MaTCH : Mapping Data-Parallel Tasks on a Heterogeneous Computing Platform Using the Cross-Entropy Heuristic

Soumya Sanyal and Sajal K. Das
Dept. of Computer Science & Engineering
The University of Texas at Arlington
Arlington, TX 76019-0015
{sanyal,das}@cse.uta.edu

Abstract

*We propose in this paper a heuristic for mapping a set of interacting tasks of a parallel application onto a heterogeneous computing platform such as a computational grid. Our novel approach is based on the Cross-Entropy (CE) method, which is a new and extremely robust rare event simulation (RES) technique. We tailor the CE method to the requirements of the problem at hand, develop a mathematical framework, and present our algorithm, called **MaTCH**. This globally iterative, randomized procedure is then compared to a previously developed genetic algorithm (GA). Our preliminary experiments demonstrate the power of **MaTCH**, exhibiting remarkable improvements in the quality of mapping. The results indicate that, when compared to the GA, the proposed heuristic **MaTCH** improves upon the application execution time by over a factor of 38 on a 50-node resource graph. We further attest our results by performing an ANalysis Of VAriance (ANOVA) test on a sample data set to prove the significance of our results.*

1. Introduction

The problem of mapping or scheduling a set of tasks over a heterogeneous computing platform such as a computational grid [17], has been a topic of active research for a number of years. The objective is to obtain an assignment of tasks of an application modeled as a *Task Graph*, to a set system resources (processors) modeled as a *System* (or *Resource Graph*), such that the application execution time after mapping is minimized. Various algorithms and heuristics have been successfully employed to solve this problem. Specifically, [5, 7, 18, 3, 6] talk about applications where the tasks are independent and do not need communication - parameter sweep applications is an example [11]. Other types include parallel applications represented as dependent (directed) task graphs modeling classical parallel applications [14, 15]. Our interest in this paper is mapping problem of a class of data parallel applications found in the *overset grid* [10] methodology found in Section 2. These applications can be modeled as undirected *Task Interaction Graphs* (TIGs). There exists limited body of work on this specific problem. This assignment can either be done through a partitioning approach [9, 20] or simple clustering and mapping techniques [2, 10, 16, 25]. No matter what the nature of the tasks, the problem is generally NP-Hard [4]. While the formulation of the problem dealt with in this paper is not new, we present an elegant solution that is proven to be more effective than a genetic algorithmic approach (GA) [16]. We use the formulation of the GA as our benchmark. This is because in [5], the GA approach was proved to be the best overall heuristic for statically mapping tasks. We develop our algorithm, called **MaTCH** (**Ma**pping **T**asks using the **C**ross-Entropy **H**euristic) using the basis of the cross-entropy (CE) method [22] which may be employed to solve difficult combinatorial optimization problems (COPs).

We use a simple set of experiments and simplify the problem by allowing the number of tasks and resources to be equal. This was done to indicate a proof of concept rather than show through practical implementation what can be achieved. We show that even for a small number of resources. Specifically, our algorithm vastly outperforms the GA method. Our results show an application execution time speedup of over 38 on a 50-node resource graph. This conclusively proves the efficacy of the technique for a sizeable number of resources. In brief, our contributions in this paper include:

- Developing the mathematical framework for the heterogeneous mapping problem and defining the objectives.
- Adopting the CE method and formulating our algorithm, **MaTCH**, to map a set of heterogeneous tasks to resources.

0-7695-2312-9/05/$20.00 (c) 2005 IEEE

- Designing the first mapping/scheduling scheme that uses the CE method to its advantage.

The rest of the paper is organized as follows. Section 2 briefly discusses the nature of the problem and formulates the objective function we wish to optimize. Section 3 introduces the notion of the CE method and outlines the generic CE based algorithm. Section 4 describes the **MaTCH** heuristic, while Section 5 presents the simulation results. Section 6 concludes the paper.

2. Application Requirements and the Mapping Problem

This paper focuses on applications that are similar in nature to the *overset grid* CFD (Computational Fluid Dynamics) problems [10], that deal with the estimation of the viscous drag of an irregularly shaped body. Typically, the domain around an arbitrary three-dimensional body is approximated by regularly shaped *grids* (not to be confused with computational grids). Each of these *grids* is composed of thousands of *grid points* where computations occur and who communicate amongst themselves. Our task interaction graph (TIG) model $G_t = (V_t, E_t)$, assumes each *grid* to be a node of V_t. Each vertex $v_i \in V_t$ has a *computational weight* W^i, which represents of the number of *grid* points it contains. Since regularly-shaped *grids* cannot exactly cover an irregular surface, they overlap in space, creating what is known as *overset grids* [10]. Hence, an edge between vertices v_i and v_j denotes adjacent *grids* that overlap. Each edge $(v_i, v_j) \in E_t$ has a *communication weight* $C^{i,j}$, which reflects the number of *grid* points that overlap. The notion of edges in this case can also be thought of as data-processing pipelines where *grids* exchange information. Figure 1 shows an abstraction of four *overset grids* and its corresponding TIG.

The system resources are represented by a weighted undirected graph $G_r = (V_r, E_r)$, which we refer to as the *resource graph*. It consists of a set of vertices $V_r = \{r_1, r_2, \ldots, r_n\}$, and a set of edges $E_r = \{(r_i, r_j) \mid r_i, r_j \in V_r\}$. Each resource (vertex) r_i has a *processing weight* w_i, modeling its processing cost per unit of computation. Each link (edge) has a *link weight* $c_{i,j}$, that denotes the cost per unit of communication between resources r_i and r_j.

The execution time of a parallel application consists of *processing time* and *communication time*. In general, the processing time of task v_t on resource $r_s \in V_r$ is $T_{cp}[t,s] = W^t \times w_s$. The communication time for v_t is $T_{cm}[t,s] = \sum_{v_a \mathcal{A} r_b,\ r_b \neq r_s} C^{t,a} \times c_{s,b}$ where $v_a \mathcal{A} r_b$ denotes that vertex v_a, which is a neighbor of v_t, has been assigned to resource r_b.

