GENETIC SIMULATED ANNEALING FOR SCHEDULING DATA-DEPENDENT TASKS IN HETEROGENEOUS ENVIRONMENTS

Pankaj Shroff* — sl0vt@sys3.cs.usu.edu
Daniel W. Watson* — watson@cs.usu.edu
Nicholas S. Flann* — flann@nick.cs.usu.edu
Richard F. Freund† — freund@nosc.mil

*Department of Computer Science
Utah State University
Logan, Utah 84322-4205

†Heterogeneous Computing Team
NCCOSC RDTE Division 4221
San Diego, California 92152-7446

KEYWORDS: heterogeneous computing; genetic algorithms; matching; simulated annealing; scheduling.

ABSTRACT

Heterogeneous Computing (HC) is considered the use of different types of processors, processing components, or connectivity paradigms to maximize performance, cost-effectiveness, and/or development effort. One of the fundamental challenges in HC is to provide a schedule of tasks on a set of hosts that will result in minimum overall execution time[1]. Compounding the problem is the need for data to be moved to different hosts in the HC system as needed by executing tasks. This paper proposes a Genetic Simulated Annealing (GSA) based approach to perform scheduling in HC environments. GSA is a hybrid method which stems from two fundamental approaches in classic, search based problem solving, namely Genetic Algorithms (GA) and Simulated Annealing (SA). GSA-based scheduling uses a population of solutions and reproduction, crossover and mutation operations among population members (as in GA), but a selection mechanism based on a temperature cooling schedule (as in SA). This paper serves to accomplish two objectives: (1) to provide a formal definition of the problem of scheduling interdependent, data-intensive tasks in HC environments, and (2) to analyze the GSA-based scheduling approach applied to this problem.

1. HC SYSTEM MODEL AND THE SCHEDULING PROBLEM

Heterogeneous Computing (HC) [2, 3, 4] systems are typically employed as cost-effective alternatives to traditional high-performance supercomputing systems. Applications implemented in HC environments achieve enhanced performance by exploiting differences among diverse resources available for the execution of computational tasks. The *overall completion time* for a related set of tasks to be executed in an HC system is the time when the last task to be scheduled is completed. HC *scheduling constraints* are the sequential and concurrency dependencies between tasks requiring transfer of data between them. A key obstacle to achieving performance improvement is the determination of schedules for the execution of these tasks such that (1) overall completion time is minimized, and (2) scheduling constraints between the task to be executed are observed.

For this study, it is assumed that all hosts in the system are available and ready for use, and that the load from other tasks is not considered. Furthermore, it is assumed that no more than one task at a time is to be executed on any individual host. Relaxing these restrictions is a goal for future research; however, solving this restricted problem is an important step toward that goal. The scheduling problem can be formally stated as follows.

Given:

- a set of n tasks which are interdependent and may require transfer of data between them, denoted as the set $T=\{T_1, T_2, ..., T_n\}$.
- a set of m hosts which are available for scheduling tasks in the HC system, denoted as the set $H=\{H_1, H_2, ..., H_m\}$.
- an $n \times n$ matrix of *etc* (estimated time to complete) values such that $etc[T_i, H_j]$ is the estimated time to complete task T_i on host H_j (e.g., [5])

a *task dependency graph*, i.e., a directed acyclic graph G that indicates the scheduling constraints.

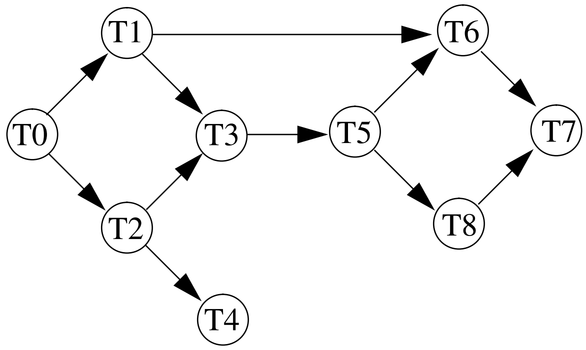


Figure 1: Example task dependency graph

and dependencies between tasks, in which the set of vertices V represents the tasks to be scheduled and the set of edges E represents precedence relations between them, as illustrated in Figure 1. An edge from node i to node j indicates that the execution of task i must precede execution of task j. Edges are weighted with values that represent amount of data transfer necessary between the dependent tasks.

