Mapping and Scheduling Heterogeneous Task Graphs using Genetic Algorithms

Harmel Singh * **Abdou Youssef** †

Department of Electrical Engineering & Computer Science
The George Washington University, Washington DC 20052.

KEYWORDS

Mapping, Scheduling, Graphs, Genetic Algorithms, Heterogeneous Processing.

ABSTRACT

Heterogeneous processing deals with decomposing a general task into code segments of diverse computation requirements and executing each segment on a machine such that the task takes a minimum amount of time. An important challenge in this growing field is the mapping and scheduling of code segments, a problem known to be NP-hard. Genetic Algorithms (GA) have been shown to be very robust heuristic optimization tools for very hard combinatorial problems. These are a class of algorithms closely modeling the evolution process in nature. In this work we develop and optimize a genetic algorithm that *evolves* optimal solutions to the problem of mapping/scheduling *general* task graphs on a cluster of heterogeneous machines. This is done by first formulating the mapping/scheduling problem in GA terms, and then evaluating various genetic algorithm parameters for obtaining best performance. This parameter set is used for mapping/scheduling *general* task graphs. We study the performance of our algorithm and compare it with optimal polynomial time algorithms for tasks of the form of *trees* and *series/parallel* graphs. The performance results show that optimal or near optimal solutions are generated very fast by our GA.

*hks@psi.com
†youssef@seas.gwu.edu

1 Introduction

Heterogeneous Processing (HP) has recently received much attention owing to its promise of delivering usable computing performance from existing computing resources. The central idea in HP is identifying the characteristics of the application to be run and the machine(s) available for it. After the characteristics of the tasks and machines have been determined this information can then be utilized for optimally *matching & scheduling* the task on the machine cluster. The characteristics of applications and machines are determined using the two analytic steps [14] viz. *code-type profiling* and *analytic benchmarking*. Using the data generated by code profiling and analytic benchmarking each code segment is mapped or assigned onto the most suitable machine. Efficient mapping and scheduling of tasks is critical for achieving the desired performance on any parallel processing system.

Mapping deals with assigning the subtasks to suitable machines using suitable subtask map-

pings on a machine in order to achieve the desired performance goals [6]. *Scheduling* is involved with determining what would be the best start time for each subtask on its host machine after the mapping is completed.

It is known that the problem of mapping and scheduling is non-trivial. Finding an optimal mapping/schedule of a general task graph is NP-Hard [21]. Researchers have tackled this problem in various ways including developing polynomial time algorithms for restricted cases of the problem [7]. This is done by limiting the structure and characteristics of tasks and/or machines. For general cases, heuristics have been used [20] for finding near-optimal solutions.

Genetic Algorithms belong to the same class of guided random search algorithms as simulated annealing [13, 11] and are modeled after the phenomena of evolution in nature. They have been shown to be very robust and to produce encouraging results for very hard multimodal search spaces. An important characteristic of GA's is that they can be explicitly parallelized. Furthermore, they are theoretically proven to be capable of implicitly and rapidly sampling large search spaces parallel [15, 19].

Among several parameter affecting the performance of a GA, the most significant are the techniques used for *encoding* the solutions and the function used for evaluating the *fitness* of the encoded solutions [4, 5]. It has also been realized that [15] there may not be a single GA for all classes of problems, rather one has to tailor the algorithm to a specific class of problems by incorporating problem-specific knowledge into the algorithm [15].

This paper presents our work in developing and analyzing a genetic algorithm for optimally mapping and scheduling arbitary task graphs in heterogeneous processing environments. It is organized as follows. The next section summarizes the related work in mapping/scheduling heterogeneous task graphs and also a brief look at genetic algorithms. Section III gives the formal problem statement and our approach towards solving it. Section IV presents the out performance evaluation and the performance results of our genetic algorithm. Finally, we conclude in section V and propose future possible directions for extending this work.

2 Related Work

2.1 Selection Theories

Mapping and scheduling in HP is addressed in a series of *selection theories* proposed by researchers. The *Optimal Selection Theory (OST)* [14] gave the initial mathematical basis for selecting a suite of machines for a particular code types. This basic theory was later extended in various ways by others [23, 10, 21].

