A NEW MAPPING HEURISTIC BASED ON MEAN FIELD ANNEALING[1]

Tevfik Bultan and **Cevdet Aykanat**

Department of Computer Engineering and Information Science,
Bilkent University, 06533 Bilkent, Ankara, Turkey

[1] This work is partially supported by Intel Supercomputer Systems Division under Grant SSD100791-2 and Turkish Science and Research Council under Grant EEEAG-5.

A NEW MAPPING HEURISTIC BASED ON MFA

Abstract

A new mapping heuristic is developed, based on the recently proposed Mean Field Annealing (MFA) algorithm. An efficient implementation scheme, which decreases the complexity of the proposed algorithm by asymptotical factors, is also given. Performance of the proposed MFA algorithm is evaluated in comparison with two well-known heuristics; Simulated Annealing and Kernighan-Lin. Results of the experiments indicate that MFA can be used as an alternative heuristic for solving the mapping problem. Inherent parallelism of MFA is exploited by designing an efficient parallel algorithm for the proposed MFA heuristic.

1 Introduction

Today, with the aid of VLSI technology, parallel computers not only exist in research laboratories, but are also available on the market as powerful, general purpose computers. Wide use of parallel computers in various compute intensive applications makes the problem of mapping parallel programs to parallel computers more crucial. The mapping problem arises while developing parallel programs for distributed-memory, message-passing parallel computers which are usually named as multicomputers. In multicomputers, processors have neither shared memory nor shared address space. Each processor can only access its local memory. Synchronization and coordination among processors are achieved through explicit message passing. Processors of a multicomputer are usually connected by utilizing one of the well-known direct interconnection network topologies such as ring, mesh, hypercube, etc. These architectures have the nice scalability feature due to the lack of shared resources and the increasing communication bandwidth with the increasing number of processors. However, designing efficient parallel algorithms for such architectures is not straightforward. An efficient parallel algorithm should exploit the full potential power of the architecture. Processor idle time and the interprocessor communication overhead may lead to poor utilization of the architecture, hence poor overall system performance.

Parallel algorithm design for multicomputers can be divided into two steps. First step is the decomposition of the problem into a set of interacting sequential sub-problems (or tasks) which can be executed in parallel. Second step is mapping each one of these tasks to an individual processor of the parallel architecture in such a way that the total execution time is minimized. The second step, named as the mapping problem [4], is very crucial in designing efficient parallel programs. In general, the mapping problem is known to be NP-hard [12, 13]. Hence, heuristics giving sub-optimal solutions are used to solve the problem [1, 4, 7, 12, 13, 21]. Two distinct approaches have been considered in the context of mapping heuristics; one phase and two phase [7]. In one phase approaches, referred as *many-to-one mapping*, tasks of the parallel program are directly mapped onto the processors of the multicomputer. In two phase approaches, *clustering* phase is followed by *one-to-one mapping* phase. In the clustering phase, tasks of the parallel program are partitioned into as many equal weighted clusters as the number of processors of the multicomputer, while minimizing the total weight of the interactions among clusters [21]. The problem solved in the clustering phase is identical to the multi-way graph partitioning problem. In the one-to-one mapping phase, each cluster is assigned to an individual processor of the multicomputer such that the total inter-processor communication is minimized [21]. Kernighan-Lin (KL) [8, 14] and Simulated Annealing (SA) [15] heuristics are two attractive algorithms widely used for solving the mapping problem [7, 19, 21, 22].

Heuristics proposed to solve the mapping problem are compute intensive. Solving the mapping problem can be considered as a preprocessing performed before the execution of the parallel program on the parallel computer. Sequential execution of the mapping heuristic may introduce unacceptable preprocessing overhead, limiting the efficiency of the parallel implementation. Efficient parallel mapping heuristics are needed in such cases. The KL and SA heuristics are inherently sequential, hence hard to parallelize. Efficient parallelizations of these algorithms remain as important issues in parallel processing research.

In this work, a recently proposed algorithm, called Mean Field Annealing (MFA) [18, 24, 25] is formulated for the many-to-one mapping problem. MFA combines the collective computation property of Hopfield Neural Networks (HNN) with the annealing notion of SA. It is originally proposed for solving traveling salesperson problem, as a working alternative to HNN [23]. MFA is also a general strategy as SA, and can be applied to different problems with suitable formulations. Previous works on MFA [5, 6, 17, 18, 24, 25] show that it can be successfully applied to various combinatorial optimization problems. MFA has the inherent parallelism that exists in most of the neural network algorithms.

Section 2 presents a formal definition of the mapping problem by modeling parallel program design process. In Section 3, general formulation of the MFA heuristic is presented. Section 4 presents the proposed formulation of the MFA algorithm for the mapping problem. An efficient implementation scheme for the proposed algorithm is also described in this section. Section 5 presents the performance evaluation of the MFA algorithm for the mapping problem in comparison with two well known mapping heuristics; SA and KL. Finally, efficient parallelization of the MFA algorithm for the mapping problem is proposed in Section 6.

2 The Mapping Problem

In various classes of problems, interaction pattern among the tasks is static. Hence, the decomposition of the algorithm can be represented by a static task graph. Vertices of this graph represent the atomic tasks and the edge set represents the interaction pattern among the tasks. Relative computational costs of atomic tasks can be known or estimated prior to the execution of the parallel program. Hence, weights can be associated with the vertices in order to denote the computational costs of the corresponding tasks.

Two different models, Task Precedence Graph (TPG) and Task Interaction Graph (TIG), are used for modeling static task interaction patterns[13, 20]. TPG is a directed graph where directed edges represent execution dependencies. Each edge denotes a pair of tasks; source and destination. The destination task can only be executed after the completion of the execution of the source task. In general, only the subsets of tasks which are unreachable from each other in TPG can be executed independently.

In the TIG model, interaction patterns are represented by undirected edges between vertices. In this model, each atomic task can be executed simultaneously and independently. Each edge denotes the need for the bidirectional interaction between corresponding pair of tasks at the completion of the execution of these tasks. Edges may be associated with weights which denote the amount of bidirectional information exchange involved between pairs of tasks. TIG usually represents the repeated execution of the tasks with intervening task interactions denoted by the edges.

The TIG model may seem to be unrealistic for general applications since it does not consider the temporal interaction dependencies among the tasks [20]. However, there are various classes of problems which can be successfully modeled with the TIG model. For example, iterative solution of systems of equations arising in finite element applications [2, 20] and power system simulations [3, 16], and VLSI simulation programs [22] are represented by TIGs. In this paper, problems which can be represented by the TIG model are addressed.

In order to solve the mapping problem, parallel architecture must also be modeled in a way that represents its architectural features. Parallel architectures can easily be represented by a

Processor Organization Graph (POG), where nodes represent the processors and edges represent the communication links. In fact, POG is a graphical representation of the interconnection topology utilized for the organization of the processors of the parallel architecture. In general, nodes and edges of a POG are not associated with weights since most of the commercially available multicomputer architectures are homogeneous with identical processors and communication links.

In a multicomputer architecture, each adjacent pair of processors communicate with each other over the communication link connecting them. Such communications are referred as *single-hop* communications. However, each non-adjacent pair of processors can also communicate with each other by means of *software* or *hardware routing*. Such communications are referred as *multi-hop* communications. Multi-hop communications are usually routed in a static manner over the shortest path of links between the communicating pairs of processors. Communications between non-adjacent pairs of processors can be associated with relative unit communication costs. Unit communication cost is defined as the communication cost per unit of information. Unit communication cost between a pair of processors will be a function of the shortest path between these processors and the routing scheme used for multi-hop communications. For example, in software routing, the unit communication cost is linearly proportional to the shortest path distance between the pair of communicating processors. Hence, the communication topology of the multicomputer can be modeled by an undirected complete graph, referred here as Processor Communication Graph (PCG). The nodes of PCG represent the processors and the weights associated with the edges represent the unit communication costs between pairs of processors. As is mentioned earlier, PCG can easily be constructed using the topological properties of POG and the *routing* scheme utilized for inter-processor communication.

The objective in mapping TIG to PCG is the minimization of the expected execution time of the parallel program on the target architecture. Thus, the mapping problem can be modeled as an optimization problem by associating the following quality measures with a good mapping : (i) interprocessor communication overhead should be minimized, (ii) computational load should be uniformly distributed among processors in order to minimize processor idle time.

A mapping problem instance can be formally represented with two undirected graphs, Task Interaction Graph (TIG) and Processor Communication Graph (PCG). The TIG $G_T(V,E)$, has $|V| = N$ vertices labeled as $(1, 2, \ldots, i, j, \ldots, N)$. Vertices of the G_T represent the atomic tasks of the parallel program. Vertex weight w_i denotes the computational cost associated with task i for $1 \leq i \leq N$. Edge weight e_{ij} denotes the volume of interaction between tasks i and j connected by edge $(i, j) \in E$. The PCG $G_P(P, D)$, is a complete graph with $|P| = K$ nodes and $|D| = (\begin{smallmatrix} K \\ 2 \end{smallmatrix})$ edges. Nodes of the G_P, labeled as $(1, 2, \ldots, p, q, \ldots, K)$, represent the processors of the target multicomputer. Edge weight d_{pq}, for $1 \leq p, q \leq N$ and $p \neq q$, denotes the unit communication cost between processors p and q.