- an $m \times m$ matrix of transfer *rate* values such that $rate[H_1, H_2]$ is the average rate of transfer of data between hosts H_1 and H_2 in bytes/second.

Given this, the goal is to determine an assignment of tasks to hosts $S(T_i), 0 \le i < n$ such that:

- the overall completion time is minimized
- all precedence relations are satisfied

Additionally, the following definitions are useful in the formulation of the problem:

- $pre(T_i)$ - the set of all tasks that *precede* task T_i in the task dependency graph. Thus, $T_k \in pre(T_i)$ is one of the tasks that precedes T_i.
- $d_{in}(T_i)$ the set of data input required to begin execution of task T_i.
- $d_{out}(T_k)$ the set of data output by a predecessor T_k of T_i. The data shared by T_i and its immediate predecessor T_k is given by $d_{in}(T_i) \cap d_{out}(T_k)$

Let k be a member of the set of $pre(T_f)$. The scheduling problem can then be recursively stated in the following way:

Find an assignment from T to H, $S(T_i)$, such that the *completion time* of task T_f, the last task in T to finish, executed on host H_j, is minimized, where

$$completionTime(T_f, H_j) = etc(T_f, H_j) + \max_{\forall k}\begin{bmatrix} \left|d_{in}((T_f) \cap d_{out}(T_k))\right| \times rate(S(T_k), S(T_f)) \\ + \\ completionTime(T_k, S(T_k)) \end{bmatrix}$$

A non-recursive algorithm to determine the completion time for an arbitrary schedule S is given in Figure 2. The algorithm resolves conflicts for multiple tasks that are assigned to the same host with a first-come, first-served policy.

2. GENETIC SIMULATED ANNEALING (GSA)

One way to approach the minimization problem is through combinatorial optimization. In this section, one such method, *Genetic Simulated Annealing* (GSA) [6] is overviewed.

Genetic Algorithms (GAs) were developed [7] to study the adaptive process of natural systems and to develop artificial systems that mimic the adaptive mechanism of natural systems. GAs differ from traditional optimization methods in that they use an encoding of the parameter set rather than the parameters themselves, and GAs search from a population of search nodes instead of from a single one. GAs use probabilistic transition rules, and typically consist of an initialization phase followed by repeated evaluation and reproduction phases [8].

```
where S is an assignment of tasks to hosts: S[i] returns the host assigned to task i
Let G be the dependency graph represented as an adjacency matrix: G[i,j] is 1
        if i depends on j, 0 otherwise
Let T[0 ...(n+1)] be an array of node structures initialized with T[i].ready_time:=0,
    T[i].start_time:=0, T[i].finish_time:=0 and T[i].#dependents:= Σ_{j=0}^{j=n+1} G[i, j]
Let node T[0] represent the start task and node T[n+1] represent the end task
Let Q be a priority Queue of T[i] based on T[i].ready_time
Let A[1 ... m] be an array of available structures initialized with A[j].available:=0

Push T[0] onto priority queue Q.
While not(Empty(Q)) do
    Let T[n]:= Pop(Q).
    Let T[n].start_time := max(T[n].ready_time,A[S[n]].available)
    Let T[n].finish_time := T[n].start_time + etc[n,S[n]]
    Let A[S[n]].available:=T[n].finish_time
    For each task 0 ≤ i ≤ (n + 1) If G[i,n] then begin
        Let v := T[n].finish_time + size(d_in(i) ∩ d_out(n) )/rate[S[n],S[i]]
        If v > T[i].ready_time then T[i].ready_time := v
        T[i].#dependents:=T[i].#dependents-1
        If T[i].#dependents==0 then Push T[i] onto Q
        end // for i
    end // while
Return(T[n+1].ready_time)
```

Figure 2: The COMPLETION_TIME(*S*) algorithm, that returns the completion time for the given assignment *S* of tasks to hosts. Ready_time is the time that all data needed by a task has arrived at the host. Start_time is the time that the task begins execution. Stop_time is the time that the task completes execution.