Let χ define the set of all possible mappings. Hence the execution time of r_s for a particular mapping $\mathcal{M}$, denoted as $\chi_{\mathcal{M}}$, is:

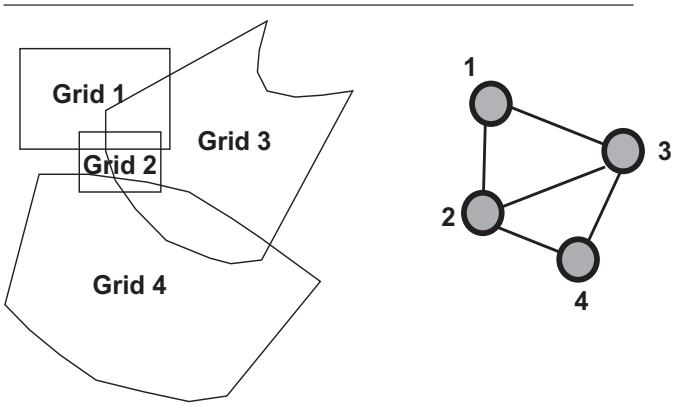

Figure 1. A set of overset grids and its corresponding TIG.

$$Exec_s^{\chi_{\mathcal{M}}} = \sum_{v_t \mathcal{A} r_s} W^t \times w_s + \sum_{v_t \mathcal{A} r_s} \sum_{v_a \mathcal{A} r_b,\ r_b \neq r_s} C^{t,a} \times c_{s,b} \quad (1)$$

and the total application execution time is

$$Exec^{\chi_{\mathcal{M}}} = \max_{r_s} \{Exec_s^{\chi_{\mathcal{M}}}\},\ \ r_s \in V_{R_i}^k. \quad (2)$$

Our objective is to find the mapping $\chi_{\mathcal{M}}$ that minimizes $Exec^{\chi_{\mathcal{M}}}$.

3. A Brief Introduction to Cross-Entropy

The Cross Entropy (CE) Method pioneered by Rubenstein [21] in 1997 was originally formulated as an adaptive algorithm for estimating probabilities of rare events. Since then, it has been broadened as a generic and efficient tool for solving a myriad of deterministic and stochastic combinatorial optimization problems (COPs) and continuous multi-extremal optimization problems [1, 23, 24]. The advantage of the CE method over other methods in tackling difficult combinatorial optimization problems is that it defines a precise mathematical framework for deriving fast optimal (or near optimal) learning rules based on some advanced simulation theory. Our intention is to present this method in the light of the GA based mapping developed in [16]. It is worthwhile to note that, compared to GAs or other randomized meta-heuristics, the CE method is a global search mechanism and hence the chances of finding a better solution of the COP is enhanced [8].

As we mentioned earlier, let the entire solution space of our mapping problem to be χ. We wish to find optimal points on this space that will minimize the application execution time. The CE method describes a fundamental way of viewing each solution as a distribution. So, our job at

hand is to find one such distribution, i.e., a mapping, that will provide us the minimum application execution time. The basic method of doing so is to generate a set of unknown parameterized distributions (mappings), also called *trajectories*, by selecting valid solutions or distributions applicable to our case. Next, we measure the performance of all such mappings by the objective or performance function S (or $Exec^{\chi,\mathcal{M}}$) defined in the previous section. If we take a statistical view of this set of samples and choose the $(1-\rho)th$ quantile measure of this set, we will end up with a smaller set of sample mappings of higher quality than the rest. The idea then is to update the current set of parameters we originally had by measuring how far off we were from the elite samples in the first place. Therefore, in the next iteration we hope to generate better solutions by using these updated set of parameters. We repeat this procedure iteratively. The CE method guarantees that we will come closer to the optimal solution so long as this method is employed. It is important to note though that χ contains a huge number of solutions and hence the method in theory can only be implemented (in a stochastic fashion) if we start off with a limited number of samples. Thus, abstractly speaking the CE algorithm consists of basically two steps, repeated in an iterative fashion:

- Generating random data samples or solutions for the problem at hand using the current dynamic set of parameters.
- Updating the parameters that govern the random data generation using the data samples themselves with the objective of improving the future data samples.

The remainder of this section explains the approach from a mathematical perspective. In keeping with the true spirit of the original authors, the notations and conventions in this section have been borrowed from [8].

In our COP, χ denotes the set of states (all mappings), and $S(.)$ denotes a real-valued *performance function* (the objective function) that we wish to minimize over χ and consequently find the states at which this is obtained. Suppose, $\mathbf{X} = X_1, X_2, \ldots, X_n$ is a random vector taking values in χ. We define $f(.;\mathbf{v})$ as the *probability mass function* (pmf) on χ, parameterized by a real-valued parameter vector $\mathbf{v}$. Thus the expected value of $S(.)$ denoted by the expectation operator $\mathbb{E}[.]$ is,

$$\mathbb{E}_{\mathbf{v}}[S(\mathbf{X})] = \sum_{\mathbf{X}} S(X_i) f(\mathbf{X};\mathbf{v})$$

Denoting by γ^*, the maximum value obtained using $S(.)$, we get

$$\gamma^* = \max_{X_i \in \mathbf{X}} S(X_i) \tag{3}$$

For a certain parameter vector $\mathbf{u}$, the probability $\mathbb{P}$ that $S(\mathbf{X})$ is greater than a threshold γ, is given by:

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) > \gamma) = \sum_{\mathbf{X}} I_{\{S(X_i)>\gamma\}} f(\mathbf{X};\mathbf{u}) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})>\gamma\}} \tag{4}$$

Here, $\{I_{\{S(\mathbf{X})>\gamma\}}\}$ is defined to be a set of *indicator* functions on χ for various thresholds $\gamma \in \mathbb{R}$. This is the starting point in the CE method. These indicator function(s) act as *estimators*. Hence, we associate an *estimation problem* with the optimization problem. An intuitive way to connect eq.(3) and eq.(4) is to assume $\gamma = \gamma^*$ and allow $f(.;\mathbf{u})$ to be a uniform pmf in χ. In a typical COP, $|\chi|$ is very large and $\ell(\gamma^*) = f(\mathbf{X}^*;\mathbf{u}) = 1/|\chi|$ is therefore a very small value. This would cast the COP problem as a *rare-event*, bringing into effect the original CE method proposed to solve *rare event simulation* (RES) problems [22].