Given an instance of the mapping problem with the TIG $G_T(V, E)$ and the PCG $G_P(P, D)$, the question is to find a many-to-one mapping function $M : V \rightarrow P$, which assigns each vertex of the graph G_T to a unique node of the graph G_P, and minimizes the total interprocessor communication cost (CC)

$$CC = \sum_{(i,j) \in E, M(i) \neq M(j)} e_{ij} d_{M(i)M(j)} \tag{1}$$

while maintaining the computational load (CL_p : computational load of processors p)

$$CL_p = \sum_{i \in V, M(i) = p} w_i, \qquad 1 \leq p \leq K \tag{2}$$

of each processor balanced. Here, $M(i) = p$ denotes the label (p) of the the processor that task i is mapped to. In Eq. (1), each edge (i, j) of the G_T contributes to the communication cost (CC), only if vertices i and j are mapped to two different nodes of the G_P, i.e. $M(i) \neq M(j)$. The amount of contribution is equal to the product of the volume of interaction e_{ij} between these two tasks and the unit communication cost d_{pq} between processors p and q where $p = M(i)$ and $q = M(j)$. The computational load of a processor is the summation of the weights of the tasks assigned to that processor. Perfect load balance is achieved if $CL_p = (\sum_{i=1}^{N} w_i)/K$ for each p, $1 \leq p \leq K$. Computational load balance of the processors can be explicitly included in the cost function using a term which is minimized when all processor loads are equal. Another scheme is to include load balance criteria implicitly in the algorithm. Figure 1 illustrates a sample mapping problem instance. Figure 1(a) shows a TIG with $N = 8$ tasks. Figure 1(b) shows POG of a 2-dimensional hypercube with $K = 4$ processors, and Figure 1(c) shows the corresponding PCG. In Figure 1, numbers inside the circles denote the vertex labels, and numbers within the parenthesis denote the vertex or edge weights. Binary labeling of the 2-dimensional hypercube is also given in Figure 1(b). Note that, unit communication cost assignment to edges of PCG

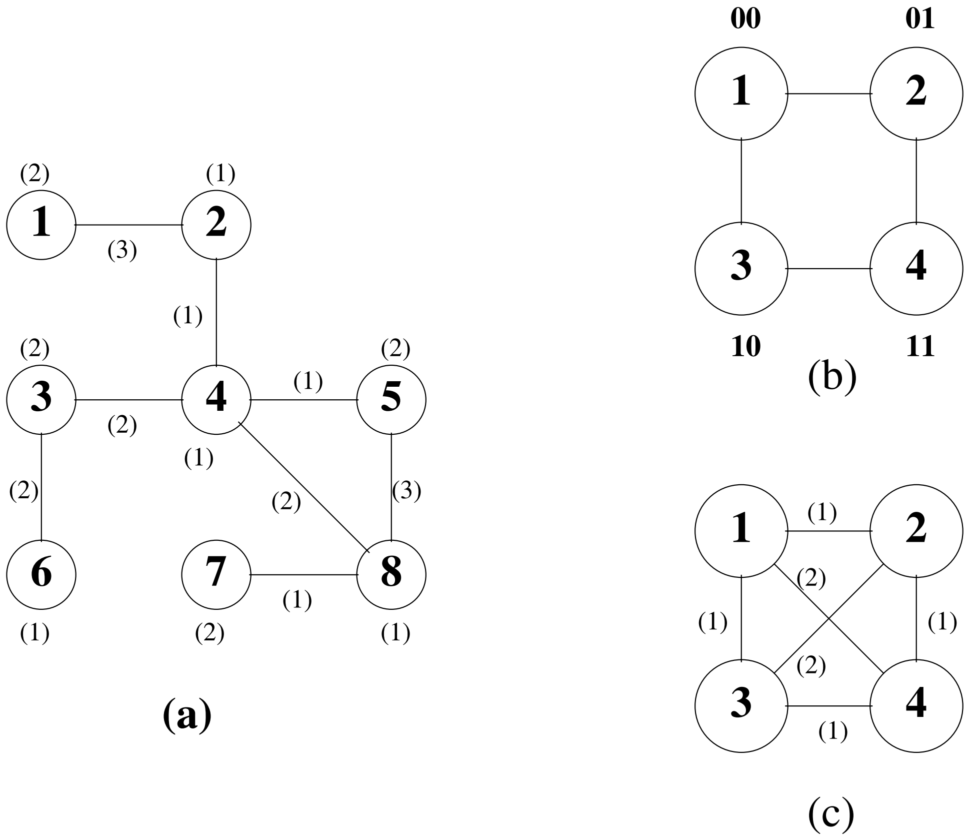

Figure 1: A mapping problem instance, with (a) TIG, (b) POG (which represents a 2-dimensional hypercube) and (c) PCG.

is performed assuming software routing protocol for multi-hop communications. A solution to the mapping problem instance shown in Figure 1 is

i	1	2	3	4	5	6	7	8
$M(i)$	4	4	2	1	3	2	1	3

Communication cost of this solution can be calculated as $CC = 8$. Computational loads of the processors are $CL_p = 3$ for $1 \leq p \leq 4$. Hence, perfect load balance is achieved, since $(\sum_{i=1}^{8} w_i)/4 = 3$.

3 Mean Field Annealing

Mean Field Annealing (MFA) merges collective computation and annealing properties of Hopfield Neural Networks (HNN) [9, 10, 11] and Simulated Annealing (SA) [15], respectively, to obtain a general algorithm for solving combinatorial optimization problems. HNN is used for solving various optimization problems and reasonable results are obtained for small size problems [9]. However, simulations of this network reveals the fact that it is hard to obtain *feasible solutions* for large problem sizes. Hence, the algorithm does not have a good scaling property, which is a very important performance criterion for heuristic optimization algorithms. MFA is proposed as a successful alternative to HNN [18, 23, 24, 25]. In the MFA algorithm, problem representation is identical to HNN [9, 23, 24], but iterative scheme used to relax the system is different. MFA can be used for solving a combinatorial optimization problem by choosing a representation scheme in which the final states of the *spins* can be decoded as a solution to the target problem. Then, an energy function is constructed whose global minimum value corresponds to the *best solution* of the problem to be solved. MFA is expected to compute the *best solution* to the target problem, starting from a randomly chosen initial state, by minimizing this energy function.

The MFA algorithm is derived by making an analogy to Ising spin model which is used to estimate the state of a system of particles or spins in thermal equilibrium. This method was first proposed for solving the traveling salesperson problem [23] and then it is applied to the graph partitioning problem [5, 6, 17, 25]. Here, general formulation of the MFA algorithm [25] is given for the sake of completeness. In the Ising spin model, the energy of a system with S spins has the following form:

$$H(\mathbf{s}) = \frac{1}{2}\sum_{k=1}^{S}\sum_{l\neq k}\beta_{kl}s_k s_l + \sum_{k=1}^{S}h_k s_k \tag{3}$$

Here, β_{kl} indicates the level of interaction between spins k and l, and $s_k \in \{0,1\}$ is the value of spin k. It is assumed that $\beta_{kl} = \beta_{lk}$ and $\beta_{kk} = 0$ for $1 \leq k, l, \leq S$. At thermal equilibrium, spin average $\langle s_k \rangle$ of spin k can be calculated using Boltzmann distribution as follows [23]

$$\langle s_k \rangle = \frac{1}{1+e^{-\phi_k/T}} \tag{4}$$

Here, $\phi_k = \langle H(\mathbf{s})\rangle|_{s_k=0} - \langle H(\mathbf{s})\rangle|_{s_k=1}$ represents the *mean field* effecting on spin k, where the energy average $\langle H(\mathbf{s})\rangle$ of the system is

$$\langle H(\mathbf{s})\rangle = \sum_{k=1}^{S}\sum_{l\neq k}\beta_{kl}\langle s_k s_l\rangle + \sum_{k=1}^{S}h_k\langle s_k\rangle \tag{5}$$

The complexity of computing ϕ_k using Eq. (5) is exponential [25]. However, for large number

```
1. Get the initial temperature $T_0$, and set $T = T_0$
2. Initialize the spin averages $\langle \mathbf{s} \rangle = [\langle s_1 \rangle, \ldots, \langle s_k \rangle, \ldots, \langle s_S \rangle]$
3. While temperature $T$ is in the cooling range DO
    3.1 While system is not stabilized for current temperature DO
        3.1.1 Select a spin $k$ at random.
        3.1.2 Compute $\phi_k$,   $\phi_k = -\sum_{l \neq k} \beta_{kl} \langle s_l \rangle - h_k$
        3.1.3 Update $\langle s_k \rangle$,   $\langle s_k \rangle = \{1 + e^{-\phi_k / T}\}^{-1}$
    3.2 Update $T$ according to the cooling schedule
```

Figure 2: The Mean Field Annealing algorithm.

of spins, *mean field approximation* can be used to compute the energy average as

$$\langle H(\mathbf{s}) \rangle = \frac{1}{2} \sum_{k=1}^{S} \sum_{l \neq k} \beta_{kl} \langle s_k \rangle \langle s_l \rangle + \sum_{k=1}^{S} h_k \langle s_k \rangle \tag{6}$$

Since $\langle H(\mathbf{s}) \rangle$ is linear in $\langle s_k \rangle$, mean field ϕ_k can be computed using the following equation.

$$\phi_k = \langle H(\mathbf{s}) \rangle|_{s_k=0} - \langle H(\mathbf{s}) \rangle|_{s_k=1} = -\frac{\partial \langle H(\mathbf{s}) \rangle}{\partial \langle s_i \rangle} = -\left(\sum_{l \neq k} \beta_{kl} \langle s_l \rangle + h_k \right) \tag{7}$$

Thus, the complexity of computing ϕ_k reduces to $O(S)$.