In the *initialization phase*, an initial population of the search nodes is randomly generated. In the *evaluation phase*, the fitness of each candidate solution is determined according to a client-specified fitness function. In the *reproduction phase*, new search nodes are generated in the search space by examining the fitness value of the search nodes and then applying genetic operators, e.g., crossover and mutation operators, to the search nodes. The evaluation and reproduction phases are repeated until the algorithm converges on a solution.

Simulated Annealing (SA) differs from GA in that SA algorithms start by generating a single candidate solution and then making successive random modifications. A temperature parameter is used to control the acceptance of modifications. Initially, the temperature is at high value that is decreased over time. If modified solution is found to have better fitness than its predecessor, it is retained and the previous solution is discarded. If modified solution has a lower fitness value, it is still retained with some probability directly related to the current temperature. As algorithm proceeds, temperature gets cooler making it less likely that unfavorable solutions will be accepted. In this way, the algorithm searches until converging on a single local solution.

GA and SA both independently are valid approaches toward problem solving. However they both have their own strengths and weaknesses. While GA can begin with a population of solutions in parallel, it has poor convergence properties. SA, on the other hand, has better convergence properties but cannot exploit parallelism. GSA blends both these approaches into a single approach in order to retain the strengths of both. GSA exploits the population-based model and crossover-mutation operator of GA by creating a multiplicity of candidate solutions. At the same time, GSA employs the temperature gradient property of SA by using a local acceptance policy based on the fitness of a new solution compared to its parent, and a probability based on a global temperature gradient. By using this approach, the algorithm maintains the diversity of GA while reducing the processing effort by following a temperature schedule as in SA. It has been shown that GSA can perform better for optimization problems than either GA or SA independently.[6]

3. GSA-BASED SCHEDULING

For the scheduling problem defined in Section 1, the goal is to find an assignment of n tasks to m hosts with a search space of size. To construct a GSA algorithm for the scheduling problem following elements are needed [8]:

- a string representation for the search nodes (i.e, an encoding of the parameter set)
- genetic operators, i.e., reproduction, crossover and mutation
- evaluation of the fitness or objective function
- determination of probabilities controlling the genetic operators
- generation of the initial temperature an annealing schedule

3.1. String Representation

A one-to-one correspondence from the tasks in the task dependency graph to the hosts in the heterogeneous network can be used as a string representation for the search nodes. Thus a list of m integers between 1 and n, such that an integer in the ith position in the list is the host on which task i is scheduled for execution can be used to as a coding of the parameter set. The initial population is randomly generated. A population of solutions in the form of a toroidal grid of size $P \times P$ is created. If $P = 16$, 256 potential solutions are created in the grid, each with an assignment of tasks to hosts and a fitness value.

3.2. Genetic Operators - Crossover, Mutation and Reproduction

The *crossover* operation is the exchange of one or more elements of a member string with one or more elements of another member string in a given population of strings. For the scheduling problem containing a grid of solutions, a single random direction and distance in the grid is generated for all solutions in each iteration to determine the visiting solution string for crossover. The crossover operation is then simply a switching of host assignments between the resident and visiting solution string for a subset of the set of tasks.

A *mutation* operation is an occasional random alternation of a value of a solution string, and is useful for avoiding premature convergence of the algorithm [8]. For this scheduling problem, mutation is implemented by randomly changing the value of the string.

Reproduction is a common operator in genetic algorithms. Reproduction forms a new population of strings by selecting in the old population based on their fitness values. The selection criterion is that the strings with higher fitness value should have a greater probability of surviving to next generation. In GSA, strings that have fitness values less than highest are not necessarily discarded, but instead are selected with a probability proportional to the current temperature (as in simulated annealing).

3.3. The Fitness Function

The fitness function for a given assignment is simply the reciprocal of the completion time. The the fitness function is maximized when the completion time is minimized.

