Allocating Data to Multicomputer Nodes by Physical Optimization Algorithms for Loosely Synchronous Computations

Nashat Mansour
Geoffrey C. Fox

CRPC-TR92262
April 1992

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

CONCURRENCY: PRACTICE AND EXPERIENCE, VOL. 4(0), 000–000 (MONTH 1992)

Allocating Data to Multicomputer Nodes by Physical Optimization Algorithms for Loosely Synchronous Computations

NASHAT MANSOUR
Computer & Information Science
Syracuse University, USA

GEOFFREY C. FOX
Northeast Parallel Architectures Center
Syracuse University, USA

SUMMARY

Three optimization methods derived from natural sciences are considered for allocating data to multicomputer nodes. These are simulated annealing, genetic algorithms and neural networks. A number of design choices and the addition of preprocessing and postprocessing steps lead to versions of the algorithms which differ in solution qualities and execution times. In this paper the performances of these versions are critically evaluated and compared for test cases with different features. The performance criteria are solution quality, execution time, robustness, bias and parallelizability. Experimental results show that the physical algorithms produce better solutions than those of recursive bisection methods and that they have diverse properties. Hence, different algorithms would be suitable for different applications. 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 algorithms. 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 problems.

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 nodes (processors). An optimal load distribution depends on the application, the algorithm and the machine. In this work we concentrate on loosely synchronous algorithms[1] for MIMD distributed-memory parallel computers, henceforth referred to as multicomputers, such as hypercubes. Loosely synchronous algorithms are applicable to many problems in science and engineering. The programming model associated with loosely synchronous computations is single-program-multiple-data, where parallelism is achieved by partitioning the underlying data set of a problem and allocating the disjoint subproblems to the processors. A subproblem determines the calculation load of a node for a given algorithm. Further, the distribution of data usually necessitates internode communication to exchange boundary information. The data allocation problem refers to mapping data elements to nodes such that the calculation load is as balanced as possible among the nodes and that internode communication is minimized and balanced.

The data allocation problem is an NP-complete optimization problem, and several heuristic methods have been proposed for finding good suboptimal solutions. Deterministic heuristics have been popular because of their speed and predictable execution time. Examples are recursive co-ordinate bisection, recursive graph bisection,

1040–3108/92/000000–00$00.00

Received 11 December 1991
Accepted 3 April 1992

~~execution time. Examples are recursive co-ordinate bisection, recursive graph bisection,~~ recursive spectral bisection, mincut-based heuristics, geometry-based mapping algorithms and scattered decomposition[2–9]. Such methods, however, seem to favor certain problem configurations. On the other hand, physical computation has been advocated for describing, simulating and solving intractable problems[10]. It uses methods borrowed or derived from physical and natural sciences. These methods usually include probabilistic components. They yield near-optimal or good suboptimal solutions and have more general applicability than deterministic heuristics; however, they are usually slower. Physical optimization algorithms have been applied to a number of NP-complete problems, such as quadratic assignment[11], graph partitioning[12,13], traveling salesperson problem[14] and VLSI placement[15]. Three physical algorithms have also been adapted to the data allocation problem. Simulated annealing[16], from statistical physics, has been applied to several cases[1,17], and continues to be of interest because of the design choices it involves. Neural networks[18], from neurobiology and using physics, have been proposed and applied to illustrative data allocation examples[19]. Recently, a genetic algorithm[20,21], from population biology, has been adapted to data allocation and has been applied to a limited number of test cases[22].

repeated line!

Although these heuristic data allocation algorithms have been known for a few years, there continues to exist a lack of comparative studies to assess their strengths, weaknesses and applicability. In this paper, the three physical allocation algorithms are reviewed and new versions are described. The 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. The performances of the different versions are evaluated, in this paper, and compared for test cases of diverse properties. We concentrate on loosely synchronous computations, explained in the next section, and hypercube multicomputers. It should be emphasized, however, that the physical methods are general and not restricted to loose synchronicity. To broaden the scope of evaluation, the results of two recursive bisection methods are included.

The paper is organized as follows. The following section defines the model of computation and the relevant objective functions. Sections 3 through 7 describe the algorithms and their modifications. Section 8 presents experimental results. Section 9 includes evaluations and comparisons, and Section 10 presents conclusions.