At each temperature, starting with initial spin averages, the mean field effecting on a randomly selected spin is computed using Eq. (7). Then, spin average is updated using Eq. (4). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. The general form of the MFA algorithm derived from this iterative relaxation scheme is shown in Figure (2). The MFA algorithm is used to find the equilibrium point of a system of S spins using an annealing process similar to SA.

HNN and SA have a major difference; SA is an algorithm implemented in software, whereas HNN is derived with a possible hardware implementation in mind. MFA is somewhere in between, it is an algorithm implemented in software, having potential for hardware realization [24, 25]. In this work MFA is treated as a software algorithm. Performance of MFA is comparable to other software algorithms as SA and KL, conforming this point of view.

4 Mean Field Annealing for the Mapping Problem

In this section, we propose a formulation of the Mean Field Annealing (MFA) algorithm for the mapping problem. The TIG and PCG models described in Section 2 are used to represent the mapping problem. The formulation is first presented for problem instances modeled by dense TIGs. The modifications in the formulation for the mapping problem instances that can be modeled by sparse TIGs are presented later. In this section, we also present an efficient implementation scheme for the proposed formulation.

4.1 Formulation

A spin matrix, which consists of N task-rows and K processor-columns, is used as the representation scheme. That is, $N \times K$ spins are used to encode the solution. The output s_{ip} of a spin (i,p) denotes the probability of mapping task i to processor p. Here, s_{ip} is a continuous variable in the range $0 \leq s_{ip} \leq 1$. When the MFA algorithm reaches to a solution, spin values converge to either 1 or 0 indicating the result. If s_{ip} converges to 1, this means that task i is mapped to processor p. For example, a solution to the mapping problem instance given in Figure 1 can be represented by the following $N \times K$ spin matrix.

$$\begin{array}{rccccc}
 & & \multicolumn{4}{c}{\mathbf{K\ Processors}} \\
 & & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\
 & \mathbf{1} & 0 & 0 & 0 & 1 \\
 & \mathbf{2} & 0 & 0 & 0 & 1 \\
 & \mathbf{3} & 0 & 1 & 0 & 0 \\
 & \mathbf{4} & 1 & 0 & 0 & 0 \\
\mathbf{N\ Tasks} & \mathbf{5} & 0 & 0 & 1 & 0 \\
 & \mathbf{6} & 0 & 1 & 0 & 0 \\
 & \mathbf{7} & 1 & 0 & 0 & 0 \\
 & \mathbf{8} & 0 & 0 & 1 & 0
\end{array}$$

Note that, this solution is identical to the solution given at the end of Section 2.

Following energy (i.e., cost) function is proposed for the mapping problem

$$H(\mathbf{s}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}\sum_{p=1}^{K}\sum_{q\neq p} e_{ij}s_{ip}s_{jq}d_{pq} + \frac{r}{2}\sum_{i=1}^{N}\sum_{j\neq i}\sum_{p=1}^{K} s_{ip}s_{jp}w_iw_j \qquad (8)$$

Here, e_{ij} denotes the edge weight between the pair of tasks i and j, and w_i denotes the weight of task i in TIG. Edge weight between processors p and q in PCG is represented by d_{pq}. Under the mean field approximation, the expression $\langle H(\mathbf{s})\rangle$ for the expected value of the cost function will be similar to the expression given for $H(\mathbf{s})$ in Eq. (8). However, in this case, s_{ip}, s_{iq} and s_{jp} should be replaced with $\langle s_{ip}\rangle$, $\langle s_{iq}\rangle$ and $\langle s_{jp}\rangle$ respectively. For the sake of simplicity, s_{ip} is used to denote the expected value of spin (i,p) (i.e., spin average $\langle s_{ip}\rangle$).

In Eq. (8), the term $s_{ip} \times s_{jq}$ denotes the probability that task i and task j are mapped to two different processors p and q, respectively. Hence, the term $e_{ij} \times s_{ip} \times s_{jq} \times d_{pq}$ represents the weighted interprocessor communication overhead introduced due to the mapping of tasks i and j to different processors. Note that, in Eq. (8), the first quadruple summation term covers all processor pairs in PCG for each edge pair in TIG. Hence, this term denotes the total interprocessor communication cost for a mapping represented by an instance of the spin matrix. Then, minimization of the first quadruple summation term corresponds to the minimization of the interprocessor communication overhead.

Second triple summation term in Eq. (8) computes the summation of the inner products of the weights of the tasks mapped to individual processors. Global minimum of this term occurs when equal amount of task weights are mapped to each processor. If there is an imbalance in the mapping, second triple summation term increases with the square of the amount of the imbalance, penalizing imbalanced mappings. The parameter r in Eq. (8) is introduced to maintain a balance between the two optimization objectives of the mapping problem.

Using the mean field approximation described in Eq. (7), the expression for the mean field ϕ_{ip} experienced by spin (i,p) is

$$\phi_{ip} = -\frac{\partial H(\mathbf{s})}{\partial s_{ip}} = -\sum_{j\neq i}^{N}\sum_{q\neq p}^{K} e_{i,j} s_{jq} d_{pq} - r\sum_{j\neq i}^{N} s_{jp} w_i w_j \tag{9}$$

In a feasible mapping, each task should be mapped exclusively to a single processor. However, there exists no penalty term in Eq. (8) to handle this feasibility constraint. This constraint is explicitly handled while updating the spin values. As is seen in Eq. (4), individual spin average s_{ip} is proportional to $e^{\phi_{ip}/T}$, i.e. $s_{ip}\ \alpha\ e^{\phi_{ip}/T}$. Then, s_{ip} can be normalized as

$$s_{ip} = \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^{K} e^{\phi_{iq}/T}} \tag{10}$$

This normalization enforces the summation of each row of the spin matrix to be equal to unity. Hence, it is guaranteed that all rows of the spin matrix will have only one spin with output value 1 when the system is stabilized.

Eq. (9) can be interpreted in the context of the mapping problem as follows. First double summation term represents the increase in the total interprocessor communication cost by mapping task i to processor p. Second summation term represents the increase in the computational load balance cost associated with processors p by mapping task i to processor p. Hence, $-\phi_{ip}$ may be interpreted as the decrease in the overall solution quality by mapping task i to processor p. Then, in Eq. (10), s_{ip} is updated such that the probability of task i being mapped to processor p increases with increasing mean field ϕ_{ip} experienced by spin (i, p). Hence, the MFA heuristic can be considered as a gradient-descent type algorithm in this context. However, it is also a stochastic algorithm, similar to SA, due to the random spin update scheme and the annealing process.

In the general MFA algorithm given in Figure 2, a randomly chosen spin is updated at a time. However, in the proposed formulation of MFA for the mapping problem, K spins of a randomly chosen row of the spin matrix are updated at a time. Mean fields ϕ_{ip}, $(1 \leq p \leq K)$ experienced by the spins at the i-th row of the spin matrix are computed using Eq. (9) for $p = 1, 2, \ldots, K$. Then, the spin averages s_{ip}, $1 \leq p \leq K$ are updated using Eq. (10) for $p = 1, 2, \ldots, K$. Each row update of the spin matrix is referred as a single iteration of the algorithm.

The system is observed after each spin-row update in order to detect the convergence to an equilibrium state for a given temperature [24]. If energy function H does not decrease after a certain number of consecutive spin-row updates, this means that the system is stabilized for that temperature [24]. Then, T is decreased according to the cooling schedule, and iteration process is re-initiated. Note that, the computation of the energy difference ΔH necessitates the computation of H (Eq. (8)) at each iteration. The complexity of computing H is $O(N^2 \times K^2)$, which drastically increases the complexity of one iteration of MFA. Here, we propose an efficient scheme which reduces the complexity of energy difference computation by an asymptotical factor.

The *incremental* energy change δH_{ip} due to the *incremental* change δs_{ip} in the value of an individual spin (i, p) is

$$\delta H = \delta H_{ip} = \phi_{ip} \delta s_{ip} \tag{11}$$

from Eq. (7). Since, $H(\mathbf{s})$ is linear in s_{ip} (see Eq. (8)), above equation is valid for any amount of change Δs_{ip} in the value of spin (i, p), that is

$$\Delta H = \Delta H_{ip} = \phi_{ip} \Delta s_{ip} \tag{12}$$

At each iteration of the MFA algorithm, K spin values are updated in a synchronous manner. Hence, Eq. (12) is valid for all spin updates performed in a particular iteration. Thus, energy difference due to the spin-row update operation in a particular iteration can be computed as

$$\Delta H = \sum_{p=1}^{K} \phi_{ip} \Delta s_{ip} \tag{13}$$

where $\Delta s_{ip} = s_{ip}^{new} - s_{ip}^{old}$. The complexity of computing Eq. (13) is only $O(K)$ since mean field (ϕ_{ip}) values are already computed for the spin updates.