3.4. Initial Temperature and Annealing Schedule

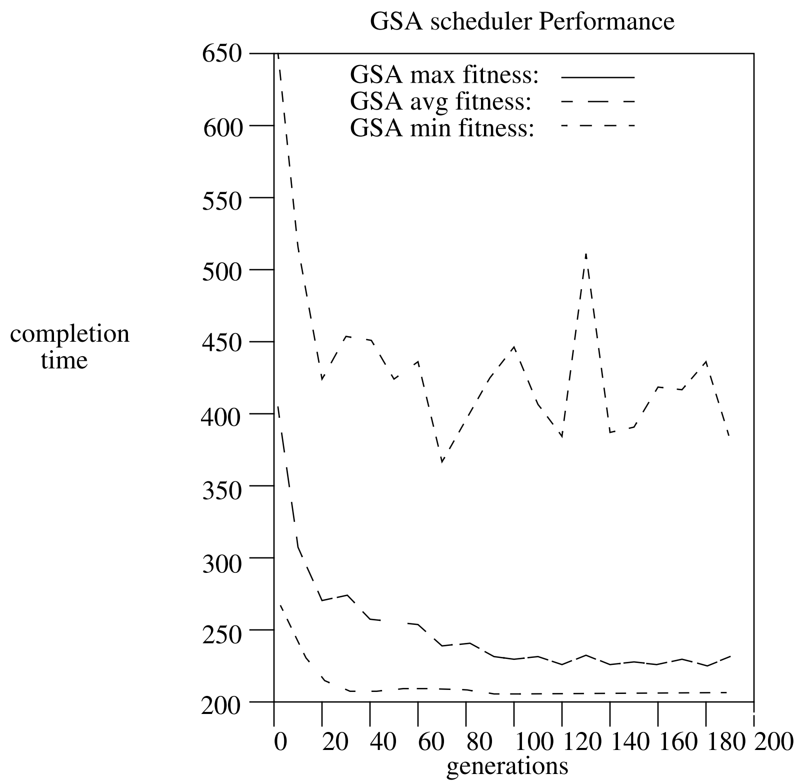
The initial temperature is set by generating the initial population and computing the average fitness for the population. The initial temperature is then set such that the initial acceptance probability for this population is nearest to 1.0, i.e., maximum. The annealing schedule may be generated by simply multiplying the current temperature by a constant less than 1.0. Thus temperature cooling is achieved. The probability of accepting unfavorable solutions in the selection process (reproduction) can be implemented as a function of the current temperature. If the difference in fitness between the original and new solution string is less than or equal to the current temperature, then the new string is selected, otherwise the old string is retained.

4. SIMULATION AND RESULTS

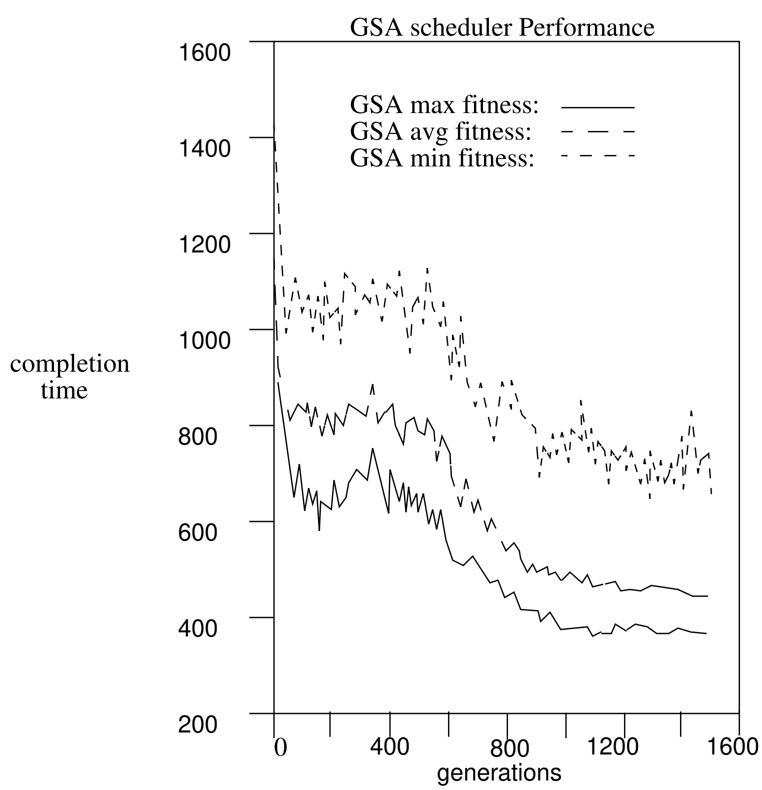
The GSA algorithm was implemented and tested on random problem sets for finding optimal schedules. The problem sets were generated randomly for 10 to 100 tasks and 5 to 35 hosts. The percentage dependence between tasks of a task graph was limited to 40% to keep the adjacency matrix sufficiently sparse and thus limit the scheduling constraints to a practical upper bound. The GSA algorithm used the following parameters throughout the simulations:

- population size = 256
- crossover probability = 0.6
- mutation probability = 0.05
- maximum generations = 1000

The simulations were performed on a HP 9000 Series 700 computer. The GSA algorithm converged in less than 1000 generations in all cases. For the sake of comparison, solutions were also generated using an entirely random scheduler to contrast them with the GSA sched



Graph 1: Progression of GSA algorithm on HC scheduling problem for 13 tasks on 6 hosts.



Graph 2: Progression of GSA algorithm on HC schediing problem for 45 tasks on 16 hosts.

hetero.	# iters	GSA	Optimal
LOW	60	120.00	119.90
MEDIUM	110	239.72	239.72
HIGH	140	1048.2	1048.2

LOW - (etc varies over 10 - 20 time units)
MEDIUM - (etc varies over 10 - 100 time units)
HIGH - (etc varies over 10 - 500 time units)

Table 1: Effect of variation in degree of heterogeneity of task execution times on GSA performance. (data/execution ratio= 1.0, connectivity = 40%, # of tasks < 20)

connectivity	# iters	GSA	Optimal
20%	40	86.10	86.10
40%	100	206.89	206.89
60%	110	224.26	224.26

Table 2: Effect of variation in degree of connectivity among tasks on GSA performance. (data/execution ratio= 1.0, # of tasks < 20)

uler. Graphs 1 and 2 show indicate that the GSA algorithm proceeds quickly towards improved solutions.

Graph 1 shows performance of GSA on a relatively small problem consisting of 13 tasks to be scheduled on 6 hosts, and with randomly generated task graph. It can be seen that the algorithm begins with an initial population having minimum completion time of 206.89 and converges in about 100 generations to yield an assignment with an optimal completion time, verified separately by the use of a branch-and-bound search of the entire solution space. Graph 2 shows performance of GSA on a larger problem of 45 tasks and 16 hosts requiring approximately 1400 generations to converge. The optimal solution to this problem is not known for Graph 2, because of the infeasibility of performing a complete search of the solution space for a problem of this size.

Using the same general problem characteristics as that for Graph 1, several additional simulations were performed to characterize the GSA approach under varying conditions. In Table 1, the effect of the degree of heterogeneity of execution times for each task across hosts is examined. In the first simulation, a low degree of heterogeneity is assumed, i.e., the execution time of each task varies only slightly for different hosts in the HC system. The second simulation assumes a moderate degree of heterogeneity, roughly equivalent to the base case in Graph 1. The third simulation assumes a large degree of heterogeneity.

In Table 1, the "#iters" column indicates the number of iterations needed by the GSA algorithm to converge to a solution. The "GSA" column indicates the fitness (i.e. the overall schedule time) for each simulation, and the "optimal" column represents the global minimum schedule, as determined by a complete search of the solution space. For each of the simulations the ratio of data size to execution time, or *data/execution ratio*, was chosen to be approximately 1.0, i.e. in general, the time needed for each task to transfer its data was roughly equivalent to the time needed to execute the task. The probability of a task having a data dependency with an earlier task, or *connectivity*, was chosen as 40% for these simulations.