2. OBJECTIVE FUNCTIONS FOR DATA ALLOCATION

A problem can be represented by a computation graph with P vertices and edges. The vertices represent the data elements in the underlying data set, and the edges represent the data dependences determined by the ∧ algorithm. A hypercube multicomputer is also a graph with size N; the vertices are the nodes and the edges are the interconnections. Data allocation refers to mapping the vertices of the computation graph onto those of the hypercube such that the total execution time is minimized.

∧ application

In the loosely synchronous computational model, the hypercube nodes iterate through a sequence of computation–communication steps. They concurrently execute the same⅄program on different computation subgraphs. But in every iteration the nodes communicate boundary information before proceeding with further computations. The total parallel execution time, per iteration, is, then, determined by the slowest node, and is typically given by

⅄ application

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

where $W(n)$ is the calculation load of processor n and $C(n,m)$ is the cost of its communication with processor $m.W(n)$ is $\Sigma_v \omega.wt(v).t_{float}$, where $wt(v)$ is the computational weight of vertex v given by its degree in the computation graph, ω is the average number of arithmetic operations per graph edge performed by the ~~parallel~~⅄ algorithm between two communication operations, t_{float} is the time for a floating point operation, and the summation is over the vertices allocated to node n $C(n,m)$ is the product of the number of the common boundary vertices between n and m and the cost of communicating one word, t_{comm}. We assume that communication cost is determined by the Hamming distance and that message start-up time and link contention can be ignored. Although an accurate objective function depends on the specific parallel algorithm and the machine's architecture and software, we consider OF_{typ} a reasonable choice 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.

⅄ application

#

Equation (1) gives the objective function that is required to be minimized for optimal data allocation, but OF_{typ} is not smooth and is computationally expensive for the large number of incremental changes needed in the physical algorithms. To bypass these drawbacks, an approximate objective function can be used, which involves 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

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

where R is the machine-dependent ratio, t_{comm}/t_{float}; γ and β and are scaling factors expressing the importance of the calculation term and the communication term, respectively. OF_{appr} is smoother than OF_{typ} and is, therefore, more suitable for optimization methods. More importantly, it enjoys a locality property which makes it much less expensive to compute than OF_{typ}. Locality means that a change in OF_{appr} due to a change in data allocation is determined by the reallocated elements only[22]. Further, OF_{appr} is more robust than OF_{typ}. 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 methods, are evaluated using OF_{typ}. 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

Genetic algorithms are based on the theory of natural evolution. In artificial evolution, a species is a population of possible solutions, called individuals, which adapt over successive generations, starting with random structures. In every generation, individuals are selected for reproduction according to their fitness, genetic operators are applied to selected mates, and offsprings replace their parents. In this process, near-optimal structures evolve by the propagation and the combination of high-performance fit building blocks[20,21].

An outline of a hybrid genetic algorithm (GA) for data allocation is given in Figure 1[22]. It is designed to alleviate the problem of premature convergence and reduce the evolution time. It also makes use of problem specific knowledge to circumvent some computational costs and to reinforce favorable characteristics of the genetic search. An individual, i.e. a data allocation instance, is encoded as 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 objective function used. 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. Ranking allows the control of selection pressure and, thus, of population convergence. After ranking, pairs of parents are randomly chosen from the list of reproduction trials. Further, the fittest individual generated, up to the current generation, always takes part in reproduction. The genetic operators employed in GA are two-point crossover, two-phase mutation and inversion. Crossover is done by swapping a random segment of the data allocation configurations of two individuals. Mutation, with very small probability, refers to a random reallocation of a data element. Inversion, with small probability, inverts the allocation configuration of a segment of data in an individual. The rates of the three operators are made adaptive to the variation in the average degree of clustering of the data elements in the individuals, in a way that counteracts premature convergence. The genetic algorithm is hybridized by including a problem-specific hill-climbing procedure that directs the genetic search to the more profitable regions in the search space. The procedure allows the transfer of boundary data elements from overloaded to underloaded processors in a data allocation instance. That is, hill-climbing only allows the incremental reallocations that increase the fitness of an individual. In the later phase of the evolution, called the tuning phase, the population loses diversity and approaches convergence. In this phase, standard 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[22]. Further, some copies of identical individuals are gradually re moved, thus reducing the population size and the evolution time.

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, POP is the population size and GEN is the number of generations. In our implementation, POP has been chosen to be 0.2*P to 0.6*P, with larger population corresponding to larger N (hypercube size).