The most common approach taken to estimate this parameter $\ell(\gamma^*)$ is by using a *likelihood ratio* (LR) estimator with reference parameter $\mathbf{v}^*$ given by

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}} \ \left[\mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})>\gamma\}}\right] \ln f(\mathbf{X};\mathbf{v}) \tag{5}$$

This estimator measures the distance between two densities and is called the *Kullback-Leibler* cross-entropy distance metric. In essence, we are finding out how much information difference exists between two densities. The parameter could be estimated stochastically using,

$$\widehat{\mathbf{v}^*} = \underset{\mathbf{v}}{\operatorname{argmax}} \ \left[\frac{1}{N}\sum_{i=1}^{N} I_{\{S(X_i)>\gamma\}}\right] \ln f(\mathbf{X};\mathbf{v}) \tag{6}$$

where X_i is generated from the pmf $f(.;\mathbf{u})$.

It is worthwhile to note that as γ approaches γ^*, the pmf $f(.;\mathbf{v}^*)$ assigns a major chunk of the vectors close to $\mathbf{x}^*$ and therefore can serve as an approximate solution to eq.(3). Next, the estimator can only be evaluated in a practical manner if $I_{\{S(\mathbf{X})>\gamma\}} = 1$ for a reasonable amount of samples. This indicates a close relationship between $\mathbf{u}$ and γ. As γ approaches γ^* we would like $\mathbf{u}$ to be such that $\mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) > \gamma)$ is a reasonable value. Hence, our intention is to obtain γ as close to γ^* and at the same time produce a large value of it to ensure we obtain an accurate estimator for $\mathbf{v}^*$. The usual way to achieve this is to approach the problem from a multi-level perspective. In an iterative fashion we proceed to update the sequence of levels $\widehat{\gamma}_1, \widehat{\gamma}_2, \ldots, \widehat{\gamma}_T$ and simultaneously update the parameter vectors $\widehat{\mathbf{v}}_1, \widehat{\mathbf{v}}_2, \ldots, \widehat{\mathbf{v}}_T$ till a terminating condition is reached. The desired end result is a $\widehat{\gamma}_T$ close to γ^* and the parameter vector $\widehat{\mathbf{v}}_T$ such that $f(.;\widehat{\mathbf{v}}_T)$ assigns most of the probability mass to states close to the optimal state $\mathbf{x}^*$, giving a high performance.

For example, if we look at Figure 3, our parameter vectors are replaced by stochastic matrices. This is because the

formulation of the mapping problem in terms of the CE method made it to be so. We notice that as the iterations proceed, the stochastic matrix evolves until finally it results in a degenerate matrix. This gives us a visual idea of how the CE method converges to a degenerate solution.

Having discussed the basic concepts, we present in Figure 2 what is considered as a generic algorithm based on CE method.

Probabilistic CE Algorithm for COPs
1. Define $\widehat{\mathbf{v}}_0 = \mathbf{u}$; $i = 1$
2. Generate samples $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from the density f(.;$\mathbf{v}_{i-1}$) and compute the sample (1-ρ)th-quantile of $\widehat{\gamma}_i$ of the performances that is, $\widehat{\gamma}_i = S_{(\lceil (1-\rho) \rceil)}$ given that $\widehat{\gamma}_i$ is less than γ. Else, set $\widehat{\gamma}_i = \gamma$.
3. Use the sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ to solve the stochastic program: $\widehat{\mathbf{v}}_i = \text{argmax}_{\ \mathbf{v}} \frac{1}{N} \sum_{j=1}^{N} I_{\{S(\mathbf{X}_j) > \widehat{\gamma}_i\}} \ln f(\mathbf{X}_j; \mathbf{v})$
4. If for some $i > k$: $\widehat{\gamma}_i = \widehat{\gamma}_{i-1} = \ldots = \widehat{\gamma}_{i-k}$ then **stop**; Otherwise, set $i = i + 1$ and reiterate from Step 2.

Figure 2. Basic CE algorithm for optimization.

4. The MaTCH algorithm

We formulate a CE method for the task mapping problem on heterogeneous grids. Given the generic framework for any CE algorithm, specific to our mapping problem we must explain how

- To generate random mappings, i.e., the family of pmf's $f(.; \mathbf{v})$
- To update the parameters at each iteration, i.e., provide the updating rules.

A recapitulation of the mapping problem conveys that we are interested in mapping a set of tasks V_t modeled as an application graph $G_t = (V_t, E_t)$ onto a set of resources $G_r = (V_r, E_r)$. Specific to our case, $|V_t| = |V_r|$. A few simple modifications of the algorithm(s) explained later will in effect take care of other cases. While in general there are many ways to generate samples from χ, the most intuitive way to explain how the random mappings are generated and how the parameters are updated subsequently, is to relate the mapping problem explained in Section 2, to an equivalent minimization problem. Specifically, let $\widetilde{\chi} = \{\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_N\} : \mathbf{X}_i = X_0, X_1, \ldots, X_{|V_r|-1}$ define the set of all possible mappings on $|V_r|$ resources, and define a function $\widetilde{S}$ on $\widetilde{\chi}$, where $\widetilde{S}$ denotes the maximum execution time of the busiest resource in that particular mapping. Then, $\widetilde{S}(\mathbf{x}) = S(\mathbf{x})$ if $\mathbf{x} \in \widetilde{\chi}$, and $\widetilde{S}(\mathbf{x}) = \infty$ otherwise. Therefore, we could write the optimization problem as

$$\min \widetilde{S}(\mathbf{x}) \quad \forall \quad \mathbf{x} \in \widetilde{\chi} \tag{7}$$