The formulation of the MFA algorithm for the mapping problem instances with sparse TIGs is as follows. The expression given for ϕ_{ip} (Eq. (9)) can be modified for sparse TIGs as

$$\phi_{ip} = -\sum_{j \in Adj(i)} \sum_{q \neq p}^{K} e_{i,j} s_{jq} d_{pq} - r \sum_{j \neq i}^{N} s_{jp} w_i w_j \tag{14}$$

Here, $Adj(i)$ denotes the set of tasks connected to task i in the given TIG. Note that, sparsity of TIG can only be exploited in the mean field computations since spin update operations given in Eq. (10) are dense operations which are not effected by the sparsity of TIG.

Figure 3 illustrates the MFA algorithm proposed for solving the mapping problem. Complexity of computing first double summation terms in Eq. (9) and Eq. (14) are $O(N \times K)$ and $O(d_{avg} \times K)$ for dense and sparse TIGs respectively. Here, d_{avg} denotes the average vertex degree in the sparse TIG. Second summation operations in Eq. (9) and Eq. (14) are both $O(N)$ for dense and sparse TIGs. Then, complexity of a single mean field computation is $O(N \times K)$ and $O(d_{avg} \times K + N)$ for dense (Eq. (9)) and sparse (Eq. (14)) TIGs respectively. Hence, complexity of mean field computations for a spin row is $O(N \times K^2)$ for dense TIGs, and $O(d_{avg} \times K^2 + N \times K)$ for sparse TIGs (step 3.1.2 in Figure 3). Spin update computations (steps 3.1.3, 3.1.4 and 3.1.6) and energy difference computation (step 3.1.5) are both $O(K)$ operations. Hence, the overall complexity of a single MFA iteration is $O(N \times K^2)$ for dense TIGs, and $O(d_{avg} \times K^2 + N \times K)$ for sparse TIGs.

```
1. Get the initial temperature $T_0$, and set $T = T_0$
2. Initialize the spin averages $\mathbf{s} = [s_{11}, \ldots, s_{ip}, \ldots, s_{NK}]$
3. While temperature $T$ is in the cooling range DO
    3.1 While $H$ is decreasing DO
        3.1.1 Select a task $i$ at random.
        3.1.2 Compute mean fields of the spins at the $i$-th row
              $\phi_{ip} = -\sum_{j \neq i}^{N} \sum_{q \neq p}^{K} e_{i,j} s_{jq} d_{pq} - r \sum_{j \neq i}^{N} s_{jp} w_i w_j$    for $1 \leq p \leq K$
        3.1.3 Compute the summation $\sum_{p=1}^{K} e^{\phi_{ip}/T}$
        3.1.4 Compute new spin values at the $i$-th row
              $s_{ip}^{new} = e^{\phi_{ip}/T} / \sum_{p=1}^{K} e^{\phi_{ip}/T}$    for $1 \leq p \leq K$
        3.1.5 Compute the energy change due to these spin updates
              $\Delta H = \sum_{p=1}^{K} \phi_{ip}(s_{ip}^{new} - s_{ip})$
        3.1.6 Update the spin values at the $i$-th row
              $s_{ip} = s_{ip}^{new}$    for $1 \leq p \leq K$
    3.2 $T = \alpha \times T$
```

Figure 3: The proposed MFA algorithm for the mapping problem.

4.2 An Efficient Implementation Scheme

As is mentioned earlier, the MFA algorithm proposed for the mapping problem is an iterative process. The complexity of a single MFA iteration is mainly due to the mean field computations. In this section, we propose an efficient implementation scheme which reduces the complexity of the mean field computations, and hence the complexity of the MFA iteration, by asymptotical factors.

Assume that, i-th spin-row is selected at random for update in a particular iteration. The expression given for ϕ_{ip} (Eq. (9)) can be rewritten by changing the order of the first double summation as

$$\phi_{ip} = -\sum_{q \neq p}^{K} d_{pq} \sum_{j \neq i}^{N} e_{i,j} s_{jq} - r \sum_{j \neq i}^{N} s_{jp} w_i w_j = -\sum_{q \neq p}^{K} d_{pq} \lambda_{iq} - r\psi_{ip} \tag{15}$$

where

$$\lambda_{iq} = \sum_{j \neq i}^{N} e_{i,j} s_{jq} \tag{16}$$

$$\psi_{ip} = \sum_{j \neq i}^{N} s_{jp} w_i w_j \tag{17}$$

Here, λ_{iq} represents the increase in the interprocessor communication by mapping task i to a processor other then q (for the current mapping on processor q), assuming uniform unit communication cost between all pairs of processors in PCG. Similarly, ψ_{ip} represents the increase in the computational load balance cost associated with processor p, by mapping task i to processors p (for the current mapping on processor p).

For an efficient implementation, the overall mean field computations involved in a single iteration can be computed using the following matrix equation

$$\mathbf{\Phi_i} = -\mathbf{D} \times \mathbf{\Lambda_i} - r\mathbf{\Psi_i} = -\mathbf{\Theta_i} - r\mathbf{\Psi_i} \tag{18}$$

Here, $\mathbf{D}$ is a $K \times K$ adjacency matrix representing PCG (i.e. $D_{pq} = d_{pq}$), and $\mathbf{\Phi_i}$, $\mathbf{\Lambda_i}$ $\mathbf{\Psi_i}$ and $\mathbf{\Theta_i} = \mathbf{D} \times \mathbf{\Lambda_i}$ are column vectors with K elements, where

$$\begin{aligned} \mathbf{\Phi_i} &= [\phi_{i1}, \ldots, \phi_{ip}, \ldots, \phi_{iK}]^T \quad \mathbf{\Lambda_i} = [\lambda_{i1}, \ldots, \lambda_{ip}, \ldots, \lambda_{iK}]^T \\ \mathbf{\Psi_i} &= [\psi_{i1}, \ldots, \psi_{ip}, \ldots, \psi_{iK}]^T \quad \mathbf{\Theta_i} = [\theta_{i1}, \ldots, \theta_{ip}, \ldots, \theta_{iK}]^T \end{aligned} \tag{19}$$

The complexity analysis of the proposed implementation scheme for dense TIGs is as follows. Complexity of computing λ_{iq} and ψ_{ip} are both $O(N)$. Complexity of constructing $\mathbf{\Lambda_i}$ and $\mathbf{\Psi_i}$ vectors are both $O(N \times K)$, since both vectors contain K such entries. Complexity of computing the matrix-vector product required in Eq. (18) is $O(K^2)$. Hence, the overall complexity of computing the $\mathbf{\Phi_i}$ vector (Eq. (18)) reduces to $O(N \times K + K^2) = O(N \times K)$, since $N \gg K$ in general. The complexity of K spin updates and the computation of ΔH are both $O(K)$. Thus, the proposed scheme reduces the computational complexity of a single MFA iteration to $O(N \times K)$ for dense TIGs with $N \gg K$.

The complexity analysis of the proposed implementation for sparse TIGs is as follows. Note that, the sparsity of TIG can only be exploited in the computation of λ_{iq} values since

$$\lambda_{iq} = \sum_{j \in Adj(i)}^{N} e_{i,j} s_{jq} \tag{20}$$

for sparse TIGs. Hence, the complexity of computing an individual λ_{iq} is only $O(d_{avg})$. Thus, the complexity of constructing the $\mathbf{\Lambda_i}$ vector reduces to $O(d_{avg} \times K)$. The complexity of computing the $\mathbf{\Theta_i}$ vector in Eq. (18) reduces to $O(d_{avg} \times K + K^2)$. However, the complexity of constructing the $\mathbf{\Psi_i}$ vector required in Eq. (18) is $O(N \times K)$, dominating the overall complexity of the mean field computations. The complexity of computing the $\mathbf{\Psi_i}$ vector can be reduced if the computation of ψ_{ip} in Eq. (17) is re-formulated as

$$\begin{aligned} \psi_{ip} &= \sum_{j \neq i}^{N} s_{jp} w_i w_j = w_i \sum_{j \neq i}^{N} w_j s_{jp} = w_i (\sum_{j=1}^{N} w_j s_{jp} - w_i s_{ip}) \\ \psi_{ip} &= w_i (\gamma_p - w_i s_{ip}) \end{aligned} \tag{21}$$

where $\gamma_p = \sum_{j=1}^{N} w_j s_{jp}$. Here, γ_p represents the computational load of processor p, for the current mapping on processor p. Note that, computationally, γ_p represents the weighted sum of spin values of the p-th column of the spin matrix. At the beginning of the MFA algorithm, initial γ_p value for each column p $(1 \leq p \leq K)$ can be computed for the initial spin values. Then, γ_p values can be updated at the end of each iteration (i.e. after spin updates) using

$$\gamma_p^{new} = \gamma_p^{old} - w_i s_{ip}^{old} + w_i s_{ip}^{new} \qquad \text{for} \quad 1 \leq p \leq K \tag{22}$$

The computation of initial γ_p values can be excluded from the complexity analysis since they are computed only once at the very beginning of the algorithm. In this scheme, the computation of an individual ψ_{ip} using Eq. (21) is an $O(1)$ operation. Hence, the construction of the $\mathbf{\Psi_i}$ vector required in Eq. (18) becomes an $O(K)$ operation. Thus, the complexity of mean field computations involved in a single iteration reduces to $O(d_{avg} \times K + K^2)$. Note that, the update of an individual γ_p value (using Eq. (22)) at the end of each iteration is an $O(1)$ operation. Hence, the overall complexity of γ_p updates is $O(K)$ since K weighted column sums should be updated at each iteration. Complexity of spin updates and energy difference computation are also $O(K)$ for sparse TIGs. Hence, the implementation scheme proposed for sparse TIGs reduces the complexity of a single MFA iteration to $O(d_{avg} \times K + K^2)$.