Two versions of GA are explored in this paper. Version GA1 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_{typ} for fitness and OF_{appr} for hill-climbing. It also includes nearly automatic invocation, based on OF_{typ}, 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 (for GEN generations)
    Set $\gamma$, $\beta$, 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 data allocation

4. SIMULATED ANNEALING ALGORITHMS

The simulated annealing approach is based on ideas from statistical mechanics and is motivated by an analogy to the physical annealing of a solid[16]. To coerce some material into a low-energy state, it is heated and then cooled very slowly, allowing it to come to thermal equilibrium at each temperature. The behavior of the system at each fixed temperature in the cooling schedule can be simulated by the Metropolis algorithm. An iteration of the Metropolis algorithm starts with proposing a random perturbation and evaluating the resultant change in the energy of the system. If the change is negative, corresponding to a downhill move in the energy landscape, the perturbation is accepted and the new lower energy configuration becomes the starting point for the next perturbation. Zero change is also accepted. If the energy change is positive, corresponding to an uphill move, the proposed perturbation may be accepted with a temperature-dependent probability. The main advantage of this Monte Carlo algorithm is that the controlled uphill movements can prevent the system from being prematurely trapped in a bad local minimum-energy state.

An outline of a simulated annealing algorithm (SAA) for data allocation is given in Figure 2. The initial data allocation is random. The energy of the system is given by the objective function used. The initial temperature is determined such that the probability of accepting uphill moves is initially 0.8. The freezing point is the temperature at which the ⋀ minimum increase in energy, resulting from the transfer of a data element from an underloaded processor to an overloaded one, is very small (2^{-30}). Perturbation is accomplished by a random reallocation of a randomly chosen data element. As explained above, a reallocation that leads to a lower or identical system energy is always accepted, whereas an increase in the energy is only probabilistically allowed. At each temperature, 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 N, the size of the multicomputer, and DEG, the average vertex degree, whereas the maximum number of accepted moves is the smaller of the two. These choices secure a sufficient

⋀ probability of accepting a

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 our implementation, this fraction varies within the range 0.91 to 0.99 in a way to counteract quenching (fast cooling) and to speed up cooling when possible. k increases if the number of accepted moves decreases, and vice versa.

```
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_typ for E;
until (freezing or convergence)
```

Figure 2. A simulated annealing algorithm for data allocation

The complexity of the SAA algorithm is of the order of (DEG*maxDEG, N*P*A), { } where A is the number of annealing steps; DEG, N 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_{typ} 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_{typ} makes SA2 much slower than SA1.

5. BOLD NEURAL NETWORK ALGORITHMS

The Bold Neural Network (BNN) is an improvement to the Hopfield and Tank model[18] applied to data allocation. It is built from $P * lg_2N$ neurons. Each neuron has a neural variable, $\underline{v}(e, i, t) = 0$ 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[19]

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

note correction of "prime" as ' not ʻ

where α is a scaling factor; G is the coupling matrix given by the computation grraph; 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, spontaneous magnetization domains of nearly equal numbers 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, α=β=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 (1/x)th root of the problem size[19], where x is the dimensionality of the problem.

The BNN algorithm is summarized in Figure 3. Its complexity is of the order of (DEG*P $*P^{1/x} lg_2N$). 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_{typ} decreases. In most cases, the execution time of NN1 is much smaller than that of NN2. However, NN2 can, sometimes, improve the