In OST it was assumed that optimal performance was achieved by executing a code segment on a best matched machine, and that all types of machines were available. The OST was extended by Wang to *Augmented Optimal Selection Theory (AOST)* [23] in the following ways - firstly it was assumed, more realistically, that only a limited number of machine types were available, secondly, mapping the code segments on suboptimal machine choices was allowed, and finally non-uniform decomposition of code-segments was considered.

It is observed in practice that the execution times of code segments depends not only on the type of machine it is assigned to but also on the different types of mappings and parallelism available on these machines. For example, the execution time of various matrix algorithms varies significantly if the matrices are mapped using *striped* or *checkerboard* partitioning on the target mesh or hypercube. The *Heterogeneous Optimal Selection Theory (HOST)* [10] extended AOST by incorporating this view.

In all the above selection theories the task graphs in some form or another was modeled as serially ordered code-segments or subtasks. The *Generalized Optimal Selection Theory (GOST)* [21] extends the model by consid-

ering task graphs that are general. Each subtask is thought of as a complete execution unit, a process. GOST also recognizes the different types of networks that may simultaneusly connect the resources in the HP environment. In GOST polynomial time algorithms for optimally mapping/scheduling series-parallel dependency graphs and tree graphs are presented. Series-Parallel graphs represent a large class of the programs containing *parbegin/parend* and *fork/join* type of constructs. Trees represent the class of *divide-and-conquer* programs. These mapping/scheduling algorithms will be used for comparing the performance of our GA to them.

2.2 Genetic Algorithms

Genetic Algorithms (GA) were first introduced by J. Holland [19] in 1975. These are stochastic algorithms belonging to a class of *Evolutionary Algorithms* which use ideas borrowed from the process of natural evolution for solving very hard optimization problems.

Here the evolution proceeds on the basis of the *survival of the fittest* theory. The *fitness* of an individual is defined by the environment in which the algorithm is implemented. The basic model of GA consists of a *population* of randomly generated initial solutions encoded in the form of bit or character strings, known as the individuals or *chromosomes*[‡]. The evolution consists of evaluating the fitness of the individuals of the present population, using a user-supplied *fitness function*, and generating a new population depending on the fitness of each individual. Fitter individuals have more representation in the new generation. Each individual in a generation goes through a *crossover* operation. In this operation two randomly selected chromosomes are crossed or spliced at a random point to generate, hopefully, a better individual. This generation process is continued in order to evolve better individuals in subsequent generations and converge towards an optimal solution. The detection of convergence, or convergence criteria, will be discussed later. To fully explore the solution space and help GA get out of local optima, the chromosomes are randomly *mutated* - another idea borrowed from nature.

Even though the essential components of GA are outlined in [19, 15], exactly what part each process plays, how important each process is in contributing towards evolution and how do these processes interact, all these are still not fully understood and remain subject of current research [5]. We now briefly present important parameters that affect any genetic algorithm. Note that only the parameters that are actually tested as part of our algorithm are being introduced.

1. ***Population size*** ($\mathcal{P}$) ***and Number of Generations*** ($\mathcal{N}$) The population size $\mathcal{P}$ should be large enough so that there is enough diversity for the algorithm to sample the complete search space, but small enough so that there is no spurious load of computation introduced. The number of generations determines basically the time a GA has to find an acceptable solution, given a set of parameters. The choice of $\mathcal{N}$ is critical and is closely tied to the termination criteria.

2. ***Encoding techniques*** Encoding can be formally defined as a function $\mathcal{E} : \mathcal{O} \rightarrow \mathcal{R}$. Here $\mathcal{O}$ is the object or valid solution space and $\mathcal{R}$ is the representation of objects or solutions in the GA. This encoding is necessary to render feasible the string based operations of GA. It has been dictated by the schema theorem [19] that any encoding scheme used must support the *building block hypothesis* [15], ie, successful encoding consists of schemata that are of short-defining length and that interact minimally.

3. ***Fitness Modification And Selection Methods*** The aim of the selection method is to rapidly promote the presence of good *schema* in the new population (by having

[‡]From this point the word individual and chromosome will be used interchangeably to mean the same thing

more representative individuals of the good schema in the new population) and at the same time preventing hitchhiking of bad schema of the old population on the better schema in the next generation. The problems of *premature convergence* and *slow finishing* [5] might be exhibited by the genetic algorithms that rely solely on the absolute value of the fitness of the individuals for generating the next population set. To handle these problems, several strategies for either modifying the fitness and/or using a modified selection method for generating the next population have been investigated [17, 15, 18]. These strategies and methods are briefly discussed next.

