copies of identical individuals are gradually removed, thus reducing the population size. This phase is referred to as the tuning phase. The complexity of GA is of the order of (DEG*P*POP*GEN), where DEG is the average vertex degree in the problem's computation graph; P is the size of this graph (i.e. problem size); POP is the population size; GEN is the number of generations. In our implementation, POP has been chosen to be 0.2*P to 0.6*P, with the larger population corresponding to the larger multicomputer.

Two versions of GA are explored in this paper. Version GA1, as described previously [13], employs OF_APPR in both fitness evaluation and hill-climbing. It also uses problem-dependent user-defined parameters to invoke the tuning stage. A second version, GA2, uses OF_EXACT for fitness and OF_APPR for hill-climbing. It also includes nearly automatic invocation, based on OF_EXACT, of the last stage which increases robustness at the expense of a small fraction of solution quality and/or execution time for some problems.

```
Random generation of initial population, size POP;
Evaluate fitness of individuals;
repeat
        Set γ , β , operator rates;
        Rank individuals & allocate reproduction trials;
        for i = 1 to POP step 2
            Randomly select 2 parents from list of reproduction trials;
            Apply crossover, mutation, inversion;
            Hill-climbing by offsprings;
        endfor
        Evaluate fitness of offsprings;
        Preserve the fittest-so-far;
until (convergence)
Solution = Fittest.
```

Figure 1: A genetic algorithm for load balancing.

4. Simulated Annealing Algorithms

An outline of a Simulated Annealing Algorithm (SAA) for load balancing is given in Figure 2. The initial data distribution is random and is associated with a high temperature and a high energy state. The energy of the configuration is given by the objective function used and the goal of SAA is to minimize this energy. The configuration is cooled down according to a schedule. At each temperature, a number of perturbations to the configuration are performed until thermal equilibrium. A perturbation is accomplished by a random transfer of a randomly chosen data element. The perturbation that leads to lower or identical objective function value (downhill move), is always accepted; that which increases it (uphill move) is allowed only with a temperature-dependent probability. Thermal equilibrium is reached when a predetermined maximum number of perturbations has been attempted or accepted. The maximum number of attempts allowed is the larger of the size of the multicomputer and the average degree in the problem's computation graph; the maximum number of accepted moves is the smaller of the two. These choices secure a sufficient number of moves for thermal equilibrium while not spending too much time at high temperatures. The cooling schedule determines the next temperature as a fraction, k, of the present one. In this implementation, this fraction varies within the range 0.91 to 0.99 in such a way to speed up cooling whenever possible and to slow it down whenever necessary. k increases if the number of accepted moves decreases, and vice versa. The initial high temperature is that which makes the probability of accepting uphill moves equal to 0.8. The freezing temperature corresponds to a small probability of 2^{-30} for the minimum possible increase in the objective function. The complexity of the SAA algorithm is of the order of (DEG*max{DEG,N}*P*A), where A is the number of annealing steps; N is the number of multicomputer processors; DEG and P are as defined before. For adaptive schedules, A is problem-dependent, although of the order of *log*(initial T / freezing T).

Two versions of SAA are explored below. The first version, SA1, uses OF_APPR for the energy until freezing. The second version, SA2, is identical to SA1 until the number of accepted perturbations is small. Then, at low temperatures, OF_EXACT is used for the energy. Also, random perturbation is replaced by neighbor perturbation. That is, only the reallocation of boundary data elements to neighboring processors is attempted, so that time is not wasted in random reallocations. The computation of OF_EXACT makes SA2 much slower than SA1.

```
Determine initial temp. T(0);
Initial configuration = Random data allocation;
/* Annealing - SA1 and SA2 */
while (T>THRESHOLD1 and #accepts>THRESHOLD2) do
    T = T(i);
    repeat
        Perturb(configuration);
        E = OF_APPR;
        if (dE <= 0) then Accept; Update configuration;
        else rnd = random number (0,1);
              if (rnd < exp(-dE/T) then Accept and Update;
              else Reject;
    until (Equilibrium);
    Determine k;
    T(i+1) = k * T(i); /* cooling schedule */
end-while
/* Annealing at low temperatures - SA2 only */
repeat
     Anneal with Neighbor-Perturb(configuration) & OF_EXACT for E;
until (freezing or convergence)
```

Figure 2: A simulated annealing algorithm for load balancing.