The most naive way to generate the random vector $\mathbf{X} = \{X_1, X_2, \ldots, X_{|V_r|-1}\}$ in $\widetilde{\chi}$ is to independently draw $\{X_1, X_2, \ldots, X_{|V_r|-1}\}$ according to fixed distributions $(p_{i0}, p_{i1}, \ldots, p_{i|V_r|-1})$, for $i = 1, \ldots, |V_r| - 1$. By amalgamating these p_{ij}'s we can get a stochastic matrix $P = (p_{ij})$ of the dimension $(|V_r| - 1) \times (|V_r| - 1)$. Since each of the rows in this matrix theoretically is a markov chain, they have the normalizing condition that each sum up to 1. This is because the sum total probability of a task being mapped to any resource is obviously 1. Thus, our pmf is $f(.; P)$ of $\mathbf{X}$ parameterized by the stochastic matrix P (instead of a vector) and is given by

$$f(\mathbf{x}; P) = \prod_{i=0}^{|V_r|-1} \sum_{j=0}^{|V_r|-1} p_{ij} 1_{\{\mathbf{x} \in \chi_{ij}\}} \tag{8}$$

where $\chi_{ij} = \{\mathbf{x} \in \widetilde{\chi} : x_i = j\}$. To specify the updating rules we need to solve the following maximization problem using Lagrange multipliers $\lambda_0, \ldots, \lambda_{|V_r|-1}$,

$$\max_P \min_\lambda \quad \Bigg[\mathbb{E}_P I_{\{\widetilde{S}(\mathbf{X}) \leq \gamma\}} \ln f(\mathbf{X}, P) + \sum_{i=0}^{|V_r|-1} \lambda_i \left(\sum_{j=0}^{|V_r|-1} p_{ij} - 1 \right) \Bigg]$$

By obeying the first order conditions to find the optimum, we differentiate with respect to p_{ij},

$$\frac{\partial}{\partial p_{ij}} \quad \Bigg[\mathbb{E}_P I_{\{\widetilde{S}(\mathbf{X}) \leq \gamma\}} \ln f(\mathbf{X}, P) + \sum_{i=0}^{|V_r|-1} \lambda_i \left(\sum_{j=0}^{|V_r|-1} p_{ij} - 1 \right) \Bigg] = 0.$$

It follows that,

$$\mathbb{E}_P I_{\{\widetilde{S}(\mathbf{X}) \leq \gamma\}} \frac{\partial}{\partial p_{ij}} \ln f(\mathbf{X}, P) \quad + \sum_{i=0}^{|V_r|-1} \lambda_i \frac{\partial}{\partial p_{ij}} \left(\sum_{j=0}^{|V_r|-1} p_{ij} - 1 \right) \quad = 0. \tag{9}$$

Now from eq.(8),

$$\ln f(\mathbf{X}; P) \quad = \quad \prod_{i=0}^{|V_r|-1} \sum_{j=0}^{|V_r|-1} 1_{\{\mathbf{X} \in \chi_{ij}\}} \ln p_{ij}$$

hence differentiating we have,

$$\frac{\partial}{\partial p_{ij}} \ln f(\mathbf{X}; P) \quad = \quad \prod_{i=0}^{|V_r|-1} \sum_{j=0}^{|V_r|-1} 1_{\{\mathbf{X} \in \chi_{ij}\}} \frac{\partial}{\partial p_{ij}} \ln p_{ij}$$
$$= \quad \frac{I_{\{\mathbf{X} \in \tilde{\chi}\}}}{p_{ij}}$$

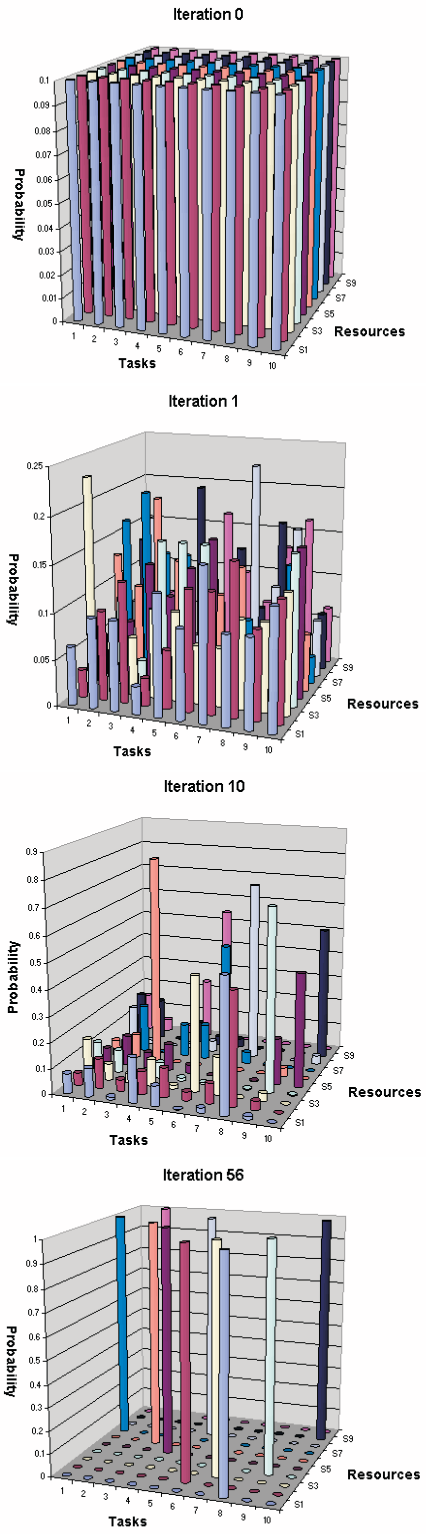

Figure 3. Evolution of the stochastic matrix in a sample run with $|V_r| = |V_t| = 10$.