The *scaling methods* modify the raw fitness value of all the individuals in a population that was computed using the fitness function. They can be formally expressed as a function $C : f_{raw} \rightarrow f, \forall \mathcal{I} : \mathcal{I} \in \mathcal{P}$. The important scaling techniques for readjusting fitness explicitly are-

- Window scaling [GAucsd/GeneSYS packages]
- Sigma truncation [15]
- Linear scaling [15]

The *selection methods* are formally the functions of the form $\mathcal{S} : Pop_i \rightarrow Pop_{i+1}$ Here Pop_i is the population of GA in generation i. The basis of selection is the fitness of the individuals of the old population computed using the fitness function. For more detailed description and implmementation of the selection methods the respective references are suggested.

- Roulette Wheel [19]
- Fitness Ranking [24]
- Stochastic Remainder [3]
- Stochastic Universal [3]
- Tournament [9]

Various selection methodologies have been experimented with, and it has been realized that none of the selection methods is clearly the best [17]. One has to experiment and tailor a selection method for each specific problem.

4. ***Recombination Operators*** $(\mathcal{R})$ The recombination or crossover operator is an important player in mixing optimal schema to generate super fit individuals from the existing generation. The amount of crossover or genetic mixing, for any generation, is controlled by a parameter called the probability of crossover (p_c). The different types of recombination methods are listed below.

   - 1-point crossover [15]
   - 2-point crossover [15]
   - Uniform Crossover [22]

   Again none of the crossover operators is clearly the best.

5. ***Mutation Operators*** $(\mathcal{M})$ The mutation operators are the main factors for introducing the lost *schemas* back into a rapidly converging GA, thereby allowing the GA to escape from local optima [15, 8]. The mutation operator acts on a chromosome by arbitrarily selecting a gene on the chromosome and modifying it randomly. The degree of mutation for a population is controlled by a parameter called the probability of mutation p_m. The probability of mutation p_m should be low enough so that good schemas are not disrupted, but large enough so that a reasonable amount of diversity in the GA is maintained.

6. ***Miscellaneous parameters*** Additional refinements to the GA's have been made. Among the more significant of these refinements and their parameters are -

   - Niching [16]
   - Incest Reduction [12]

- Elitism [2]

In the above discussion we have presented a set of important parameters that we would be evaluating to arrive at a *working set* of parameters for our primary problem. It should be noted that finding an optimal parameter set for any application is very hard. The complexity is introduced not only by the presence of several distinct parameters, but by the presence of sub parameters for each parameter. Another reason for the complexity is the mutual dependence of various parameters [8, 1].

Therefore, for simplicity, we will be limiting our evaluation only to the parameters tabulated in Table 1. We will be using standard acceptable values suggested in the literature for all the parameters not evaluated in this paper.

Parameter	Value/Type
Population Size	positive Integer (50-500)
Encoding Technique	presented in next section
Max. Generations	positive Integer (50-200)
Scaling Methods	window, sigma trunc. linear
Selection Methods	rank, roulette, stoch. remainder, stoch. universal, tournament
Crossover Operators	1 point, 2 point & uniform
Crossover Prob.	0.15-0.35
Mutation Operators	simple field-mutation
Mutation Prob.	0.001-0.02
Niching	enabled/disabled
Elitism	enabled/disabled

Table 1: The List of GA Parameters to be tested

3 GA For Mapping & Scheduling

In this work we will be determining a static compile-time schedule for an arbitary precedence constrained directed acyclic task graph. The task graph will be consisting of subtasks as the nodes. The nodes are the basic units of execution on each machine available. We assume there is a finite number of machine and network types available, but there are unlimited number of machines and links for each type. The communication model adopted is the blocked send/receive type. That is, a subtask can either be communicating or executing but not both at the same time. Without any loss of generality we can assume that each subtask first executes the code assigned to it and then communicates with its successors and transfers the necessary data to them. The communication time is dependent not only on source and destination machines, but also on the mapping used by them. Of course the type of link used by them also is factor in the communication time. The communication time includes the time for setup, data reformatting and the actual transfer time. Any subtask starts executing when the last of the set of its predecessors has transfered the data to it.