5. Bold Neural Network Algorithms

The Bold Neural Network (BNN) [6] is based on Hopfield and Tank's network [11]. It is built from $P*lg_2N$ neurons, where P and N are the sizes of the problem domain and the multicomputer, respectively. Each neuron has a neural variable, $v(e,i,t) = 0$ or 1, associated with it. The neuron's label (e , i) corresponds to data element e and bit i of the node label in the multicomputer (hypercube). Note that the label of a hypercube node is given by $\Sigma (v(i) \cdot 2^i)$, where the summation is over i = 0 to (lg_2N-1). The neural variables represent the amount of local information about the solution at time t. The network starts with random neural values and converges to the fixed point, where the solution is given by the global map of the converged neural values in the whole network. The BNN repeats this procedure lg_2N times, each time determining one bit i in the network and, hence, the

subcube to which each data element belongs. That is, each iteration corresponds to a bisection of the subproblems from the preceding iteration. After the last iteration, the problem will be partitioned into N subproblems. It is noteworthy that the neural representation used for BNN provides a natural way for removing ambiguities, such as placing the same element in two subdomains. Hence, it dispenses with the redundant synaptic connections that would have been required to enforce the problem constraints.

The fixed point of the network is associated with the minimum of the energy, OF_APPR. To determine the network equations, the neural variables are replaced by spin variables, s(e,i,t) = -1 or +1, and the energy expression is rewritten in terms of spin variables. Then, a standard mean field approximation technique from physics is used to derive the BNN formula [6]

$$s(e,i,t+1) = tanh\ \{-\alpha\ s(e,i,t) + \beta \sum_{s'} C(s,s') - \frac{\gamma}{D_{i-1}} \sum_{e'} s(e',i,t)\}, \qquad (3)$$

where α is a scaling factor; D is the size of the current subproblem (to be further bisected) to which data element e belongs; e' in the third term refers to elements only in the current subproblem. The BNN formula can be interpreted in the light of magnetic properties of materials. At a critical temperature (Curie point), spontaneous magnetization domains of nearly-equal number of spins are formed in solids in such a way that spins within each domain are lined up, but have opposite direction to those in the other domain. In equation (3), the second term can be interpreted as the ferromagnetic interaction that aligns the neighboring spins. The third term can be interpreted as the long-range paramagnetic force responsible for the global up/down spin balance. The first term is inserted in the BNN equation as a noise term that tries to flip the current spin and, thus, helps the system avoid local minima. The scaling factors have the following effects. α determines how stable a solution can be after a number of iterations, β determines the speed of the formation of the domain structure, and γ controls the spin balance in the configuration. In our implementation, $\alpha=\beta=2$ and γ is gradually increased from 2 to 20 for every bisection level. β plays the role of inverse temperature, and its value is chosen to ensure that the system is near the critical point, as explained before. With these values, it has been shown that the number of iterations required for the network convergence is a small number times the square root of the problem size [6].

The BNN algorithm is summarized in Figure 3. Its complexity is of the order of (DEG*(P **3/ 2)*lg_2N). This algorithm is henceforth called NN1. NN2 is a second version which includes a local optimization step for adjusting the boundaries of the data distribution produced by NN1. In this step, boundary data elements are transferred to neighboring processors only if OF_EXACT decreases. In most cases, the execution time of NN1 is much smaller than that of NN2. However, NN2 can, sometimes, improve the quality of NN1's solutions significantly and is included here for comparison with the genetic and annealing algorithms.

```
for i = 0 to (lg2N -1) do
    Generate random spins s(e,i,t) over whole domain;
    repeat
        Determine γ;
        for all spins in the domain
            pick a spin randomly;
            Compute s(e,i,t+1); /* equation (3) */
        end-for
    until (convergence)
    Determine bit i in the neurons;
end-for
```

Figure 3: Bold neural network algorithm for load balancing.