Therefore, from eq.(9) yields $\forall j = 0, \ldots, |V_r| - 1$,

$$\frac{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} I_{\{\mathbf{X} \in \tilde{\chi}\}}}{p_{ij}} + \lambda_i \quad = \quad 0$$

Summing over $j = 0, \ldots, |V_r| - 1$,we have

$$\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} \quad = \quad -\lambda_i$$

Therefore, this gives us

$$p_{ij} \quad = \quad \frac{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}} I_{\{\mathbf{X} \in \tilde{\chi}\}}}{\mathbb{E}_P I_{\{\tilde{S}(\mathbf{X}) \leq \gamma\}}}.$$

The above can be estimated in a stochastic fashion (by a limited number of samples),

$$\widehat{p}_{ij} \quad = \quad \frac{\sum I_{\{\tilde{S}(\mathbf{X}^{(k)}) \leq \gamma\}} I_{\{\mathbf{X}^{(k)} \in \tilde{\chi}\}}}{\sum_{k=1}^{N} I_{\{\tilde{S}(\mathbf{X}^{(k)}) \leq \gamma\}}}. \qquad (10)$$

The interpretation of the above relation is rudimentary. We count all the $\mathbf{X}^{(k)}$'s that have a value according to our cost metric $S(\tilde{\mathbf{X}}^{(k)})$ less than or equal to some threshold γ and from those count how many of them have their ith coordinate equal to j. From the mapping problem standpoint, we are counting from among all the set of mappings, those mappings that have a cost less than or equal to a threshold value and then among these (hopefully good) mappings, count how many of them have mapped task i to resource j. A ratio of the latter to the former, gives us the probability of successful generation and hence the probability of finding a good overall mapping. Hence, p_{ij} is updated with this probability. And, this is how the proposed CE based **MaTCH** algorithm evolves with iterations. Figure 3 demonstrates visually how the stochastic matrix evolves with iterations. The figure indicates rows as the tasks to be mapped against the columns which are the resources. Initially, since the probability of any task being mapped to any resource is chosen equal, the matrix is evenly distributed. As **MaTCH** evolves, each task develops a bias towards a particular resource given the past history of generated samples. Finally, as the algorithm converges, the stochastic matrix assumes a degenerate form where each task maps to a unique resource with a probability of 1.

The above is the fundamental principle. We first generate X_i from the first row of P and then X_2 from the second row of P, and so on. And then for a sample size of N, we use eq.(10). Of course, $\tilde{\chi}$ contains a lot of undesirable mappings, since we are interested in assigning an unique resource for each task. Hence, when we generate an initial set of vectors or mappings, we decide to choose only those that will serve our purpose. This state space is denoted by the usual χ. We define algorithm GenPerm (Figure 4) to avoid generation of the irrelevant vectors. It is important to note

that the rules for updating $P = (p_{ij})$ remain the same. That is,

$$\widehat{p}_{ij} \quad = \quad \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}} I_{\{\mathbf{X}^{(k)} \in \chi\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}}}. \qquad (11)$$

ALGORITHM GenPerm
1. Generate a random permutation $(\pi_0, \pi_1, \ldots, \pi_{|V_r|-1})$ of $(0, \ldots, |V_r| - 1)$
2. i = 0;
3. **for** each π_i next in order **do**
   Allocate $X_{\pi_{i,j}}$ according to $(p_{\pi_{i,0}}, p_{\pi_{i,1}}, \ldots, p_{\pi_{i,|V_r|-1}})$;
   Set the $X_{\pi_{i,j}}$th column of P to 0;
   Normalize $X_{\pi_{i+1}}$ to sum up to 1;
   $i = i + 1$;
   endfor
4. Return $\mathbf{X}^{(k)}$

Figure 4. Algorithm to generate permutations.

To consummate the **MaTCH** algorithm we need to specify the initialization and stopping criterion for it. For the initial matrix P_0, we simply take the values of all p_{ij}'s to be equal to $1/|V_r|$. The stopping criterion for any CE based method is generally predicting the convergence of the sequence of matrices $P_0, P_1, \ldots$, which ultimately converges to a degenerate matrix, i.e., all the rows have exactly one value equal to 1 where the rest are 0s. In particular, we look for a small constant $c = 5$ (say) such that,

$$\mu_k^i = \mu_{k-1}^i = \ldots = \mu_{k-c}^i \qquad (12)$$

where μ_k^i denotes the maximal element of the ith row of the stochastic matrix P_k. Summarizing what we have discussed so far, we present the **MaTCH** algorithm in Figure 5.

In this algorithm, ρ is termed as the focus parameter. The smaller the value, the quicker the convergence of the algorithm. We choose it to be $0.01 \leq \rho \leq 0.1$. The term N is the sample size of vectors chosen and generated during a single iteration. We choose $N = 2|V_r|^2$. This is because there are $|V_r|^2$ number of elements in the matrix and to evaluate each of them we need a sample size of that order. The other important thing to note about the updating of the p_{ij}'s is that we can introduce the updates to be smoothed instead of coarse as explained above. The smoothing can be done as follows

$$P_{k+1} = \zeta Q_{k+1} + (1 - \zeta) P_k \qquad (13)$$

Where Q_{k+1} is the matrix formed by updating its values according to eq.(11). Obviously, what this smoothing factor achieves is a slower convergence factor, thus allowing the algorithm to converge to a better time.

ALGORITHM MaTCH
1. Initialize P_0 such that all the elements have a value $1/|V_r|$
2. k = 0;
3. Draw a random sample of vectors(mappings) $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ according to **ALGORITHM GenPerm** with $P = P_k$;
4. Calculate the cost of mapping $S(\mathbf{x}^{(i)}), i = 0, 1, \ldots, N-1$ and order these from the largest; to the smallest, $s_0 \geq s_1 \geq \ldots \geq s_{N-1}$;
5. Allow $\lfloor \rho N \rfloor$ to be an integral value and define $\gamma_k = s_{\lfloor \rho N \rfloor}$;
6. Using the same sample, calculate $P_{k+1} = p_{ij}^{(k+1)}$ from
$$\widehat{p}_{ij} = \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}} I_{\{\mathbf{X}^{(k)} \in \chi\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}}};$$
7. $k = k + 1$;
8. **if** $\mu_k^i = \mu_{k-1}^i = \ldots = \mu_{k-c}^i$ **then** STOP;
   else REPEAT from Step 3;

Figure 5. The MaTCH routine.