Problem Statement *Given* a general directed acyclic task graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the nodes in $\mathcal{V}$ represent the subtasks of a general task, and $\mathcal{E}$ are the edges representing the precedence relation between the nodes. Assume the number of nodes in the graph is equal to η. There are τ machine types interconnected by λ types of links. There are ϕ mappings available per subtask on each machine type τ. ϕ is equal to one on machine types other than SIMD, MIMD etc., i.e, on serial machines. Also given -

$\mu_i^{k_m}$ The execution time of subtask i on machine k using the mapping m, where $1 \leq k \leq \tau$ and $1 \leq m \leq \phi$.

$\delta(i_a^m, j_b^n, l)$ The communication time from task i assigned to a machine of type a using mapping m to task j assigned on a machine of type b using the mapping n over a link of type l, For all $i, j \in \mathcal{V}$, $1 \leq a, b \leq \tau$, $1 \leq m, n \leq \phi$ and $1 \leq l \leq \lambda$.

An assignment is represented as π, where $\pi(i) = (a, m)$ gives the machine assignment of

node i to machine a with mapping m. A schedule σ_π corresponding to an assignment π is a function of the form : $\sigma_\pi(i) = (x, y)$; Here x is the start time of subtask i on machine a with mapping m, where $\pi(i) = (a, m)$, and y the stop time of subtask i. We assume $y - x = \mu_i^{k_m}$ for all $i \in \mathcal{V}$.

Let $\mathcal{F}_i$ be the *finish time* of node i, and $\mathcal{F}_{last} = \max_{\forall i \in \mathcal{V}} \mathcal{F}_i$ the finish time of the last node(s) in $\mathcal{G}$. The problem is:

> ***Find*** the optimal assignment π and schedule σ_π for $\mathcal{G}$ which minimizes $\mathcal{F}_{last}$.

The template of our genetic algorithm and description of the parameters is given in Figure 1.

mga($\mathcal{P}, \mathcal{N}, \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{S}, \mathcal{R}, \mathcal{M}, \mathcal{T}$)
/* $\mathcal{P}$-population size, $\mathcal{N}$-max. number of generations,
$\mathcal{E}$-encoding function, $\mathcal{F}$-fitness function,
$\mathcal{C}$-scaling function, $\mathcal{S}$-selection method,
$\mathcal{R}$-recombination method, $\mathcal{M}$-mutation method,
$\mathcal{T}$-termination criteria */
begin
Read Graph ($\mathcal{G}$) ; /* with η nodes */
initialize ;
$\gamma = 0$; /* current generation */
/* Individuals for generation γ , $\mid \Psi \mid = \mathcal{P}$,
encoded using $\mathcal{E}$*/
population : Ψ_γ ;
compute fitness : $\mathcal{F}(\mathcal{I})$; $\forall \mathcal{I} \in \Psi$;
Scale fitness: Apply $\mathcal{C}$ on $\mathcal{F}(\mathcal{I})$, $\forall \mathcal{I} \in \Psi$
while ($\gamma < \mathcal{N}$ AND $\mathcal{T}(\Psi_\gamma) = FALSE$) **loop**
the next generation : $\Psi_\gamma \xrightarrow{\mathcal{S}} \Psi'_{\gamma+1}$;
/* Here Ψ' is a temporary population
for generating the new population */
Recombine : $\Psi'_{\gamma+1} \xrightarrow{\mathcal{R}} \Psi_{\gamma+1}$;
Mutate : $\Psi_{\gamma+1} \xrightarrow{\mathcal{M}} \Psi_{\gamma+1}$;
Compute fitness : $\mathcal{F}(\mathcal{I})$; $\forall \mathcal{I} \in \Psi$;
Scale fitness: Apply $\mathcal{C}$ on $\mathcal{F}(\mathcal{I})$, $\forall \mathcal{I} \in \Psi$
$\gamma := \gamma + 1$;
end loop
end mga

Figure 1: GA template

It should be noted that an actual algorithm will be an *instance* of that template with specific parameters. The algorithm starts by reading a random task graph. Depending on the parameter $\mathcal{P}$ for the population size, a set of random individuals corresponding to parameters of the graph generated are initialized. Each individual in the population is evaluated for its fitness using the fitness function provided. After these initial steps, the cycle of successive scaling, selection, crossover and mutation is continued until the termination criteria is met or the maximum number of generations is reached.