6. Recursive Bisection

The bisection method considered here is orthogonal recursive coordinate bisection (ORCB) [16]. It is simpler than the physical methods and is included in the comparison to give some indication of their performance.

ORCB's operation is deterministic and not guided by an objective function. Instead, it utilizes the physical coordinates of the data elements to recursively bisect a problem into two subproblems with equal sizes. In each bisection step, a direction (x or y) is chosen as a separator and directions alternate in successive steps. Data elements are sorted by coordinates in the selected separator direction and each half of the elements is assigned to a subproblem. The recursive process continues until the number of subproblems equals the number of processors in the multicomputer. Then, the resultant subproblems are mapped to the processors. For mapping, we use a simulated annealing algorithm that minimizes OF_EXACT. The complexity of the ORCB is of the order of $P*lg_2N(lg_2P - lg_2N)$.

7. Versions Involving Hybrids and Preprocessing

Two versions of the physical optimization methods are described in this section. Both aim for reducing the execution time, especially of GA and SA. The objective of studying these versions is to further explore the practical applicability of these load balancing methods.

The first version is based on two observations. The first observation is that methods such as NN1 or ORCB yield solutions of lower quality than those of SA and GA, but are considerably faster. The second observation is that SA and GA take longer time to evolve solutions of the same quality as those of NN1. Therefore, hybrid methods which start with NN1, or other methods with similar speed, and continue with GA or SA can be faster than pure GA or SA. In this work NN1-GA and NN1-SA are explored. SA picks up at a low temperature which gives uphill moves with probability

15%. GA creates its initial population by randomizing the boundary regions of the solution provided by NN1.

The second version utilizes a graph contraction step prior to load balancing. In this preprocessing step, edges in the problem's computation graph are contracted and vertices are merged together to form a multigraph whose super-vertices are weighted by the sum of the merged data elements and whose edges are weighted by the sum of the communication requirements. The level of contraction can be determined such that the size of the resultant multigraph is K times the size of the multicomputer. Following the allocation of the contracted multigraph to the processors, the original problem graph can be restored and more SA or GA iterations can be carried out for improving the quality of the solution. Contraction can also speed up BNN in the same way, although K cannot be small as will be seen below. Methods involving contraction are henceforth referred to as CONT-M, where M is a load balancing method.

Graph contraction, with parameter K, leads to a great reduction in the search space of data distribution from N**P to N**(K*N), where K*N is the size of the contracted graph and can be considerably smaller than the original size, P. The assignment of the contracted graph to the processors becomes a fast step. Subsequent SA iterations on the restored original graph, therefore, start with a reasonable solution at a low temperature. For GA's, the smaller contracted graph allows a smaller population size, POP, which can be kept the same in the subsequent generations after the restoration of the original graph.

Graph contraction itself is not of concern here. It can be done by any fast procedure. For example, at each contraction level, edges can be selected randomly and the two vertices at both ends are then merged. In our simulations, NN1 is used for contraction.

8. Simulation Results

The performance of the load balancing algorithms is presented in this section. Test cases of different geometric shapes, dimensions, sizes and granularities are employed for hypercube multicomputers. The performance measures are the hypercube's efficiency and execution time. Efficiency is defined as the ratio of the sequential time to the product of the parallel time, *OF_EXACT*, and the hypercube size, N.

The test cases used are shown in Figure 4. GRID1 and GRID2 yield computation graphs with a typical vertex degree of 4. GRID1 is uniform with irregular geometry. GRID2 has large variation in the density of its points. FEM1 and FEM2 are finite-element meshes with vertex degrees ranging from 8 to 12. FEM1 is 2-dimensional and nonuniform. FEM2 is 3-dimensional. FEM3 is the most realistic of the five examples; it represents a part of an airplane. It is 3-dimensional with typical vertex degree 16. We have concentrated on FEM3 and GRID1 because of their interesting and distinct properties.