/ lg_2

```
for i = 0 to (N -1) do
    Generate random spins s(e, i, t) over whole domain;
    repeat
        Determine $\gamma$;
        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 data allocation

quality of NN1's solutions significantly and is included here for comparison with the genetic and annealing algorithms.

6. RECURSIVE BISECTION

Two recursive bisection methods are considered here to give some indication of the performance of the physical algorithms. These are orthogonal recursive coördinate bisection (RCB)[8] and recursive spectral bisection[6,7].

/ (RSB)

The operation of both methods is not guided by an objective function. Instead, RCB utilizes the physical coördinates of the vertices (data elements) of the computation graph to recursively bisect the graph into two subgraphs 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 coördinates in the selected separator direction, and each half of the elements is assigned to a subgraph. The recursive process continues until the number of subgraphs equals N. The complexity of the RCB is of the order of $(P * lg_2N(lg_2P - lg_2N))$.

RSB utilizes the properties of the Laplacian matrix associated with the computation graph. Briefly, each bisection step consists of computing the eigenvector corresponding to the second largest eigenvalue of the Laplacian matrix. The components of this vector provide distance information about the vertices of the graph. Then, the vertices are sorted according to the size of the eigenvector's components and split into two subgraphs accordingly. The complexity of this algorithm is of the order of (DEG*P $\sqrt{P} * lg_2N$)[6].

/ are

For a consistent comparison of the bisection methods with the physical algorithms, we have added a second step to map the subgraphs produced to the hypercube nodes. The mapping step is carried out by a simulated annealing algorithm that minimizes OF_{typ}.

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 object of studying these versions is to further explore the practical applicability of these data allocation methods.

The first version is based on two observations. The first observation is that methods such as NN1, RSB or RCB yield solutions of lower quality than those of SA and GA, but are considerably faster. The second observation is that SA and GA take a longer time to evolve solutions of the same quality as those of NN1 or the bisection methods. Therefore, hybrid methods which start with NN1, for example, 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 accepts 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 data allocation. 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 computational weight of the merged data elements and whose edges are weighted by the sum of the edges in the original graph. 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, with NN2's local optimization applied to the restored graph. However, K should not be small, as discussed below. Methods involving contraction are henceforth referred to as CONT-M, where M is the data allocation method used.

Graph contraction, with parameter K, leads to a big reduction in the search space of data allocation 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 GAs, 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.

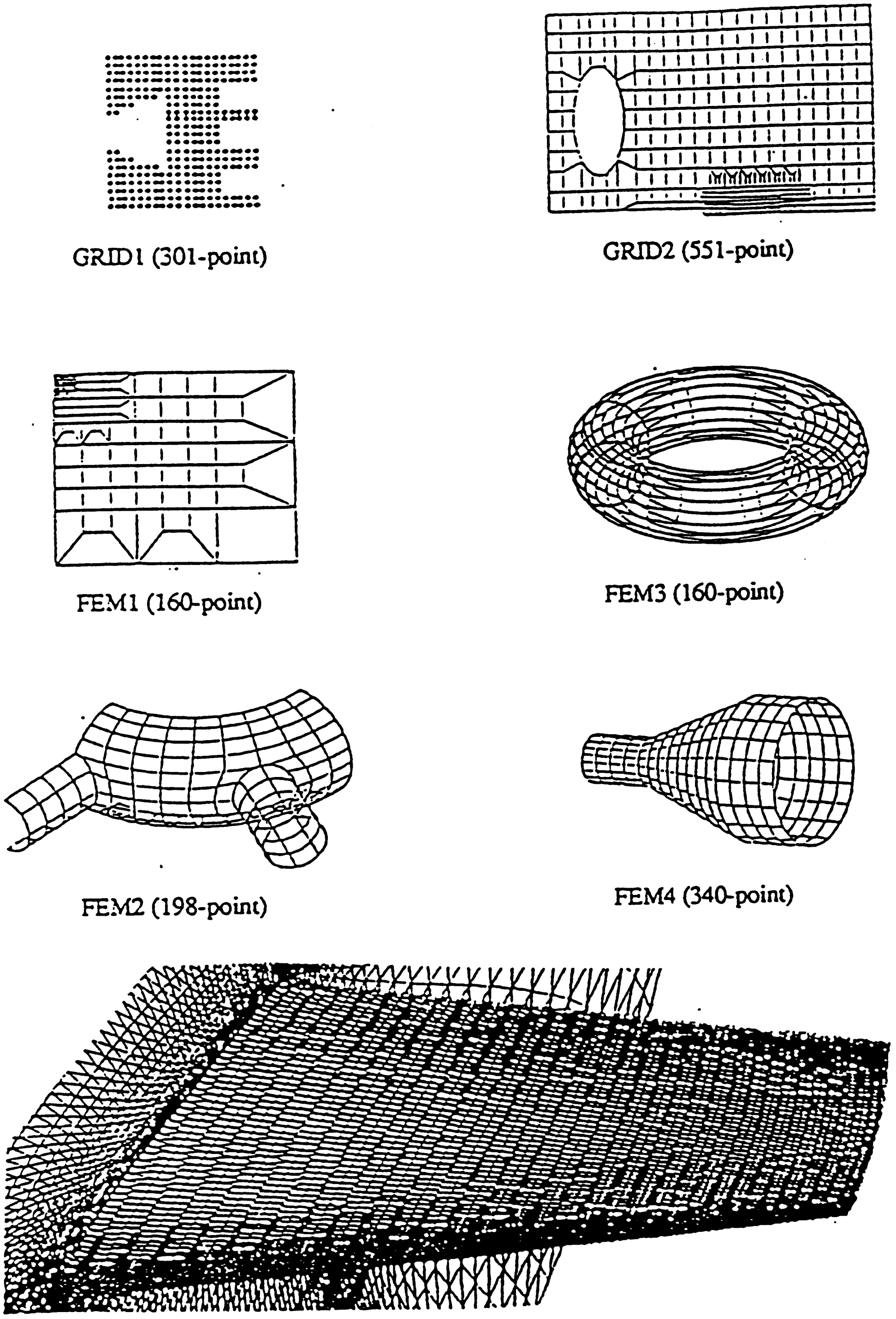
⅄ N^P ⅄ N^{K*N}

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. EXPERIMENTAL RESULTS

The performance of the data allocation algorithms is presented in this Section. Test cases of different geometric shapes, dimensions, sizes and granularities are allocated to hypercube multicomputers. The performance measures are the hypercube's efficiency and execution time. Efficiency is defined as $\Sigma_n W(n) / (N * OF_{typ})$.