No operator including all the selection and recombination and mutation operators, in the GA takes more than $\mathcal{O}(\mathcal{P}.\eta)$. The Genetic Algorithm cycle hence consists of a single run of all the operators with $\mathcal{P}$ runs of the fitness function. It can be easily seen that the fitness function accounts for more than 90% of the time taken by the GA. The total time taken by the GA will approximately be $\mathcal{N} * \mathcal{P}$ times the time taken by the fitness function. If the time taken by the fitness function is significant, then the fitness of the population can be computed in parallel slashing down the time by a factor of $\mathcal{P}$.

Discussion

Now we discuss briefly some important parameters used for our Genetic Algorithm.

Encoding ($\mathcal{E}$): Each chromosome $\mathcal{I}$ has n genes. n is equal to number of nodes in $\mathcal{G}$. Each gene of $\mathcal{I}$ is a number in the number system of base $\tau * \phi$. The elements of the chromosome are decoded as follows-

$$\pi(i) = ((\lfloor \mathcal{I}_i/\phi \rfloor + 1), (\mathcal{I}_i \ mod \ \phi + 1)),$$
$$\forall i : i \in \mathcal{V}.\ \mathcal{I}_i \text{ is } i^{th} \text{ digit of } \mathcal{I}.$$

The schedules are not coded because, as will be seen in the next subsection, the optimal schedule corresponding to an assignment π is derived directly from π.

Termination criteria ($\mathcal{T}$): As mentioned earlier the GA is a stochastic algorithm; there is *no* indication as to how far or close the algorithm

is to the solution. We terminate the algorithm if there is no improvement of the fitness over a span of t generations. t can be the tolerance of the GA. We choose t to be 30 from experimental results.

Fitness function ($\mathcal{F}$): The pseudo code for the fitness function used is given in Figure 2. The fitness function starts off by first decoding

Fitness($\mathcal{I}$)
begin
decode : $\forall i \in \mathcal{V}$, $\pi(i) = (\lfloor \mathcal{I}_i/\phi \rfloor + 1,\ \mathcal{I}_i \ mod\ \phi + 1)$
compute schedule σ_π :
/* reorder the nodes so that $i < j$,
if i is predecessor of j */
Perform topological sort on $\mathcal{G}$
for $i = 1$ to n **do**
  if i has NO predecessor **then**
    $\sigma(i) = (0, \mu_i^{a_b})$, where $\pi(i) = (a, b)$
  else
    for every predecessor p of i **do**
      /* $\sigma(p) = (p_{start}, p_{finish})$
      This was computed in the previous
      iterations, because of topological
      ordering */
      $F_p = MIN_{\forall links}[p_{finish} + \delta(p_c^d, i_a^b, l)]$
        /* $\pi(p) = (c, d)$ */
      $i_{start} = MAX_{\forall p}\ F_p$
        /* i_{start} is the start time of node i */
      $i_{finish} = i_{start} + \mu_i^{a_b}$
      $\sigma(i) = (\ i_{start},\ i_{finish}\ )$,
        where $\pi(i) = (a, b)$
    end for
  end if
end for
return finish time : $\mathcal{F}_{last} = MAX_{\forall i \in \mathcal{V}}(i_{finish})$
end

Figure 2: The Fitness Function

$\mathcal{I}$ into π. The next step is to topologically sort the graph. This is a simple way to ensure that for any given node the finish times of all its predecessors are already computed. If the node is a root(i.e, has no predecessors) in graph $\mathcal{G}$, then finish time for it is just the execution time, since the start time for all the roots is zero. If the current node is not a root we compute the earliest time for each predecessor to complete execution and transfer of data to it. The earliest time any given node can start then is the time when the last of its predecessors have transferred the data to it. This way we compute the schedule for all the nodes in the task graph, according to the mapping π given by our chromosome. The fitness of the chromosome or mapping is the finish time of the graphs which will be the finish time of the last node of the graph. The last node will be a leaf of the graph, therefore the fitness is the maximum finish time of all the leaves belonging to graph $\mathcal{G}$.

The time complexity of topologically sorting the graph is $\mathcal{O}(\eta + e)$. where e is the number of edges in the graph. The outer for loop clearly takes time $\mathcal{O}(\eta + e.\lambda)$, where λ is the number of link types in the system. Therefore the fitness function takes $\mathcal{O}(\eta + e.\lambda)$

4 Simulation Plan/Results

Our simulation plan and results for tuning the various parameters and for operator selection is presented next. The subsequent subsection summarizes the simulation results for mapping and scheduling.