Tables 1 through 5 show results for the algorithms described above. Each result is the average of ten runs. Each entry in the best efficiency column is the best result obtained from all runs carried out. The time given is for a SPARC 1+ workstation. For clarity, the efficiency figures are shown as percentages of the best efficiency which itself is kept as an absolute number. Since the optimum is not known, the best efficiency column serves as an indicator of how good the individual average figures are. To further illustrate the performance of the physical methods and to broaden the scope of comparison, Table 6 presents results for other examples. Each result in Table 6 is the average of five runs. ORCB's results are given only for FEM3 and GRID1 whose coordinates were available. The ORCB time given covers only the partitioning step. The time for the second step is not included because the annealing algorithm was optimized for mapping quality and not for time.

9. Discussion

This section starts with a discussion of the individual Tables, 1 to 6, leading into overall evaluations of the results. The measures considered for assessing and comparing the performance of the algorithms are solution quality (i.e. hypercube efficiency) and execution time. In addition, bias, sensitivity to design parameters, and parallelizability are qualitatively discussed in this section.

Table 1 shows the results of the versions of the physical methods that use the approximate objective function, OF_APPR. Table 2 shows the results of the versions that use OF_EXACT explicitly in their second phase (SA2 and NN2) or only partially (GA2). The results of ORCB are also included. From the two tables, the following observations can be made. When OF_APPR only is used, GA1 yields the best solutions, but at a high cost in terms of execution time. For GRID1, all the solutions are good. For the more interesting test case, FEM3, the quality of the solutions of the physical methods is substantially better than that of ORCB. Nevertheless, the pronounced difference between the solutions of GA1, SA1 and NN1 themselves warrants the exploration of SA2 and NN2. Table 2 shows a clear improvement in the efficiency values with a substantial increase in time for SA2 and NN2. GA2 is more favorable than GA1. Due to its higher sensitivity to design parameters, GA1 is not pursued further. For FEM3, SA2 yields the best efficiency and is the slowest. NN2 produces smaller efficiency values than those of SA2 and GA2, but is the fastest of the three. It should be emphasized here that the difference in solution quality and execution time of SA2 and GA2 for load balancing loosely-synchronous computations is mostly a result of the way OF_EXACT is used, explicitly by SA2 and only partially by GA2. This difference is not sufficient to evaluate the basic methods themselves for load balancing other classes of computations. However, the difference in the results given in Tables 1 and 2 highlights the importance of the formulation of the objective function for both solution quality and time.

The long time taken, especially by GA2 and SA2 in Table 2, justifies the exploration of the other versions, as in Tables 3,4 and 5. Table 3 gives the results for the hybrid methods, which start with

NN1 and continue with SA1, SA2, or GA2. It can be seen that starting from partial information about the solution leads to a reduction in time for SA1 and SA2 without degrading the final solution. The reduction in GA2's time is more pronounced, at a small price in terms of solution quality. In comparison with Table2, SA2 is still the slowest, with the best efficiency; the quality of NN1-GA2's solutions is only a little better than that of NN2 while still being slower. The ability of SA2 and GA2 to start from partial information about the solution is, nevertheless, an advantage over ab initio methods such as NN2 or ORCB. However, the times taken by NN1-SA2 and NN1-GA2 are still long and their decrease, as illustrated next, is of interest.

Tables 4 and 5 show results based on graph contraction whose time is assumed to be relatively small. These tables show a remarkable reduction in time for all algorithms, the reduction for GA2 and SA1 being the greatest. Table 4 includes results for different values of K, which is the ratio of the size of the contracted graph to the number of processors. The efficiency values for CONT-GA2, CONT-SA2 and CONT-SA1 are consistent with those in Table 2 and K can be as small as 2, leading to the greatest decrease in time without degrading the solution quality. However, for lower quality contraction, we suspect that K should be 4 or greater. For CONT-NN1 and CONT-NN2, K should be greater than or equal to 16 to maintain reasonable efficiency values. In this case, K should be greater because NN1 only allocates the contracted graph and does not share the flexibility of operating on the restored original graph. CONT-SA2 still yields the best efficiency values followed by GA2, but the time difference has become more pronounced in favor of CONT-GA2. Further, unlike the case of the uncontracted graph, CONT-SA1 is, for most cases, comparable to CONT-NN2 in terms of time and efficiency values.