The test cases used are shown in Figure 4. GRID1 and GRID2 are 2-dimensional and yield computation graphs with a maximum vertex degree, DEG_{max}, of 4. GRID1 is uniform with symmetric irregular geometry. GRID2 has large variation in the spatial density of its points. FEM1 and FEM2 are finite-element meshes with DEG_{max} ranging from 8 to 12. FEM1 is 2-dimensional and nonuniform. FEM2 is 3-dimensional. FEM3 and FEM4 are 3-dimensional structures with DEG_{max} equal to 8. FEMW is the most realistic of the seven examples; it is a coarse unstructured discretization of an aircraft wing with DEG_{max} of 16. We concentrate on FEMW and GRID1 because of their interesting and



GRID1 (301-point)

GRID2 (551-point)

FEM1 (160-point)

FEM3 (160-point)

FEM2 (198-point)

FEM4 (340-point)

FEMW (545-point)

Figure 4. Test cases.

distinct properties. We assume that $R = 5$ and $\omega = 12$; that is, the computation workload is moderate and communication is not inexpensive.

Table 1 provides a summary of the algorithms considered. Tables 2 through 6 show results for these algorithms. The results for the most interesting case, allocating FEMW to a 16-node hypercube, are also given in graphical form, in Figures 5–7. Each result of a physical algorithm 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, in minutes, 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 7 presents results for more test cases. Each result in Table 7 is the average of five runs. RCB's results are given only for FEMW and GRID1 where coordinates were available. The execution time shown for RCB and RSB does not include the second mapping step because the annealing algorithm aimed for the best mapping and was not optimized for time. The '+' sign refers to this additional time.

Table 1. Summary of algorithms

Version	.Description
GA1, SA1, NN1	Basic algorithms using only OF_{appr}.
GA2	OF_{typ} for fitness, OF_{appr} for hill climbing.
SA2	SA1, then uses OF_{typ} and neighbor-perturbation.
NN2	NN1, then local optimization based on OF_{typ}.
NN1-M hybrids	NN1, followed by physical algorithm M.
CONT-M	Physical algorithm M preceded by graph contraction (size of contracted graph = K*N).

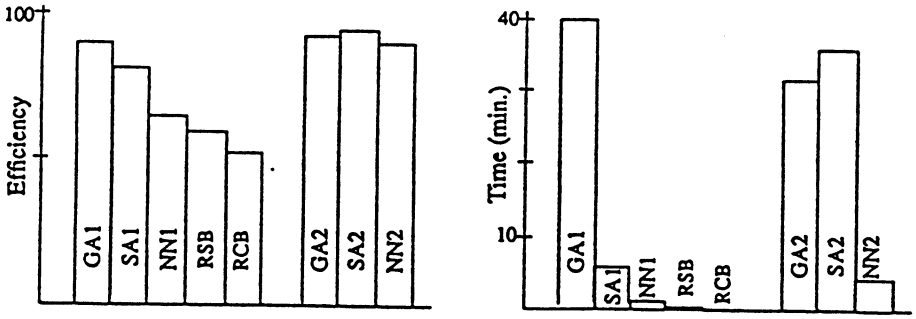


Figure 5. Results of versions involving OF_{typ} and OF_{appr} for FEMW and N=16

Please Note

Since fig. 5 is not discussed until the second paragraph in Section 9, it would look better if Fig. 5 is moved from this page to the next page; then it will appear after being referred to.

9. DISCUSSION

This Section starts with a discussion of the individual Tables, 2 to 7, 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, robustness and parallelizability are qualitatively discussed.

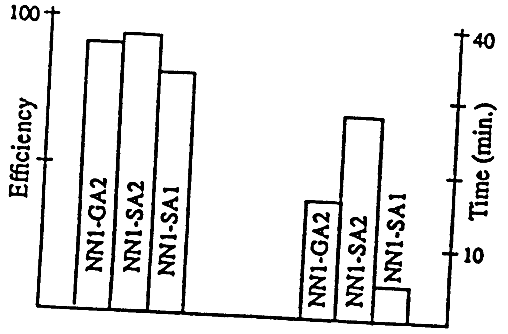


Figure 6. Results of NN1-M hybrid versions for FEMW and N=16

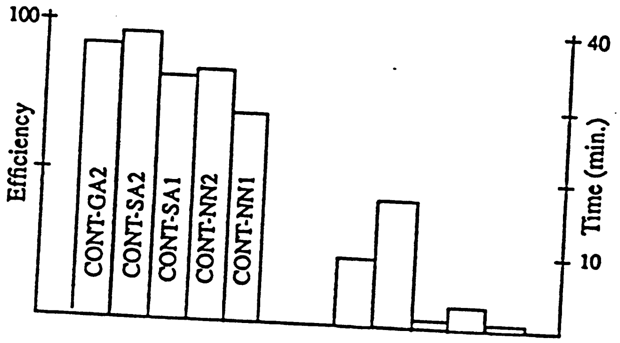


Figure 7. Results of CONT-M versions for FEMW, N=16;K=4 for GA and SA, and K=16 for NN

; instead of .

Table 2 shows the results of the versions of the physical algorithms that use the approximate objective function, OF_{appr}. Table 3 shows the results of the versions involving OF_{typ}. The results of RCB and RSB are also included. Figure 5 illustrates efficiency and time comparisons for FEMW. From the two Tables and the Figure, 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 FEMW, the quality of the solutions of the physical algorithms is clearly better than those of RCB and RSB, for a longer execution time. Nevertheless, the pronounced difference between the solutions of GA1, SA1 and NN1 themselves justifies the exploration of SA2 and NN2. Table 3 and Figure 5 show a clear improvement in the efficiency values with a substantial increase in time for SA2 and NN2. GA2 is more favorable than GA1. Also, due to its higher sensitivity to design parameters, GA1 is not pursued further. For FEMW, 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 loosely synchronous computations is mostly a result of the way OF_{typ} is used, explicitly by SA2 and only partially by GA2. This difference is not sufficient to evaluate the generic methods themselves. However, it highlights the importance of the formulation of the objective function for both solution quality and time.