4.1 GA Parameter Tuning

Finding the optimal values for the parameter set for a GA is non-trivial. Hence, in this simulation we adopt a greedy approach for finding the best parameter set. We start our search from universally acceptable parameter values and search for good values in their neighborhood. We will use the following 2 mutually dependent metrics to measure of *goodness* of the set of parameter values -

Metric 1: The number of generations that the GA needed to converge according to termination criterion ($\mathcal{T}$). If the GA doesn't converge

in the given number of generations, it is terminated and the value of metric is taken as ($\mathcal{N}$).
Metric 2: The fitness error of the final fitness is computed using the relation:

$$f_{error} = [(f_{\mathcal{T}} - f_{optimal})/f_{optimal}] * 100.$$

where $f_{\mathcal{T}}$ is the fitness of the best individual in the last generation of the GA. Note that in our simulations we will use graphs for which $f_{optimal}$, which is the fitness of the optimal solution, is known.

All the graphs have the following characteristics: Number of Nodes (η) = 40, Number of Machine types (τ) = 2, Number of Mappings available per machine (ϕ) = 3 and Number of Link types λ = 2.

Results

The results are shown in Tables 2 to 8. All the parameters of GA were kept the same and different selection methods were used. As can be seen *tournament* selection method is clearly the best selection method both in speed of convergence of the algorithm and accuracy of the solution (i.e, closeness to optimality). Next, we experimented with tournament sizes varying from 2 to 8 in steps of 2. As can be seen from Table 2, as we increase the tournament size from 2 the GA tends to converge more rapidly at the expense of larger error in the final fitness. We initially compared the 3 crossover operators under the tournament selection method. The results are summarized in Table 3. Uniform crossover was found to produce the best results. Different crossover probabilities were also tried for the uniform crossover and the results are tabulated. As for mutation different mutation probabilities were compared. Table 4 shows the results. With best parameters chosen from previous generations we experimented with turning niching on and off for the GAs. We kept the crowding factor for Niching to 2 and turned on Incest reduction. As can be seen, niching significantly improved the performance of the algorithm. The results of using different scaling methods is summarized in Table 6. As can be seen from Table 7, leaving elitism on has positive effects on the algorithm. We experimented with populations of sizes ranging from 50 to 200 in steps of 50. The results match with the theory that as we increase the population size there is more rigorous search of the solution space and hence a GA results in lesser final fitness errors. Table 8 gives the summary of the average performance metrics for different population sizes.

SELECTION TYPES		
Sel. Method	Mean Error(%)	Mean # of Gen.
rank	49.9	55.6
roulette	24.06	80.5
st. remainder	12.62	105.5
st. universal	12.29	111.8
tournament	4.89	82.0
TOURNAMENT SIZES		
Tour. size	Mean Error(%)	Mean # of Gen.
2	4.89	82.0
4	5.28	65.0
6	6.51	59.9
8	6.31	60.5

Table 2: Average metric values : selection methods

CROSSOVER TYPES		
Crossover type	Mean Error(%)	Mean # of Gen.
1-point	4.89	82.0
2-point	4.16	72.5
uniform	3.14	69.2
CROSSOVER PROBABILITIES		
p_c	Mean Error(%)	Mean # of Gen.
0.15	3.85	72.6
0.25	3.14	69.2
0.35	2.8	56.4

Table 3: Average metric values : crossover methods

MUTATION PROBABILITIES		
p_m	Mean Error(%)	Mean # of Gen.
0.001	6.7	56.8
0.01	3.14	69.2
0.02	1.85	81.4

Table 4: Average metric values : mutation probabilities

Niching	Mean Error(%)	Mean # of Gen.
Enabled	3.08	67.0
Disabled	3.14	69.2

Table 5: Average metric values : niching

WINDOW SCALING		
Window Size	Mean Error(%)	Mean # of Gen.
0	3.08	67.0
1	3.08	67.0
2	3.08	67.0
SIGMA TRUNCATION		
Sigma	Mean Error(%)	Mean # of Gen.
1	3.08	67.0
2	59.1	2.8
3	62.2	3.92
LINEAR SCALING		
Scaling Multiple	Mean Error(%)	Mean # of Gen.
1.2	3.08	67.0
1.4	3.08	67.0
1.6	3.08	67.0

Table 6: Average metric values : scaling methods

4.2 Mapping & Scheduling

In this subsection we use the best parameters found in the previous section to map and schedule both regular and general graphs types. The working GA parameter set is summarized in Table 9.