Table 6 includes results for other examples, with K=16 for CONT-NN2 and K=2 for CONT-SA1, CONT-SA2 and CONT-GA2. These results clearly support the assessments made above, based on Tables 2, 4 and 5. In the remaining paragraphs, overall evaluations are presented.

The solutions evolved by GA2, SA2 and NN2 are good sub-optimal solutions. The results for the various topologies and sizes indicate that the annealing and genetic algorithms are not biased towards any particular problem topology. Recursive coordinate bisection, as expected, favors 2-dimensional uniform problems. The neural network seems to perform better for 2-dimensional uniform geometrical shapes, such as GRID1, than for 3-dimensional irregular structures, such as FEM2.

The annealing and genetic algorithms share the property of unpredictable convergence and, thus, execution time. Nevertheless, their execution times increase with the size of the problem and the multicomputer. The complexity expressions, mentioned above, serve only as indicators of the factors that determine a loose bound for the execution time. Although the bold neural network involves a probabilistic component, it has deterministic convergence.

The sensitivity of the physical methods to their design parameters is greatly reduced by making some important parameters adaptive. These parameters are the cooling schedule in SAA, the operator frequencies in GA, and γ in BNN. In this work, they vary within a range of commonly acceptable values. BNN is the most robust among the three methods. Interestingly, SAA and GA have analogous sensitivities to their design parameters. Both the cooling schedule for SAA and the frequencies of the genetic operators for GA control the convergence speed and have been made adaptive in our implementation. The number of attempted perturbations at a particular temperature for SAA and the population size in each generation for GA determine how many points in the solution space can be sampled. Both parameters have been empirically determined. However, GA is the least robust.

Parallel implementation, especially for GA and SAA, is important for their practical use. Current SAA parallelization techniques involve conflicts in the concurrent decisions made on different elements in the subproblems [17]. Conflicts are due to the presence of global terms in the computation of the change in objective function resulting from perturbations. The effect of these conflicts on the solution quality and execution time requires further studies. Parallel BNN would also involve conflicts because of the global paramagnetic term in equation (3). GA enjoys easier parallelizability based on distributed population models [14]. Parallelization can even reduce the sensitivity of GA to its design parameters.

10. Conclusions and Further Research

Versions of a Genetic Algorithm, a Simulated Annealing Algorithm and a Bold Neural Network for load balancing have been described. Their performances have been evaluated and compared for examples of various geometric shapes, dimensions and sizes. The solutions produced by these physical optimization methods are good sub-optimal solutions and are, for general problems, considerably better than those for Orthogonal Recursive Coordinate Bisection. However, the diverse properties of the methods and their versions suggest that the choice of one of them depends on the particular application. SA2 produces the best solution quality, followed by GA2, NN2, SA1, NN1 and ORCB in the order of decreasing quality. The order of decreasing execution time is the same with the order of NN2 and SA1 swapped in most cases.

The annealing and genetic algorithms have the ability of starting from partial information about the solution. This property results in a reduction in the overall execution time; the reduction is the biggest for GA. BNN and ORCB do not share this property. The applicability of the physical methods to realistic applications has been explored by adding a preprocessing graph contraction step. The results show that this step is advantageous for large problems because it leads to a significant reduction in execution time without sacrificing the solution quality. It has been found that SAA and GA make better use of graph contraction than does BNN. Concerning bias to particular problem structures, ORCB is clearly biased to 2-D uniform topologies; BNN exhibits some tendency to be

biased; SAA and GA do not show a bias. Concerning the robustness of the physical methods, BNN comes first, followed by SAA; GA is the least robust. Further research will explore the performances of parallel implementations, which are important for large-scale applications [14].