5 Performance of Mean Field Annealing Algorithm

This section presents the performance evaluation of the Mean Field Annealing (MFA) algorithm for the mapping problem, in comparison with two well-known mapping heuristics; Simulated Annealing (SA) and Kernighan-Lin (KL). Each algorithm is tested using randomly generated mapping problem instances. Following sections briefly present the implementation details of these algorithms.

5.1 MFA Implementation

The MFA algorithm (Figure 3) described in Section 4 is implemented in order to evaluate its performance. Cooling process is started from an initial temperature which is found experimentally. It is not feasible to search for an initial temperature for each problem instance, as this process may take more time than solving the original problem. In order to avoid this, we performed experiments for only a small number of instances and chose an initial temperature which works for each one. For the mapping problem instances used in these experiments, initial temperature was found to be $T_0 = 5.0$. This value for T_0 is used for all 26 mapping problem instances involved in the experiments.

Coefficient r, which determines the balance between two optimization criteria of the mapping problem, is computed at the beginning of the MFA algorithm. After the spins are initialized randomly, r is computed using these initial spin values as

$$r = \frac{\sum_{i=1}^{N} \sum_{j \neq i} \sum_{p=1}^{K} \sum_{q \neq p} e_{ij} s_{ip} s_{jq} d_{pq}}{K \times \sum_{i=1}^{N} \sum_{j \neq i} \sum_{p=1}^{K} s_{ip} s_{jp} w_i w_j} \tag{23}$$

As is seen from the equation, r is used for balancing of the two summation terms in the cost function. Note that, r is inversely proportional to the number of processors.

At each temperature, iterations continue until $\Delta H < \epsilon$ for L consecutive iterations where $L = N$ initially. Parameter ϵ is chosen to be 0.5. Cooling process is realized in two phases; slow cooling followed by fast cooling, similar to the cooling schedules used for SA [18]. In the slow cooling phase, temperature is decreased using $\alpha = 0.9$ until T is less than $T_0/1.5$. Then, in the fast cooling phase, L is set to $L/4$ and α is set to 0.5 and cooling is continued until T is less then $T_0/5.0$. At the end of this cooling process, maximum spin values at each row are set to 1 and all other spin values are set to 0. Then the result is decoded as described in Section 4, and the resulting mapping is found. Note that, all parameters used in this implementation are either constants or found automatically. Hence, there is no parameter setting problem.

5.2 Kernighan-Lin Implementation

Kernighan-Lin heuristic is not directly applicable to the mapping problem since it was originally proposed for graph bipartitioning. The two phase approach is used to apply the KL heuristic to the mapping problem. In the first phase, TIG is partitioned into K clusters, where K is equal to the number of processors. These K clusters are then mapped to PCG using a one-to-one mapping heuristic in the second phase. One-to-one mapping heuristic used in this work is a variant of the KL heuristic.

For the clustering phase, Kernighan-Lin heuristic is implemented efficiently as described by Fiduccia and Mattheyses [8]. Two different schemes are utilized to apply KL to K-way graph partitioning. First scheme, partitioning by recursive bisection (KL-RB), recursively partitions the initial graph into two partitions until K partitions are obtained. Other scheme, partitioning by pairwise min-cut (KL-PM), starts with an initial K-way partitioning and then iteratively minimizes the cutsizes between each pair of partitions until no improvement can be achieved. In the KL heuristic, computational load balance is maintained implicitly by the algorithm. Vertex (task) moves causing intolerable load imbalances are not considered.

In the beginning of the second phase, K clusters formed in the first phase are mapped to the K processors of the multicomputer randomly. After this initial mapping, communication cost is minimized by performing a sequence of cluster swaps between processor pairs.

5.3 Simulated Annealing Implementation

The SA algorithm, implemented for solving the mapping problem, uses the one phase approach to map TIG onto PCG. In simulated annealing, starting from a randomly chosen initial configuration, configuration space is searched for the best solution using a probabilistic hill climbing algorithm. A configuration of the mapping problem is a mapping between TIG and PCG, which assigns each task in TIG to a processor in PCG. In order the search the configuration space, neighborhood of a configuration must be defined. For the implementation in this work, neighborhood of a configuration consists of all configurations which results with moving one vertex (task) of TIG from the maximum loaded node (processor) of PCG to any other node of PCG. At each iteration of the simulated annealing algorithm, one of the possible moves is chosen randomly as a candidate move. Then, the resulting decrease in the total communication cost caused by the candidate move is calculated without changing the configuration. If the candidate move decreases the cutsize, it is realized. If it increases the cutsize, then it is realized with a probability which decreases with the amount of increase in the total cutsize. Acceptance probabilities of the moves that increase the cost are controlled with a temperature parameter T which is decreased using an annealing schedule. Hence, as the annealing proceeds acceptance probabilities of uphill moves decrease. An automatic cooling schedule is used in the implementation of the SA algorithm [18].

5.4 Experimental Results

In this section, performance of the MFA algorithm is discussed in comparison with the SA and KL algorithms. These heuristics are experimented by mapping randomly generated TIGs onto mesh and hypercube connected multicomputers.

Six test TIGs are generated with $N = 200$ and 400 vertices. Vertices of these TIGs are weighted by assigning a randomly chosen integer weight between 1 and 10 to each vertex ($1 \leq w_i \leq 10$, for $1 \leq i \leq N$). Interaction patterns among the vertices of these TIGs are constructed as follows. A maximum vertex degree, d_{max}, is selected for each test TIG ($d_{max} = 8, 16, 32$), and degree d_i of each vertex i is randomly chosen between 1 and d_{max} (i.e. $1 \leq d_i \leq d_{max}$, for $1 \leq i \leq N$). Then, each vertex i of TIG is connected to d_i randomly chosen vertices. Resulting edges are weighted randomly with integer values varying between 1 and 10. These TIGs are mapped to 3-, 4-, 5-dimensional hypercubes and 4×4, 4×8 two dimensional meshes. PCGs corresponding to these interconnection topologies are constructed assuming software routing as is described in Section 2.

Tables 1, 2 and 3 illustrate the performance results of the KL-RB, KL-PM, SA and MFA heuristics for the generated mapping problem instances. In these tables, N and $|E|$ denote the number of vertices and edges in the test TIGs, respectively, and K denotes the number

Table 1: Total communication cost averages (and standard deviations) of the solutions found by the KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE COMMUNICATION COST			
N	$\|E\|$	K	T	KL-RB	KL-PM	SA	MFA
200	544	8	H	1807.4 (68.7)	1846.0 (56.2)	1595.1 (28.1)	1701.6 (45.7)
200	544	16	H	2819.9 (67.1)	2747.1 (83.5)	2180.0 (16.3)	2318.2 (35.3)
200	544	32	H	4098.8 (123.3)	4710.4 (102.0)	2881.1 (32.4)	2971.6 (43.4)
200	1120	8	H	5421.9 (56.2)	5494.7 (62.9)	4946.4 (34.7)	5215.8 (83.2)
200	1120	16	H	7742.4 (104.5)	7816.1 (86.4)	6699.1 (54.9)	7013.8 (25.5)
200	1120	32	H	10377.1 (136.7)	11280.2 (153.6)	8495.7 (99.0)	8893.1 (67.4)
200	2152	8	H	12721.6 (152.3)	12959.0 (101.0)	12018.5 (62.1)	12349.0 (208.0)
200	2152	16	H	17828.9 (142.5)	17959.9 (132.6)	16197.0 (79.6)	16519.4 (173.6)
200	2152	32	H	23127.6 (109.8)	24260.3 (131.3)	20393.7 (183.6)	20607.3 (220.1)
400	1227	8	H	4360.6 (69.9)	4444.5 (51.3)	3772.3 (21.9)	4526.9 (66.9)
400	1227	16	H	6096.0 (98.3)	6073.2 (55.8)	5086.4 (33.1)	6046.5 (101.2)
400	1227	32	H	8420.2 (109.1)	7999.9 (100.9)	6466.4 (47.6)	7641.3 (98.4)
400	2283	8	H	11247.1 (126.3)	11491.5 (129.8)	10152.1 (67.6)	10838.7 (60.2)
400	2283	16	H	15566.7 (142.1)	15896.9 (159.2)	13629.6 (46.9)	14591.6 (142.1)
400	2283	32	H	20543.8 (154.8)	20527.1 (203.1)	17199.1 (42.4)	18365.2 (99.5)
400	4298	8	H	25318.3 (164.9)	25832.1 (219.8)	23506.9 (82.9)	25052.7 (147.5)
400	4298	16	H	34590.6 (230.6)	35395.4 (173.9)	31417.7 (84.8)	33597.3 (215.4)
400	4298	32	H	45053.8 (286.2)	45098.1 (300.3)	39507.2 (105.3)	42249.0 (175.8)
200	544	16	M	3364.2 (122.0)	3318.7 (83.4)	2658.5 (53.2)	2726.6 (67.7)
200	544	32	M	5618.7 (217.8)	6822.5 (147.3)	4260.5 (34.4)	4134.3 (101.1)
200	1120	16	M	9234.2 (161.6)	9318.2 (176.1)	8432.3 (135.7)	7875.1 (65.6)
200	1120	32	M	14659.9 (163.4)	16485.4 (104.1)	13556.0 (221.0)	11710.6 (188.2)
400	1227	16	M	7341.4 (105.5)	7357.0 (174.6)	6295.3 (86.3)	7401.6 (131.2)
400	1227	32	M	12207.4 (246.6)	11758.6 (240.5)	9909.5 (80.0)	11619.0 (162.0)
400	2283	16	M	18670.9 (177.8)	19133.0 (200.6)	17484.4 (143.7)	16845.9 (48.9)
400	2283	32	M	29827.0 (375.9)	30156.3 (280.7)	28308.7 (251.5)	25208.3 (174.6)

of processors in the target PCG. Interconnection topology of the target POG is denoted by T, where H denotes the hypercube interconnection topology and M denotes the mesh interconnection topology. Each algorithm is executed 10 times for each problem instance starting from different, randomly chosen initial configurations. Averages and standard deviations of the results are illustrated in Tables 1, 2 and 3.