The simulations were conducted using graphs

Elitism	Mean Error(%)	Mean # of Gen.
enabled	2.8	59.1
disabled	3.4	75.0

Table 7: Average metric values : elitism

Population ($\mathcal{P}$)	Mean Error(%)	Mean # of Gen.
50	7.41	68.3
100	2.80	59.1
150	1.52	66.7
200	1.30	64.1

Table 8: Average metric values : population size

of general, trees and series-parallel type. For each graph type (general, tree and series/parallel), graphs were generated with different number of nodes, mapping types, machine types and link types. The graphs were grouped under 3 groups depending on the value of the parameters which are summarized in Table 10. To sum up simulations were carried out for 10 graphs for each group, for each type of graph, making it a total of 90 graphs.

Parameter	Value
Population size ($\mathcal{P}$)	500
Max. # of Gen. ($\mathcal{N}$)	200
Scaling method ($\mathcal{S}$)	sigma truncation $\sigma = 1$
Selection method ($\mathcal{S}$)	tournament $tour.size = 2$
	Niching = yes (cf=2 , ir=yes)
	Elitism = yes
Crossover method ($\mathcal{R}$)	Uniform with $P_c = 0.25$
Mutation method ($\mathcal{M}$)	Field $P_m = 0.01$
Termination criteria	No improvement over 30 generations
Fitness Function	from figure 2

Table 9: The *working* parameter set for our GA

Results

The results of running the Genetic Algorithm

parameter	values		
	Group A	Group B	Group C
Nodes	10	50	100
Mappings (ϕ)	3	2	2
Machine types (τ)	3	3	2
Link types (λ)	2	2	2

Table 10: Characteristics of test graph groups

on the 3 groups of graphs with different graphs (series-parallel, tree & general) is shown in the figures 3 to 5.