Acknowledgment

This work was supported by the Joint Tactical Fusion Program Office. We wish to thank Wojtek Furmanski for useful discussion about the neural net, Lainie Hyde for her comments on the first draft, Joel Saltz and Ravi Ponnusamy for providing the FEM3 data, and Yeh-Ching Chung for FEM1&2 data.

References

[1] M. Berger and S. Bokhari. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Trans. Comp.*, Vol C-36,No. 5, May 1987, 570-580.

[2] Y-C. Chung and S. Ranka. Mapping Finite Element Graphs on Hypercubes. Syracuse University, *SU-CIS-90-19*.

[3] F. Ercal. *Heuristic Approaches to Task Allocation For Parallel Computing*. Doctoral Dissertation, Ohio State University, 1988.

[4] J. Flower, S. Otto, and M. Salama. A Preprocessor for Finite Element Problems. *Caltech C3P #292c*, 1986.

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

[6] G. C. Fox and W. Furmanski. Load Balancing Loosely Synchronous Problems with a Neural Network. *Proc 3rd Conf. Hypercube Concurrent Computers, and Applications*, 1988, 241-278.

[7] G. C. Fox. Physical Computation. *Int. Conf. Parallel Computing: Achievements, Problems and Prospects*, Italy, June 1990.

[8] G. C. Fox. A Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube. *Caltech C3P #327b*, 1986.

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

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

[11] J. J. Hopfield and D. W. Tank. Computing with Neural Circuits: A Model. *Science* 233, 1986, 625-639.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science* 220, 1983, 671-680.

[13] N. Mansour and G. C. Fox. A Hybrid Genetic Algorithm for Task Allocation. *Int. Conf.*

Genetic Algorithms, July 1991, 466-473.

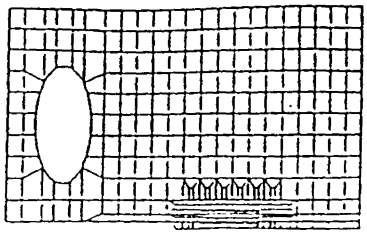
[14] N. Mansour and G.C. Fox. Parallel Physical Optimization Methods for Load Balancing Parallel Computations. In preparation.

[15] H. Simon. Partitioning of Unstructured Mesh Problems for Parallel Processing. *Proc. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Permagon Press, 1991.

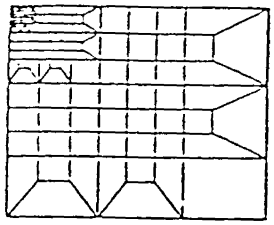
[16] D. Walker. Characterizing the Parallel Performance of a Large-scale, Particle-In-Cell Plasma Simulation Code. *Concurrency Practice and Experience*, December 1990, 257-288.

[17] R. D. Williams. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. Submitted to *Concurrency Practice and Experience*, 1990.

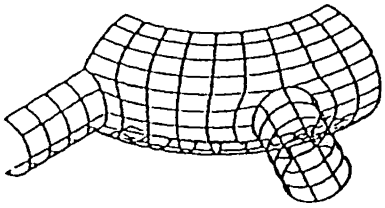
GRID1 (301-point)



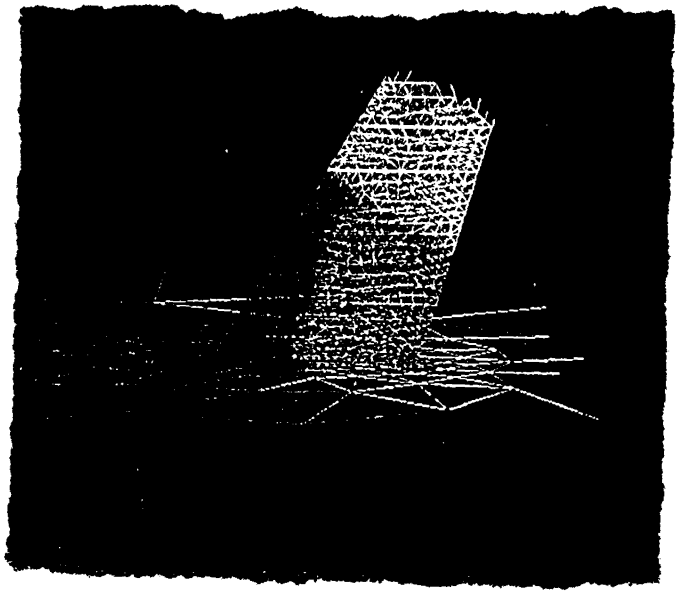
GRID2 (551-point)



FEM1 (160-point)



FEM2 (198-point)



FEM3 (545-point)

Figure 4: Test Cases.