5. Experimental Study

In this section we describe some simulation experiments we carried out in order to prove the efficacy of the **MaTCH** algorithm. We compare the performance of **MaTCH** against the GA part of our earlier scheme **FastMap** [16], which we shall term as **FastMap-GA**.

FastMap [16] is a hierarchical mapping strategy using a clustering and distribution technique, in which a GA is used to map the tasks. **FastMap-GA** was actually shown to run much faster than a heterogeneous partitioning method [20] yet giving comparable application execution time. The motivation behind comparing with **FastMap-GA** is that, specific to the particular problem at hand, we do not have readily available mapping heuristics. Moreover, GA based heuristics is among the best heuristics available in the literature. The GA encoding explained next is similar to the ones developed in [5, 7, 16].

5.1. The Genetic Algorithm

Genetic algorithms (GAs) provide an efficient optimization and search technique for computationally hard problems [12]. We chose to represent the chromosome as a string of length $|V_r|$ whose values are integers denoting a TIG node and indexed by the resource node, that also are integers. Figure 6 shows an example of such chromosomes. This is known as a *permutation encoding*. The initial population of individuals is generated by assigning a random permutation of TIG nodes mapped to the resource node (index).

The fitness function Ψ of a mapping $\chi_{\mathcal{M}}$ is defined as the reciprocal of the maximum execution time multiplied by a constant $\mathcal{K}$, i.e., $\Psi(\chi_{\mathcal{M}}) = \mathcal{K}/\ Exec^{\chi_{\mathcal{M}}}$. We select the next generation of solutions (children) by the *roulette wheel* se-

lection strategy [12], where the probability of a parent being selected depends directly on its fitness.

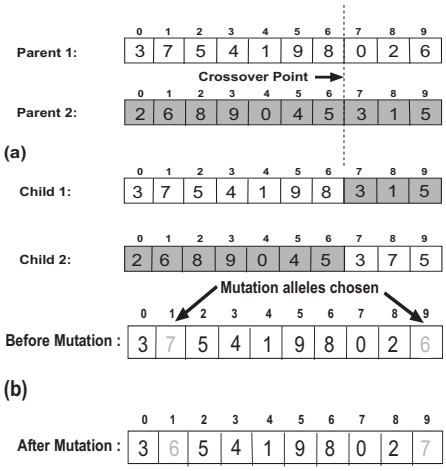

Figure 6. (a) Crossover and (b) mutation operator for our GA.

We employed a simple single point *crossover* operator to enable the GA to converge. The crossover method was implemented in the following manner :

- Copy the first half of the parent onto the child chromosome.
- If any of the genes of the second half of the second parent causes a duplicate mapping, choose (in order) a gene from the first half of the second parent that does not cause a duplicate.

The method is illustrated in Figure 6(a). The crossover probability was set at 0.85. The mutation operator is applied on the intermediate population (after crossover) on each gene based on the mutation probability (set at 0.07) to enable the GA to escape local convergence. The operation is illustrated in Figure 6(b).

We employ *Elitism* which ensures the best individual of a population in the next generation and guarantees faster convergence. Finding a suitable termination criterion for this GA is not trivial. The termination criterion of the GA is actually based on an arbitrary, predefined number of runs of the GA.

5.2. Simulation Setting

The simulation was carried out by comparing the performance of the two algorithms under the same input task and resource graphs. We generated a set of five synthetic graphs with varying computation to communication ratio for each. As mentioned earlier, $|V_t| = |V_r|$. The system graphs generated had node weights randomly varying from 1 to 5. The edge weights that represented the communication overhead was allowed to vary from 10 to 20. Similarly, for the TIG the node weights were taken from 1 to 10 and the edges were randomly generated with weights varying between 50 to 100. Note that we also chose to randomize the generation of the edges so as to represent regions of high density and regions of lower density. The number of nodes in each graph was allowed to vary from 10 to 50 in steps of 10.

For the **MaTCH** algorithm, the number of vectors (samples or mappings) in each iteration was kept to be $2|V_r|^2$ as mentioned earlier. The set that generated the vector did so by choosing a unique resource for each task. Because the probability of mapping a particular task to the set of resources, i.e., one single row of the stochastic matrix, evolved with iterations, and because we wanted a smooth convergence, we chose to employ a selection function similar to the roulette wheel selection employed by **FastMap-GA**. This gave a resource with a higher probability of mapping to be chosen again in the next iteration. However, to guard against premature convergence, we also updated the probability matrix according to eq.(11), by introducing a smoothable factor of $\zeta = 0.3$ (eq.(13)) and hence allowing the algorithm enough time to converge.

The **FastMap-GA** was tuned with a fixed initial population size of 500 and the total number of iterations to a 1000. We also did a few runs with higher number of iterations without noticing any significant improvement in the quality of the mapping. We also tried changing the population size and obtained an ANalysis Of VAriance (ANOVA) [26] analysis for the results obtained. For the analysis the population size was varied from 100 to 1000 (in one step) and the number of generations scaled down from 10000 to 1000 respectively. The crossover probability was fixed at 0.85 and the mutation probability was set a value of 0.07. The low mutation probability was kept in order to allow the GA to converge gracefully.

5.3. Simulation Results

The simulations were performed on a standard desktop machine with Fedora (Linux) running on a Pentium III processor. We quote the results keeping in mind that each result was averaged over 5 different runs of the algorithms on each pair of TIG and resource graphs. It is to be noted that ET represents the same parameter as $Exec^{\chi\mathcal{M}}$ discussed in Section 2 while MT represents the time of execution of the algorithm(s). We also tabulate in Tables 1 and 2, the improvement factor over **FastMap-GA**.