Tables 1 and 2 illustrate the quality of the solutions obtained by the KL-RB, KL-PM, SA and MFA heuristics. Total communication cost averages (and standard deviations) of the solutions are displayed in Table 1, and percent computational load imbalance averages (and standard deviations) are displayed in Table 2. Percent load imbalance for each solution is computed

Table 2: Percent computational load imbalance averages (and standard deviations) of the solutions found by the KL-RB, KL-PM, SA, MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE PERCENT LOAD IMBALANCE			
N	$\lvert E\rvert$	K	T	KL-RB	KL-PM	SA	MFA
200	544	8	H	10.2 (1.9)	8.4 (0.8)	2.7 (1.3)	4.5 (1.5)
200	544	16	H	15.1 (1.9)	8.4 (0.4)	7.5 (1.3)	9.4 (1.9)
200	544	32	H	18.8 (1.9)	8.9 (1.4)	16.2 (4.1)	18.8 (3.4)
200	1120	8	H	12.4 (1.4)	9.0 (0.3)	3.1 (1.0)	3.9 (0.5)
200	1120	16	H	16.1 (1.0)	8.4 (0.6)	7.9 (3.3)	9.2 (1.4)
200	1120	32	H	19.7 (2.7)	11.0 (4.6)	21.1 (4.5)	16.3 (3.8)
200	2152	8	H	13.3 (0.7)	9.2 (0.2)	3.7 (1.3)	5.9 (1.5)
200	2152	16	H	17.7 (0.5)	8.7 (0.0)	9.2 (1.7)	14.9 (2.3)
200	2152	32	H	22.5 (2.5)	8.7 (0.0)	18.3 (3.4)	28.5 (3.5)
400	1227	8	H	12.0 (1.2)	9.4 (0.3)	1.6 (0.4)	1.6 (0.5)
400	1227	16	H	14.1 (1.9)	9.6 (0.3)	3.7 (0.9)	2.5 (0.5)
400	1227	32	H	18.8 (1.0)	9.5 (0.3)	7.5 (1.3)	5.4 (0.9)
400	2283	8	H	13.0 (1.3)	9.5 (0.2)	1.7 (1.0)	2.2 (0.8)
400	2283	16	H	16.0 (1.6)	9.3 (0.3)	4.0 (0.9)	4.8 (1.1)
400	2283	32	H	20.4 (1.5)	8.6 (0.3)	8.5 (1.7)	7.8 (1.0)
400	4298	8	H	13.3 (1.2)	9.9 (0.2)	2.1 (0.8)	1.7 (0.4)
400	4298	16	H	16.4 (1.7)	9.4 (0.3)	4.8 (1.5)	4.0 (0.6)
400	4298	32	H	20.3 (2.2)	9.6 (0.2)	7.7 (2.5)	8.4 (1.3)
200	544	16	M	15.2 (1.2)	8.4 (0.4)	8.4 (2.2)	13.0 (2.1)
200	544	32	M	18.3 (2.0)	8.7 (0.0)	15.1 (2.7)	33.9 (3.9)
200	1120	16	M	16.3 (1.5)	8.5 (0.4)	8.5 (1.5)	16.6 (1.4)
200	1120	32	M	19.4 (2.0)	9.4 (2.2)	20.7 (4.2)	37.0 (2.4)
400	1227	16	M	16.1 (1.7)	9.6 (0.2)	3.4 (0.7)	3.4 (0.6)
400	1227	32	M	18.7 (1.7)	9.7 (0.2)	12.0 (4.1)	7.1 (0.7)
400	2283	16	M	16.0 (1.6)	9.4 (0.1)	5.3 (1.1)	10.6 (1.3)
400	2283	32	M	20.1 (1.6)	8.6 (0.2)	10.4 (1.5)	22.6 (1.9)

proportional to the computational load difference between maximum and minimum loaded processors. Table 3 displays the execution time averages of the KL-RB, KL-PM, SA and MFA heuristics. Table 4 is constructed for a better illustration of the overall performance of the MFA algorithm in comparison with the KL and SA heuristics. For each problem instance, results given in Tables 1, 2 and 3 are normalized with respect to the results of the MFA algorithm. The averages of the normalized results of Table 1, Table 2 and Table 3 constitute the first, second and fourth rows of Table 4, respectively. The average solution quality for each algorithm is computed using

$$\text{SOL'N QUALITY} \;=\; 1 \,/\, (\text{COMM. COST} \;+\; \text{LOAD IMBALANCE}) \tag{24}$$

Third row of Table 4, illustrates solution quality value of each algorithm normalized with respect

Table 3: Execution time averages (in seconds) of the KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE EXECUTION TIMES			
N	$\|E\|$	K	T	KL-RB	KL-PM	SA	MFA
200	544	8	H	1.1	5.7	67.3	2.3
200	544	16	H	1.5	13.7	127.2	5.6
200	544	32	H	3.3	29.6	245.1	15.6
200	1120	8	H	1.6	7.6	64.1	2.5
200	1120	16	H	2.2	14.6	144.0	6.6
200	1120	32	H	5.1	40.5	282.7	16.6
200	2152	8	H	2.5	10.9	64.2	2.2
200	2152	16	H	3.5	23.7	156.7	5.5
200	2152	32	H	7.6	45.4	373.9	14.0
400	1227	8	H	2.2	10.1	168.9	5.2
400	1227	16	H	3.0	29.7	310.7	12.0
400	1227	32	H	6.4	68.0	681.1	34.0
400	2283	8	H	3.3	16.0	167.1	5.6
400	2283	16	H	4.4	39.8	383.2	15.2
400	2283	32	H	8.6	88.9	632.8	42.0
400	4298	8	H	5.4	25.5	155.3	6.3
400	4298	16	H	7.1	64.9	403.0	15.4
400	4298	32	H	12.6	125.1	604.9	32.2
200	544	16	M	1.5	13.7	135.0	5.7
200	544	32	M	3.3	29.6	258.7	16.5
200	1120	16	M	2.3	14.8	124.2	5.7
200	1120	32	M	5.6	38.4	293.1	13.4
400	1227	16	M	3.1	26.7	280.5	10.9
400	1227	32	M	6.7	60.4	565.1	30.0
400	2283	16	M	4.4	41.7	363.8	13.3
400	2283	32	M	8.7	82.8	573.5	35.0

Table 4: Average performance measures of the KL-RB, KL-PM and SA heuristics normalized with respect to the MFA heuristic.

	KL-RB	KL-PM	SA	MFA
COMM. COST	1.114	1.148	0.957	1.0
LOAD IMBALANCE	2.718	1.714	0.875	1.0
SOL'N QUALITY	0.522	0.699	1.092	1.0
EXECUTION TIME	0.407	2.800	23.308	1.0

to the MFA algorithm.

As is seen in Tables 1, 2 and 4, the quality of solutions obtained by the MFA and SA heuristics are superior to those of the KL-RB and KL-PM heuristics. Solutions produced by SA are slightly better compared with the solutions produced by MFA, whereas the MFA algorithm is significantly faster (23 times on the average). As is seen in Table 3 and 4, average execution time of the MFA algorithm is comparable with that of the efficient KL heuristic. The MFA algorithm is 2.8 times faster than the KL-PM heuristic and 2.5 times slower than the KL-RB heuristic on the average. These results indicate that the proposed MFA algorithm is a promising alternative heuristic for solving the mapping problem.

6 Parallelization of Mean Field Annealing Algorithm

As is mentioned earlier, heuristic used for solving the mapping problem is a preprocessing overhead introduced for the efficient implementation of a given parallel program on the target multicomputer. If the mapping heuristic is implemented sequentially, this preprocessing can be considered in the serial portion of the parallel program which limits the maximum efficiency of the parallel program on the target machine. For a fixed parallel program instance, the execution time of the parallel program is expected to decrease with increasing number of processors in the target multicomputer. However, as is seen in Table 3, for a fixed TIG, the execution time of all mapping heuristics increase with increasing number of processors in the target multicomputer. Hence, the serial fraction of the parallel program will increase with increasing number of processors. Thus, this preprocessing will begin to constitute a drastic limit on the maximum efficiency of the overall parallelization due to Amdahl's Law. Hence, parallelization of these mapping heuristics on the target multicomputer is a crucial issue for efficient parallel implementations.