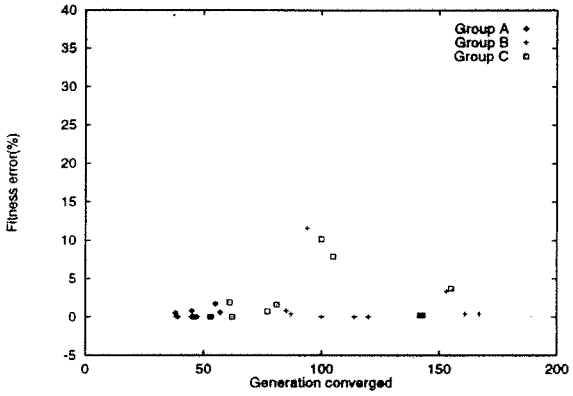

Figure 3: GA results for Tree type of graphs

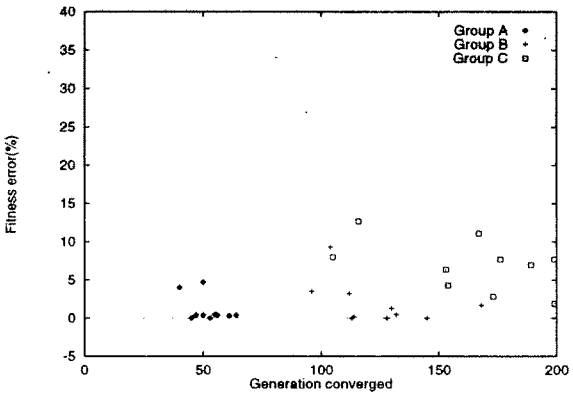

Figure 4: GA results for series-parallel graphs

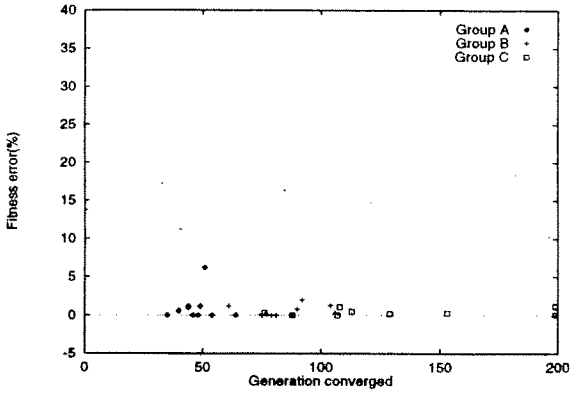

Figure 5: GA results for General Graphs

5 Conclusion

Our initial step towards determining a viable set of parameters produced results that agree with the theory. This working set of parameters was then used for an algorithm for mapping and scheduling heterogeneous task graphs on a heterogeneous machine cluster. It was found that the algorithm produced near optimal results for not only tree or series-parallel task graphs, but also general graphs, within acceptable error rates. In fact the GA produced optimal solutions in many instances and the maximum error experienced in overwhelming number of cases was below 10%. It should be noted that the algorithm invariably terminated within predicted bounds (i.e, ≤ 200 generations) for all the test graphs. That is, our genetic algorithm is very fast, indeed considerably faster than the deterministic algorithms in [21]. In summary, our experiences strongly indicate the usefulness and robustness of genetic algorithms in tackling this complex optimization problem.

For future work, one might attempt to tighten our problem formulation by developing the mapping & scheduling with finite numbers of machines and communication links for each type. The algorithm could be tested with actual task graphs as opposed to randomly generated ones,

and with wider parameter ranges. This might better reveal the strengths and weaknesses of this approach. Finding a good parameter set is also important for a realistic range of data. The work can proceed further by incorporating more problem-specific information into our GA or trying to hybridize the Genetic Algorithms with other known optimization techniques.

Implementation

The simple genetic algorithm template and various genetic operators are used from the *GALOPPS v2.36* toolkit. GALOPPS (Genetic Algorithm Optimized for Portability and Parallelism) provides a rich set of genetic operators and statistic gathering and processing routines. The toolkit also provides a highly extensible interface to easily adapt the simple genetic algorithm template for any application specific processing by providing various hooks for application specific routines into itself. The task graphs and related routines were defined on top of the *Directed Acyclic Graphs* package implemented in Ada.

References

[1] T. Bäck. The interaction of mutation rate, selection, and self- adaptation within genetic algorithms. Technical report, University of Dortmund, D-44221 Dortmund, Germany, 1993.

[2] T. Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, 1991.

[3] J. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, 1987.

[4] D. Beasly, D. Bull, and R. Martin. An overview of genetic algorithms: part1, fundamentals. *University Computing*, 1993.

[5] D. Beasly, D. Bull, and R. Martin. An overview of genetic algorithms: part2, research topics. *University Computing*, 1993.

[6] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, Mar 1981.

[7] Shahid H. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, Jan 1988.

[8] M. Bramlette. Initialization, mutation and selection methods in genetic algorithms for function optimization. In *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, 1991.

[9] A. Brindle. Ga for function optimization. Technical report, University of Alberta, Edmonton, 1981.

[10] Song Chen, M. Eshagian, A. Khokhar, and E. Shabaan. A selection theory and methodology for heterogeneous supercomputing. *Workshop on Heterogeneous Processing*, 1993.

[11] L. Davis. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufman Publishers, Inc., 1987.

[12] L. Eshelman and J. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, 1991.

[13] J. Filho, P. Treleaven, and C. Alippi. Genetic-algorithm programming environments. *IEEE Computer*, Jun 1994.

[14] R. F. Freund. Optimal selection theory for superconcurrency. *Supercomputing '89*, Nov 1989.

[15] D. Goldberg. *Genetic algorithms in search, optimisation, and machine learning.* Addison-Wesley, 1989.

[16] D. Goldberg and K. Deb. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989.

[17] D. Goldberg and K. Deb. A comparative study of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

[18] E. Goodman. The ' genetic algorithm optimized for portability and parallelism ' system, 1994. User guide for GALOPPS 2.36.

[19] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan press, 1975.

[20] Virginia Mary Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on Computers*, Nov 1988.

[21] B. Narahari, A. Youssef, and H. Choi. Matching and scheduling in a generalized optimal selection theory. Technical report, The George Washington University, Washington DC 20052, 1993.

[22] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989.

[23] Mu-Cheng Wang et al. Augmenting the optimal selection theory of superconcurrency. *Workshop on Heterogeneous Processing*, 1992.

[24] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reporoductive trials is the best. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989.