As can be seen, **MaTCH** clearly outperforms **FastMap-GA** in terms of the execution time of the application. How-

$\|V_r\| = \|V_t\|$	10	20	30	40	50
ET_{GA} in units	16585	125579	307158	534124	921359
ET_{MaTCH} in units	3516	8489	13817	17610	23858
ET_{GA}/ET_{MaTCH}	4.717	14.793	23.292	30.33	38.618

Table 1. Comparison of the Execution times between FastMap-GA and MaTCH.

$\|V_r\| = \|V_t\|$	10	20	30	40	50
MT_{GA} in seconds	13.62	22.25	32.58	42.97	50.66
MT_{MaTCH} in seconds	13.47	58.65	268.32	883.96	1587.75
MT_{MaTCH}/MT_{GA}	0.989	2.636	8.23	20.57	31.34

Table 2. Comparison of the Mapping times between FastMap-GA and MaTCH.

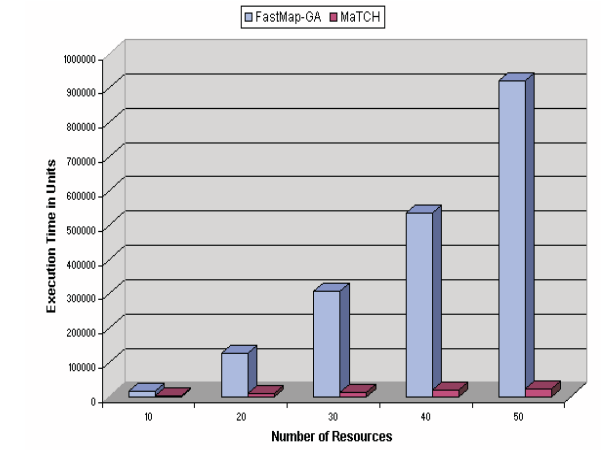

Figure 7. Execution Time in Units for FastMap-GA and MaTCH.

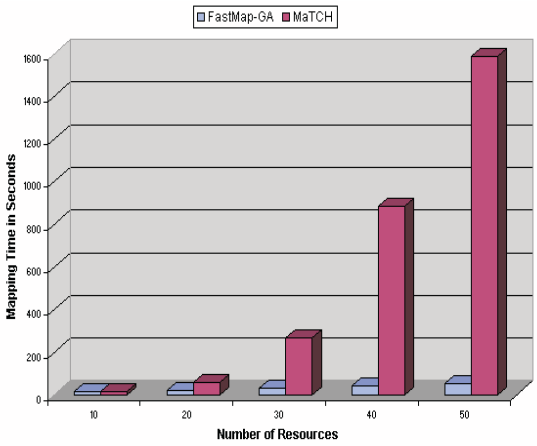

Figure 8. Mapping Time in seconds for FastMap-GA and MaTCH.

ever, it does this sacrificing the time of mapping. This reason stems from the fact that the CE method is a globally iterative procedure and the set of vectors generated at any iteration is directly dependent on the problem size. **FastMap-GA** on the other hand, had a constant population size. However, since the application execution time quoted here is in abstract units, it may in reality take days to actually run. Therefore, the running time of the algorithm is not significant because it is actually quoted to be a few seconds on a desktop machine. We will shortly combine the two parameters to estimate the total time required to map and then execute an application. In other words, we can safely say even though **MaTCH** compromises on the running time, it is in the units of a few seconds and not close to being large enough to affect the overall application turnaround time which includes the execution time that is a much larger quantity in reality.

We further substantiate our results in the following two ways.

- We statistically analyzed through the ANOVA method the performance of **MaTCH** against two different configured versions of **FastMap-GA**. The latter was configured on two parameters, the population size and the number of generations. In one we kept a low population size (100) and let the GA run for a large number of generations (10,000), while in the other we kept a large population size (1000) and a smaller number of generations (1000). We termed the former to be **FastMap-GA 100/10000** and the latter to be **FastMap-GA 1000/1000**. Then each of the above three heuristics were run 30 indepen-

Parameter	MaTCH	FastMap-GA 100/10000	FastMap-GA 1000/1000
Absolute Mean of Mapping Time in seconds	3559	18720	16700
95% Confidence Interval for Mean of Mapping Time in seconds	3143-3975	18300-19132	16288-17120
Standard Deviation	207	1789	836
Median	3535	18770	16730

ANOVA parameters	Value
F value	1547
P value assuming null hypothesis	< 0.0001

Table 3. Statistical and ANOVA analysis of the Execution Time Performance over 10 resources of MaTCH, FastMap-GA 100/10000 and FastMap-GA 1000/1000

dent times on $|V_r| = |V_t| = 10$. The result of the ANOVA test are shown in Table 3.

- We define a quantity called the application turnaround time (ATN) which is expressed as $ATN = ET + MT$. This ATN is the sum of the mapping time and the application execution time and indicates an estimate of the total time required, for parallel applications discussed in Section 2. The graph of the performance of **MaTCH** and **FastMap-GA** against such parameter over an increasing number of resources is shown in Figure 9.

Table 3 shows that the F (comparison between the actual variation of the group averages) value is 1547. Since $F \gg 1$ the result obtained by **MaTCH** is significant. This is attested by the low P value (< 0.0001) associated with the null hypothesis that the results of **MaTCH** as compared with **FastMap-GA** are not significant. The comparison of the ATN's between **MaTCH** and **FastMap-GA** in Figure 9 shows that despite the MT of **MaTCH** increasing sharply with the rise in the number of tasks and resources (Figure 8), the net ATN is significantly smaller than that of **FastMap-GA**.

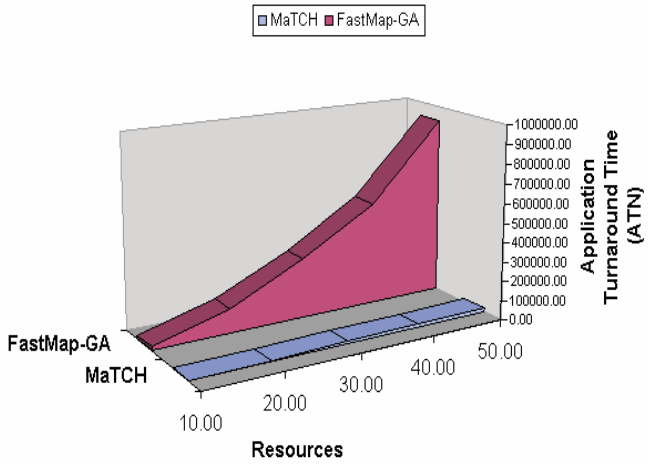

Figure 9. Application Turnaround time for FastMap-GA and MaTCH.