Unfortunately, parallelization of the mapping heuristics introduces another mapping problem. The computations of the mapping heuristics should be mapped to the processors of the same target architecture. However, in this case, the parallel algorithm for the mapping heuristic should be such that its mapping can be achieved *intuitively*. Furthermore, the intuitive mapping should lead to an efficient parallel implementation of the mapping heuristic. For these reasons, the target mapping heuristic to be parallelized should involve regular and inherently parallel computations. The MFA algorithm proposed in Section 4 for the general mapping problem has such nice properties for an efficient parallelization. Following paragraphs discuss the efficient parallelization of the proposed mapping heuristic for multicomputers.

Assume that, the MFA algorithm is used to map a given parallel program represented with a TIG having N vertices on a target multicomputer with K processors. The MFA algorithm will use an $N \times K$ spin matrix for the mapping operation. The question is to map the computations of the MFA algorithm to the same target multicomputer (with the same number of K processors). As is mentioned earlier, the MFA algorithm is an iterative algorithm. Hence, the mapping scheme can be devised by analyzing the computations involved in a particular iteration of the algorithm. Atomic task can be considered as the computations required for updating an individual spin. Note that, K spin averages at a particular row of the spin matrix are updated at each iteration. Hence, these K spin updates can be computed in parallel by mapping each spin in a row of the spin matrix to a distinct processor of the target architecture. Thus, the $N \times K$ spin matrix is partitioned column-wise such that each processor is assigned an individual column of the spin matrix. That is, column p of the spin matrix is mapped to processor p of the target architecture. Each processor is responsible for maintaining and updating the spin values in its local column. Assume that, task-i is selected at random in a particular iteration. Then, each processor is responsible for updating the probability of task i being mapped to itself.

A single iteration of the MFA algorithm can be considered as a three phase process, namely, mean field computation phase, spin update phase, and energy difference computation phase. Each processor p should compute its local mean field value ϕ_{ip} (Eq. (9) or Eq. (14)) in the first phase, in order to update its local spin value s_{ip} (Eq. (10)) in the second phase. As is mentioned earlier, mean field computation phase is the most time consuming phase of the MFA algorithm. Fortunately, mean field computations are inherently parallel since there are no interactions among the mean field computations involved in a particular iteration. However, a close look to Eq. (9) reveals that each processor needs most recently updated values of all spins except the ones in the i-th row in order to compute its local mean field value. Recall that, each processor maintains only a single column of updated spin values due to the proposed mapping scheme. Hence, this computational interaction necessitates *global* interprocessor communication just prior to the distributed mean field computation at each iteration. The volume of global interprocessor communication is proportional to $O(N \times K)$ for dense TIGs. As is seen in Eq.(14), the volume of global interprocessor communication is proportional to $O(d_{avg} \times K)$ for sparse TIGs. The volume of global interprocessor communication can be reduced to $O(K)$ for both dense and sparse TIGs by considering the parallelization of the matrix equation given in Eq. (18).

Eq. (18) involves the following operations : construction of the $\mathbf{\Lambda_i}$ and $\mathbf{\Psi_i}$ vectors, dense matrix vector product $\mathbf{\Theta_i} = \mathbf{D} \times \mathbf{\Lambda_i}$ and vector addition $\mathbf{\Phi_i} = -\mathbf{\Theta_i} - r\mathbf{\Psi_i}$. Note that, each processor p only needs to compute the p-th entry θ_{ip} of the $\mathbf{\Theta_i}$ vector, and the p-th entry ψ_{ip} of the $\mathbf{\Psi_i}$ vector in order to compute its local mean field value ϕ_{ip} in parallel. The matrix vector product can be performed in parallel by employing the *scalar accumulation* (SA-MVP) scheme. In this scheme, each processor needs only the p-th row $\mathbf{d_p}$ of the dense $\mathbf{D}$ matrix and the whole column vector $\mathbf{\Lambda_i}$.

Each processor p can concurrently compute the p-th entry λ_{ip} of the $\mathbf{\Lambda_i}$ vector using Eq. (16) or Eq. (20) without any interprocessor communication. Note that, q in these equations should be replaced by p in these computations. Then, a global collect (GCOL) operation is required for each processor to obtain a local copy of the $\mathbf{\Lambda_i}$ vector. The GCOL operation is essentially appending K local scalars, in order, into a vector of size K and then duplicating this vector in the local memory of each processor. The GCOL operation requires global interprocessor communication. Note that, only K local spin values should be collected globally thus reducing the volume of communication during the GCOL operation by an asymptotical factor.

After the GCOL operation, each processor has a local copy of the global $\mathbf{\Lambda_i}$ vector. Hence, each processor p can concurrently compute its local θ_{ip} by performing the inner-product $\theta_{ip} = \mathbf{d_p} \times \mathbf{\Lambda_i}$. Then, each processor p should compute the p-th entry ψ_{ip} of the $\mathbf{\Psi_i}$ vector. Note that, each processor p already maintains the γ_p^{old} value. Hence, each processor can concurrently compute ψ_{ip} using Eq. (21). Then, each processor p can concurrently compute its local mean field value ϕ_{ip} by performing the local computation $\phi_{ip} = -\theta_{ip} - r\psi_{ip}$. Note that, these computations are completely local computations and involve no interprocessor communication.

The second phase of an individual MFA iteration is highly sequential since global interaction exists among spin updates due to the normalization process indicated by Eq. (10). Fortunately, this global interaction can be relieved by noting the independent exponentiation operations involved in the numerator of Eq. (10). Hence, each processor p can concurrently compute its local $e^{\phi_{ip}/T}$ value. Then, a global sum (GSUM) operation is required for each processor to obtain a local copy of the global sum of the local exponentiation results. The GSUM operation requires *global* interprocessor communication. After the GSUM operation each processor p can concurrently update its local spin value by computing Eq. (10). After computing s_{ip}^{new}, each processor p should concurrently update its local γ_p values according to Eq. (22) for the use in the next iteration.

In the third phase, each processor should compute the same local copy of the global energy difference ΔH for global termination detection. Each processor p can concurrently compute its local energy difference $\Delta H_{ip} = \phi_{ip}\Delta s_{ip} = \phi_{ip}(s_{ip}^{new} - s_{ip}^{old})$ due to its local spin update. Then, a GSUM operation, which requires global interprocessor communication, is required for each processor to compute a local copy of the global sum $\Delta H = \sum_{p=1}^{K} \Delta H_{ip}$.

Hence, the proposed parallel MFA algorithm necessitates three global communication operations due to the GCOL operation involved during the first phase and two GSUM operations involved in the second and third phases. In fine grain multicomputers, the volume of interprocessor communication is the important factor in predicting the complexity of the interprocessor communication overhead. However, in medium grain multicomputers, the number of communications is also important since high *set-up* time overhead is associated with each communication step. The set-up time is the dominating factor for short messages in such architectures. Note that, only a single floating-point variable, representing the running sum, is communicated during the GSUM operations involved in the last two phases of the parallel MFA algorithm.

Reducing the number of GSUM operations required in the MFA algorithm will be a valuable asset in achieving efficient implementations on medium grain multicomputers. As seen in Eq. (13), there is an execution dependency between the computation of the energy difference ΔH and spin-row updates. This execution dependency between the second and the third phase computations can be relieved by rewriting the expression for ΔH as follows

$$\begin{aligned}\Delta H &= \sum_{p=1}^{K} \phi_{ip}(s_{ip}^{new} - s_{ip}^{old}) = \sum_{p=1}^{K} \phi_{ip} s_{ip}^{new} - \sum_{p=1}^{K} \phi_{ip} s_{ip}^{old} \\ &= \sum_{p=1}^{K} \phi_{ip} \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^{K} e^{\phi_{iq}/T}} - \sum_{p=1}^{K} \phi_{ip} s_{ip}^{old} = \frac{1}{A_i} \sum_{p=1}^{K} \phi_{ip} e^{\phi_{ip}/T} - C_i = \frac{B_i}{A_i} - C_i \qquad (25)\end{aligned}$$

where $A_i = \sum_{p=1}^{K} e^{\phi_{ip}/T} = \sum_{p=1}^{K} a_{ip}$, $B_i = \sum_{p=1}^{K} \phi_{ip} e^{\phi_{ip}/T} = \sum_{p=1}^{K} b_{ip}$ and $C_i = \sum_{p=1}^{K} \phi_{ip} s_{ip}^{old} = \sum_{p=1}^{K} c_{ip}$. Hence, after each processor p computes its local $a_{ip} = e^{\phi_{ip}/T}$, $b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$ and $c_{ip} = \phi_{ip} s_{ip}^{old}$ values, three global summations $A_i = \sum_{p=1}^{K} a_{ip}$, $B_i = \sum_{p=1}^{K} b_{ip}$ and $C_i = \sum_{p=1}^{K} c_{ip}$ can be accumulated in a *single* GSUM operation. After this single GSUM operation, each processor p can concurrently update its local spin value and compute the same local copy of the global energy difference as $s_{ip} = a_{ip}/Ai$ and $\Delta H = B_i/A_i - C_i$, respectively. Note that, this scheme reduces the number of GSUM operation from two to one. Three floating point variables, representing the running sums A_i, B_i, and C_i, are communicated during the communications involved in the GSUM operation.