Test case	Best Eff	GA1 %Eff	GA1 time	SA1 %Eff	SA1 time	NN1 %Eff	NN1 time	ORCB %Eff	ORCB time
FEM3 N = 16	0.338	89	40.2	80	5.26	77	0.58	52	0.06
GRID1 N = 8	0.85	95	2.11	89	0.66	88	0.17	88	0.03

Table 1: Results of versions using OF_APPR.

Test case	Best Eff	GA2 %Eff	GA2 time	SA2 %Eff	SA2 time	NN2 %Eff	NN2 time	ORCB %Eff	ORCB time
FEM3 N = 16	0.338	92	32.81	95	36.54	89	3.64	52	0.06
GRID1 N = 8	0.85	96	2.41	94	1.03	93	0.21	88	0.03

Table 2: Results of versions involving OF_EXACT.

Test case	Best Eff	NN1-GA2 %Eff	NN1-GA2 time	NN1-SA1 %Eff	NN1-SA1 time	NN1-SA2 %Eff	NN1-SA2 time
FEM3 N = 16	0.338	90	15.96	82	4.82	96	28.38
GRID1 N = 8	0.85	94	0.91	90	0.60	95	0.91

Table 3: Results of NN1-M hybrid versions.

	Best Eff	CONT-GA2 %Eff	CONT-GA2 time	CONT-SA1 %Eff	CONT-SA1 time	CONT-SA2 %Eff	CONT-SA2 time	CONT-NN1 %Eff	CONT-NN1 time	CONT-NN2 %Eff	CONT-NN2 time
K = 2	0.338	91	5.89	82	0.84	95	15.05	44	0.01	54	1.12
K = 4		92	9.17	85	1.10	97	17.63	47	0.02	63	1.44
K = 8		93	12.7	85	1.65	97	19.58	62	0.06	83	1.96
K = 16								73	0.18	86	2.21

Table 4: Results of CONT-M versions for FEM3 & N=16.

	Best Eff	CONT-GA2 %Eff	CONT-GA2 time	CONT-SA1 %Eff	CONT-SA1 time	CONT-SA2 %Eff	CONT-SA2 time	CONT-NN1 %Eff	CONT-NN1 time	CONT-NN2 %Eff	CONT-NN2 time
K = 2	0.85	93	0.18	91	0.21	96	0.36				
K = 16	0.85							74	0.03	92	0.10

Table 5: Results of CONT-M versions for GRID1 & N=8.

Test case	Best Eff	CONT-GA2 %Eff	CONT-GA2 time	CONT-SA1 %Eff	CONT-SA1 time	CONT-SA2 %Eff	CONT-SA2 time	CONT-NN1 %Eff	CONT-NN1 time	CONT-NN2 %Eff	CONT-NN2 time	ORCB %Eff	ORCB time
FEM3 N = 8	0.452	92	2.92	88	1.61	96	7.15	76	0.05	90	1.50	53	0.04
FEM1 N = 8	0.569	93	0.27	86	0.06	93	0.24	83	0.09	85	0.10		
FEM2 N = 8	0.578	92	0.26	86	0.06	97	0.43	81	0.05	89	0.10		
FEM2 N = 16	0.432	92	0.81	80	0.18	96	0.85	78	0.12	83	0.13		
GRID2 N = 16	0.787	92	1.42	77	0.28	94	1.94	73	0.20	85	0.30		

Table 6: Results of CONT-M versions, K=2 for GA & SA, K=16 for NN.