Table 2. Results of versions using OF_{appr}

Test case	Best eff.	GA1		SA1		NN1		RCB		RSB	
		%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	time
FEMW $N = 16$	0.338	89	40.2	80	5.26	77	0.58	52	0.06+	72	0.16+
GRID1 $N = 8$	0.85	95	2.11	89	0.66	88	0.17	88	0.03+	90	0.06+

Table 3. Results of versions involving OF_{typ}

Test case	Best eff.	GA2		SA2		NN2		RCB		RSB	
		%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	time
FEMW $N = 16$	0.338	92	32.81	95	36.54	89	3.64	52	0.06+	72	0.16+
GRID1 $N = 8$	0.85	96	2.41	94	1.03	93	0.21	88	0.03+	90	0.06+
FEM3 $N = 8$	0.490	100	1.42	100	0.51	100	0.07			96	0.03+
FEM4 $N = 16$	0.495	96	13.40	94	9.01	85	0.41			79	0.07+

In Table 3, results for FEM3 and FEM4 are given. Owing to its symmetry and convenient number of points, FEM3 turns out to be an easy problem. The three physical algorithms find what seems to be an optimum. For FEM4, GA2 finds the best solution for the longest time.

The long time taken, especially by GA2 and SA2 in Table 3, justifies the exploration of the other versions, as in Tables 4–6. Table 4 and Figure 6 give 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 due to restricting randomness. In comparison with Table 3 and Figure 5, 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, RCB and RSB. However, the times taken by NN1-SA2 and NN1-GA2 are still long, and their decrease, as illustrated next, is of interest.

Tables 5 and 6 and Figure 7 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 5 includes results for different values of K. The efficiency values for CONT-GA2, CONT-SA2 and CONT-SA1 are consistent with those in Table 3, 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

Table 4. Results of NN1-M hybrid versions

Test case	Best eff.	NN1-GA2		NN1-SA1		NN1-SA2	
		%eff.	time	%eff.	time	%eff.	time
FEMW $N = 16$	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 5. Results of CONT-M versions for FEMW and N=16

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

← Pl. note correctn.

Table 6. Results of CONT-M versions for GRID~~2~~ and N=8

1 to become GRID1

	Best eff.	CONT-GA2		CONT-SA1		CONT-SA2		CONT-NN1		CONT-NN2	
		%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	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 7. Results of CONT-M versions, K=2 for GA and SA, K=16 for NN

Test case	Best eff.	CONT-GA2		CONT-SA1		CONT-SA2		CONT-NN1		CONT-NN2		RCB		RSB	
		%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	time	%eff.	time
FEMW N=8	0.452	92	2.92	88	1.61	96	7.15	76	0.05	90	1.50	53	0.04	81	0.10
FEM1 N=8	0.569	93	0.27	86	0.06	93	0.24	83	0.09	85	0.10			84	0.03
FEM2 N=8	0.578	92	0.26	86	0.06	97	0.43	81	0.05	89	0.10			76	0.04
FEM2 N=16	0.432	92	0.81	80	0.18	96	0.85	78	0.12	83	0.13			74	0.07
GRID2 N=16	0.787	92	1.42	77	0.28	94	1.94	73	0.20	85	0.30			84	0.16

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, with GA and SAA, the flexibility of operating on the restored original graph. Also, when the solution quality of NN1 is low, local optimization in NN2 gets trapped in high local minima. 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.

to become
followed

Table 7 includes results for other examples, with K=16 for CONT-NN and K=2 for CONT-SA and CONT-GA2. These results clearly support the assessments made above, based on Tables 3, 5 and 6. We note, however, that for GRID2 and FEMW, in Table 7, RSB yields better solutions than CONT-NN1 with comparable execution time. This is partly due to the use of contraction and because RSB performs well on 2-dimensional graphs. In the remaining paragraphs, overall evaluations are presented.