6. Conclusion

We have developed in this paper a novel mapping heuristic based on the cross-entropy (CE) method. We have shown how it outperforms the GA in terms of the quality of mapping by over a factor of about 38. It is worthwhile to note that it was not our intention here to disprove the efficacy of GAs in general but, prove the power of our **MaTCH** algorithm. This of course is done at the cost of sacrificing on the mapping time. This is because the CE method is inherently slow in its execution. However, as we later show through ATN calculations, that the total application turnaround time also improves greatly. To the best of the our knowledge, this is the only work that solves the mapping problem using the CE technique. Our future work includes extending **MaTCH** into a fully distributed implementation using agent based scheduling. Our motivation comes from the fact that such a distributed implementation has already been done in routing of telecommunication networks [13]. We hope to eliminate the defects of the CE method, i.e. the slow running time, by such an implementation.

References

[1] G. Alon, D. P. Kroese, T. Raviv, and R. Y. Rubinstein, "Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment," *Annals of Operations Research*, to appear.

[2] S. Barnard, R. Biswas, S. Saini, R. Van der Wijngaart, M. Yarrow, L. Zechter, I. Foster, O. Larsson, "Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus," in Proc. of Frontiers '99, 1999.

[3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. Robert, "Bandwidth-centric allocation of independent tasks on heterogeneous platforms,"in *Proceedings of IPDPS 2002*.

[4] S.H. Bokhari, "On the Mapping Problem," IEEE Trans. Computers, Vol. C-30, No. 3, Mar. 1981, pp. 207214.

[5] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, 61 (2001) 810–837.

[6] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," In *Proceedings of the 9th Heterogeneous Computing Workshop*, 1999.

[7] A. Dogan and F. Ozunger, "Genetic Algorithm based Scheduling of Meta-Tasks with Stochastic Execution Times in Heterogeneous Computing Systems," *Cluster Computing*, Kluwer 7 (2004) 177-190.

[8] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, to appear.

[9] S. K. Das, D. J. Harvey, and R. Biswas, "A Latency-Tolerant Partitioner for Distributed Computing on the Information Power Grid," *Intl. Parallel and Distributed Processing Symp.*, San Francisco, CA, April 2001.

[10] M. J. Djomehri, R. Biswas, R. F. Van der Wijngaart, and M. Yarrow, "Parallel and Distributed Computational Fluid Dynamics: Experimental Results and Challenges," *HiPC 2000 High Performance Computing: Proc. 7th Intl. Conf.*, Bangalore, India, Dec 2000, Springer-Verlag LNCS Vol. 1970, pp. 183–193.

[11] Marcio Faerman, Adam Birnbaum , Henri Casanova and Fran Berman, "Resource Allocation for Steerable Parallel Parameter Searches," In *Proceedings of Grid Computing - GRID 2002 : Third International Workshop*, Lecture Notes in Computer Science 2536 (2002), pp. 157-168.

[12] D.Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," Addison-Wesley, New York, 1989.

[13] B. E. Helvik and O. Wittner, "Using the cross-entropy method to guide/govern mobile agent's path finding in networks," In *3rd International Workshop on Mobile Agents for Telecommunication Applications - MATA'01*, 2001.

[14] B. Hong and V. K. Prasanna, "A Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput," *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

[15] B. Hong and V. K. Prasanna, "Bandwidth-Aware Resource Allocation for Computing Independent Tasks in Heterogeneous Computing Systems," The *15th Annual International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, November 2003.

[16] A. Jain, S. Sanyal, S. Das and R. Biswas, "FastMap: A Distributed Scheme for Mapping Large Scale Applications onto Computational Grids", In proccedings of *The Challenges of Large Applications in Distributed Environments* Workshop, Honolulu, 2004.

[17] W. E. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid," *8th Intl. Symp. on High Performance Distributed Computing*, Redondo Beach, CA, Aug 1999.

[18] M. Maheswaran, S. Ali, H.J. Siegel, D. Hengsen and R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing* (59), pp. 107-131, 1999.

[19] V. Di Martino and M. Mililotti, "Scheduling in a grid computing environment using Genetic Algorithms," *3rd Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications* (2002).

[20] S. Kumar, S. Das, and R. Biswas, "Graph partitioning for parallel applications in heterogeneous grid environments," *16th Intl. Parallel and Distributed Processing Symp.* (2002).

[21] R. Y. Rubinstein, "Optimization of computer simulation models with rare events," In European Journal of Operations Research (99):89-112, 1997.

[22] R. Y. Rubinstein, "Combinatorial optimization via cross-entropy," In S. Gass and C. Harris, editors, *Encyclopedia of Operations Research and Management Sciences*, pages 102-106. Kluwer, 2001.

[23] R. Y. Rubinstein, "The cross-entropy method and rare-events for maximal cut and bipartition problems," *ACM Transactions on Modelling and Computer Simulation*, 12(1):27-53, 2002.

[24] R. Y. Rubinstein, "Combinatorial optimization, cross-entropy, ants and rare events," In S. Uryasev and P. M. Pardalos, editors, *Stochastic Optimization: Algorithms and Applications*, pages 304-358. Kluwer, 2001.

[25] K. Taura and A. Chien, "A Heuristic Algorithm for Mapping Communicating Tasks on Heterogeneous Resources," in *Proceedings of Heterogeneous Computing Workshop*, 2000.

[26] J. R. Turner and J. Thayer, "Introduction to analysis of variance," *Sage Publications*, Thousand Oaks, CA, 2001.