The node program (of processor p, for $1 \leq p \leq K$) for a single iteration of the parallel MFA algorithm proposed for solving the mapping problem is given in Figure 4. Note that, variables with "*ip*" and "*p*" subscripts denote the local variables. Variables with "*i*" subscripts denote the global variables which are constructed and duplicated at the local memory of each processor after performing the indicated global operations. As is seen in Figure 4, the proposed parallel MFA algorithm exhibits very regular computational structure even for mapping arbitrarily irregular TIGs. The communication structure is also very regular since it necessitates only GSUM and GCOL operations. Hence, the proposed parallel MFA algorithm can easily be implemented on both MIMD and SIMD types of multicomputers.

The parallel communication complexity of a single MFA iteration can be analyzed as follows. The interconnection schemes used in the processor organization of the multicomputers are usually symmetric in nature (i.e. POG is symmetric). Hence, GSUM and GCOL type global operations in such architectures are usually performed by a sequence of concurrent communication steps. Each communication step, involves concurrent single-hop communications. The number of concurrent single-hop communications is proportional to the diameter of POG for both GSUM and GCOL operations. For example, diameters of hypercube and mesh POGs are $log_2 K$ and $K^{1/2}$, respectively. The overall volume of concurrent interprocessor communications

```
1. Select a task $i$ at random.
2. Compute $\lambda_{ip} = \sum_{j \in Adj(i)} e_{ij} s_{jp}$
3. Perform GCOL operation to obtain a local copy of
       $\mathbf{\Lambda_i} = [\lambda_{i1}, \ldots, \lambda_{ip}, \ldots, \lambda_{iK}]^T$
4. Compute the inner product $\theta_{ip} = \mathbf{d_p}^T \times \mathbf{\Lambda_i}$
5. Compute $\psi_{ip} = w_i(\gamma_p - w_i s_{ip})$
6. Compute the local mean field value $\phi_{ip} = -\theta_{ip} - r\psi_{ip}$
7. Compute $a_{ip} = e^{\phi_{ip}/T}$, $b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$ and $c_{ip} = \phi_{ip} s_{ip}$
8. Perform GSUM to compute the local copies of
       $A_i = \sum_{p=1}^{K} a_{ip}$    $B_i = \sum_{p=1}^{K} b_{ip}$    and    $C_i = \sum_{p=1}^{K} c_{ip}$
9. Compute $s_{ip}^{new} = a_{ip}/A_i$ and then $\Delta s_{ip} = s_{ip}^{new} - s_{ip}$
10. Compute $\Delta H = B_i/A_i - C_i$
11. Update $\gamma_p = \gamma_p + w_i \Delta s_{ip}$
12. Update $s_{ip} = s_{ip}^{new}$
```

Figure 4: Node program (of processor p, for $1 \leq p \leq K$) for one iteration of the parallel MFA algorithm for the mapping problem.

is proportional to the diameter and the number of processors (K) of POG for GSUM and GCOL operations, respectively.

As is seen in Figure 4, the proposed parallel MFA algorithm achieves perfect load balance. The parallel computational complexity of a single MFA iteration can be obtained as follows. During the parallel computation of λ_{ip} values (step 2) each processor performs $N-1$ (d_i) multiplication/addition operations for dense (sparse) TIGs. Here, d_i denotes the degree of vertex i in TIG. During the parallel SA-MVP computation (step 4), each processor performs K multiplication/addition operations for both dense and sparse TIGs since the $\mathbf{D}$ matrix is a dense matrix. Each processor performs the same constant amount of arithmetic operations in the remaining steps (steps 5-7 and steps 9-12). Hence, the parallel computational complexity of the proposed algorithm is $O(N + K)$ and $O(d_{avg} + K)$ for dense and sparse TIGs respectively. Hence, linear speed-up can easily be achieved if communication overhead remains negligible. Note that, the number of concurrent communications increases with the diameter of POG (e.g. $log_2 K$, $K^{1/2}$), whereas, computational granularity per processor increases with the number of processors (K) of POG. Hence, percent communication overhead will reduce with increasing number of processors. Thus, the proposed parallel algorithm is expected to scale even on medium-to-coarse grain multicomputers.

7 Conclusion

In this paper, recently proposed Mean Field Annealing (MFA) algorithm is formulated for the mapping problem. An efficient implementation scheme is also developed for the proposed algorithm. The performance of the proposed algorithm is evaluated in comparison with two well known heuristics (Simulated Annealing (SA) and Kernighan-Lin (KL)) for a number of randomly generated mapping problem instances. The qualities of the solutions obtained by the MFA and SA heuristics are found to be superior to the qualities of the solutions obtained by the KL heuristic. Execution time of the MFA algorithm is comparable to that of the efficient KL heuristic. The SA heuristic produces slightly better solutions than the MFA algorithm, whereas MFA is significantly faster. An efficient parallel algorithm is also developed for the proposed MFA heuristic.

References

[1] Arora, R. K., and Rana, S. P., Heuristic algorithms for process assignment in distributed computing systems. *Information Processing Letters,* vol. 11, no. 4-5, pp. 199-203, 1980.

[2] Aykanat, C., Özgüner, F., Erçal, F., and Sadayappan, P. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transactions on Computers,* vol. 37, no. 12, pp. 1554-1567, 1988.

[3] Behnam-Guilani, K. Fast decoupled load flow: the hybrid model. *IEEE Transactions on Power Systems,* vol. 3, no. 2, pp. 734-739, 1988.

[4] Bokhari, S. H. On the mapping problem. *IEEE Trans. Comput.,* vol. 30, no. 3, pp. 207-214, 1981.

[5] Bultan, T., and Aykanat, C. Parallel mean field algorithms for the solution of combinatorial optimization problems. *Proc. ICANN-91,* vol. 1, pp. 591-596, 1991.

[6] Bultan, T., and Aykanat, C. Circuit Partitioning Using Parallel Mean Field Annealing Algorithms. *Proc. 3rd IEEE Symposium on Parallel Processing,* 1991.

[7] Erçal, F., Ramanujam, J., and Sadayappan, P. Task allocation onto a hypercube by recursive mincut bipartitioning. *J. Parallel Distrib. Comput.* vol. 10, pp. 35-44, 1990.

[8] Fiduccia, C. M., and Mattheyses, R. M. A linear heuristic for improving network partitions. *Proc. Design Automat. Conf.,* pp. 175-181, 1982.

[9] Hopfield, J. J., and Tank, D. W. 'Neural' Computation of Decisions in Optimization Problems. *Biolog. Cybern.,* vol. 52, pp. 141-152, 1985.

[10] Hopfield, J. J., and Tank, D. W. Computing with neural circuits: a model. *Science,* Vol. 233, pp. 625-633, August 1986.

[11] Hopfield, J. J., and Tank, D. W. Collective computation in neuronlike circuits, *Scientific American,* vol. 257, no. 6, pp. 104-114, 1987.

[12] Indurkhya, B., Stone H. S., and Xi-Cheng, L. Optimal partitioning of randomly generated distributed programs. *IEEE Trans. Software Engrg.,* vol. 12, no. 3, pp. 483-495, 1986.

[13] Kasahara, H., and Narita, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.,* vol. 33, no. 11, pp. 1023-1029, 1984.

[14] Kernighan, B. W., and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.,* vol. 49, pp. 291-307, 1970.

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

[16] Lee, S. Y., Chiang, H. D., Lee, K. G., and Ku, B. Y. Parallel power system transient stability analysis on hypercube multiprocessors. *IEEE Transactions on Power Systems,* vol. 6, no. 3, pp. 1337-1343.

[17] Peterson, C., and Anderson, J. R. Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem. *Complex Syst.* vol. 2, pp. 59-89, 1988.

[18] Peterson, C., and Soderberg, B. A new method for mapping optimization problems onto neural networks. *Int. J. Neural Syst.*, vol. 1, no. 3, 1989.

[19] Ramanujam, J., Erçal, F., and Sadayappan, P. Task allocation by simulated annealing. *Proc. International Conference on Supercomputing.* Boston, MA, May 1988, vol. III, *Hardware & Software,* pp. 475-497.

[20] Sadayappan, P., and Erçal, F.Nearest-neighbour mapping of finite element graphs onto processor meshes. *IEEE Trans. Comput.* vol. 36, no. 12, pp. 1408-1424, 1987.

[21] Sadayappan, P.,Erçal, F., and Ramanujam, J. Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing.* vol. 13, pp. 1-16, 1990.

[22] Shield, J. Partitioning concurrent VLSI simulation programs onto a multiprocessor by simulated annealing. *IEEE Proc. Part G,* vol. 134, no. 1, pp. 24-28, 1987.

[23] Van den Bout, D. E., and Miller, T. K. A Traveling Salesman Objective Function That Works. *IEEE Int. Conf. Neural Nets,* vol. 2, pp. 299-303, 1988.

[24] Van den Bout, D. E., and Miller, T. K. Improving the performance of the Hopfield-Tank neural network through normalization and annealing. *Biolog. Cybern.*, vol. 62, pp. 129-139, 1989.

[25] Van den Bout, D. E., and Miller, T. K. Graph partitioning using annealed neural networks. *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 192-203, 1990.