The solutions evolved by GA2, SA2 and NN2 are very good sub-optimal solutions. They are consistently better than those of recursive bisection. SA1 and NN1 also generate better solutions than recursive bisection for general, unstructured and 3-dimensional problems. For FEMW, for example, the improvements over RSB's solutions by NN1, SA1, GA2 and SA2 are 7%, 11%, 28% and 32%, respectively. The results for the various topologies and sizes indicate that the annealing and genetic algorithms are not biased towards particular problem topologies. Recursive bisection methods tend to favor 2-dimensional problems. The neural network performs better for 2-dimensional geometrical shapes, such as GRID1, than for 3-dimensional irregular structures, such as FEM2, but it does not show a strong bias. Therefore, the physical methods seem promising for a variety of problems with different topologies and complexities. Interestingly, comparative studies of algorithms for another NP-complete optimization problem, VLSI placement, have given similar conclusions about the better solution qualities of annealing and genetic algorithms[23].

- - - They

The better solutions of the physical algorithms come at a price; physical algorithms are slower than bisection algorithms. For FEMW, the ratios of the execution times of NN1, SA1, GA2 and SA2 to that of RSB are 2, 16, 100 and 120, respectively. These ratios decrease to 1, 3, 20 and 60 when contraction is employed. SA2 is generally the slowest and NN1 the fastest. It is worth noting that although NN1 and RSB have identical complexity, NN1 is slower by a small factor. The difference would be smaller if the mapping time is added to RSB's time.

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 the execution time. Although the bold neural network involves a probabilistic component, it has deterministic convergence.

The three physical methods, with their parameters chosen as described above, can be considered to be fairly robust, where robustness, in this paper, refers to insensitivity to design and problem parameters. Their robustness is enhanced by making some important parameters adaptive; these are the cooling schedule in SAA, the operator frequencies in GA, and γ in BNN. In our implementation, they vary within a range of 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 affect 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 has been observed to be less robust than SAA.

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[9]. 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[24].

10. CONCLUSIONS AND FURTHER RESEARCH

Versions of a genetic algorithm, a simulated annealing algorithm and a bold neural network for data allocation 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 suboptimal solutions. They are, for non-uniform problems, clearly better than those of recursive bisection methods, RCB and RSB, especially for 3-dimensional irregular and unstructured problems. However, the diverse properties of the physical methods and their versions suggest that the choice of one of them depends on the particular application. SA2 produces the best solution quality. It is followed by GA2, NN2, SA1, NN1, RSB and RCB in order of decreasing quality, for general problems. For uniform 2-dimensional problems, the solutions of NN1 and RSB are rather close. The order of decreasing execution time is the same as that for solution quality, with NN2 and SA1 swapped in several cases. The physical algorithms are slower than recursive bisection. However, the execution times of NN1 and RSB do not differ greatly.

The annealing and genetic algorithms have the ability to start 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 recursive bisection 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, BNN exhibits some tendency to favor 2-dimensional problems; 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. Based on the promising results of the physical optimization algorithms, further research is underway to study their performances for various classes of problems, such as large-scale data sets, complex geometries, adaptive meshes, heterogeneous problems or dynamically varying problems. In particular, the performance of parallel physical algorithms is being explored[24]. s/

ACKNOWLEDGEMENTS

This work was supported by the Joint Tactical Fusion Program Office. We wish to thank the referees for their useful comments. We also thank H. Simon for providing the RSB code, W. Furmanski for useful discussion about BNN, J. Saltz and R. Ponnusamy for providing the FEMW data, and Y-C. Chung for FEM1-4 data.

REFERENCES

1. G.C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, 1988.
2. M. Berger and S. Bokhari, 'A partitioning strategy for nonuniform problems on multiprocessors', *IEEE Trans.*, C-36(5), 570–580 (1987).
3. N.P. Chrisochoides, E.N. Houstis and C.E. Houstis, 'Geometry based mapping strategies for PDE computations', *Int. Conf. on Supercomputing*, ACM Press 1991, pp. 115–127.
4. F. Ercal, 'Heuristic approaches to task allocation for parallel computing', Doctoral Dissertation, Ohio State University, 1988.
5. G.C. Fox, 'A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube', In *Numerical Algorithms for Modern Parallel Computers*, M. Schultz (Ed.), Springer-Verlag, Berlin, 1988.
6. A. Pothen, H. Simon and K-P. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl.*, 11(3), 430–452 (1990).
7. 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.
8. D. Walker, 'Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code', *Concurrency Practice and Experience*, 2, 257–288 (1990).
9. R.D. Williams, 'Performance of dynamic load balancing algorithms for unstructured mesh calculations', *Concurrency Practice and Experience*, 3(5), 457–481 (1991).
10. G.C. Fox, 'Physical computation', *Int. Conf. Parallel Computing: Achievements, Problems and Prospects*, Italy, June 1990.
11. H. Muhlenbein, 'Parallel genetic algorithms, population genetics, and combinatorial optimization', *Proc. Int. Conf. on Genetic Algorithms*, 1989, pp. 416–421.
12. D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, 'Optimization by simulated annealing: an experimental evaluation; Part I, Graph partitioning, *Operations Research*, 37(6), 865–892 (1989).
13. G. Laszewski, 'Intelligent structural operators for the k-way graph partitioning problem', *Proc. Int. Conf. on Genetic Algorithms*, July 1991, pp. 45–52.
14. R. Durbin and D. Willshaw, 'An analogue approach to the travelling salesman problem using an elastic net method', *Nature*, 326(16) (1987).
15. H. Date, M. Seki and T. Hayashi, 'Module placement methods using neural computation networks', *Int. Joint Conf. on Neural Networks*, Vol. III, 1990, pp. 831–836.
16. S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, 'Optimization by simulated annealing', *Science*, 220, 671–680 (1983).
17. J. Flower, S. Otto and M. Salama, 'A preprocessor for finite element problems', *Proc. Symp. Parallel Computations and their Impact on Mechanics*, ASME Winter Meeting, Dec. 1987.
18. J.J. Hopfield and D.W. Tank, 'Computing with neural circuits: a model', *Science*, 233, 625–639 (1986).
19. G.C. Fox and W. Furmanski, 'Load balancing loosely synchronous problems with a neural network', *Proc 3rd Conf. Hypercube Concurrent Computers and Applications*, 241–278 (1988).
20. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
21. J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, 1975.
22. N. Mansour and G.C. Fox, 'A hybrid genetic algorithm for task allocation', *Proc. Int. Conf. Genetic Algorithms*, July 1991, pp. 466–473.

23. K. Shahookar and P. Mazumder, VLSI Cell Placement Techniques, *ACM Computing Surveys*, 23(2), 143–220 (1991).
24. N. Mansour and G.C. Fox, 'Parallel physical optimization methods for load balancing parallel computations', in preparation.