In Table 2, the effect of the degree of connectivity among tasks is examined. In the first simulation, a low degree of connectivity is assumed, i.e., data dependencies between tasks are rare. The second simulation assumes a moderate degree of connectivity, roughly equivalent to the base case in Graph 1. The third simulation assumes a large degree of connectivity.

5. SUMMARY

In this paper we gave a formal definition of the HC scheduling problem and proposed a Genetic Simulated Annealing (GSA) based approach towards solving this problem. The proposed GSA algorithm was implemented and tested for various randomly generated problem sets. The simulation studies demonstrate that a GSA approach to scheduling tasks on HC systems is simple, elegant, and effective.

REFERENCES

[1] C. Leangsusksun, J. Potter and S. Scott, "Dynamic Task Mapping Algorithms for a Distributed Heterogeneous Computing Environment," in *Proceedings of the Heterogeneous Computing Workshop (HCW '95)*, April 1995, pp. 30-35.

[2] R. F. Freund, "Optimal Selection Theory for Superconcurrency," in *Proceedings of Supercomputing ' 89*, November 1989, pp. 47-50.

[3] R.F. Freund and H.J. Siegel, "Guest Editors Introduction: Heterogeneous Processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[4] A. Khokhar, V.K. Prasanna, M. Shaaban and C.L. Wang, "Heterogeneous Computing: Challenges and Opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[5] Y. Li, J. Antonio, H.J. Siegel, M. Tan and D.W. Watson, "Estimating the Distribution of Execution Times for SIMD/SPMD," in *Proceedings of the Heterogeneous Computing Workshop (HCW '95)*, April 1995, pp. 35-46.

[6] H.Chiu and N.F. Flann, "Parallel Simulated Annealing and Genetic Algorithms: a space of hybrid methods," in *Proceedings of the International Conference on Evolutionary Computation-PPSN III*, 1994, pp. 428-438.

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

[8] E.S. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No.2, February 1994, pp. 113-120.

AUTHOR BIOGRAPHIES

Pankaj P. Shroff is a Graduate Student at the Department of Computer Science at Utah State University. He received his B.E. degree in Computer Engineering at Maharashtra Institute of Technology, University of Pune, India in 1995. During his undergraduate study, he was the author on papers in the areas of concurrent programming, distance education through Multimedia, and object-oriented systems. His final year project in microprocessor based motor protection received the Best New Product award at the Industrial Exhibition '96 in Bombay, India. He is currently working on dynamic scheduling approaches for data dependent tasks in heterogeneous computing environments. His research interests include heterogeneous computing, genetic algorithms, scientific visualization and inter-networking.

Daniel W. Watson is an assistant professor at the Department of Computer Science at Utah State University. He received the B.S. degree in Electrical Engineering at Tennessee Tech University in 1985 and received the M.S.E.E. and Ph.D. degrees from the School of Electrical Engineering at Purdue University in 1989 and 1993. While at Purdue, he codeveloped an NSF-sponsored course on parallel programming, and supervised the development of ELP, the Explicit Language for Parallelism. He has coauthored numerous conference papers, journal articles, and book chapters on the topics of mixed-mode parallelism and heterogeneous computing. In heterogeneous computing, his recent work includes the static matching of program segments to machines and dynamic scheduling approaches for data-dependent tasks. Dan has been the recipient of faculty fellowships at Phillips and NRaD research laboratories for his work in heterogeneous computing, and is the Vice General Chair for the 1997 Heterogeneous Computing Workshop (HCW `97). He is a member of the IEEE Computer Society, a member of the Gamma Beta Phi, Tau Beta Pi, and Eta Kappa Nu honorary societies, and is a recipient of the distinguished Ronald G. Harber Service Award from HKN's Beta Chapter.

Dr. Flann holds a Ph.D in Computer Science and has been an assistant professor at the Computer Science Department, USU for 5 years. He is a specialist in the areas of machine learning, knowledge-based systems, and combinatorial optimization. He has supervised many successful research and development projects in these three areas. He is PI and CoPI of grants from Government and Private sources including an NSF funded project supporting education improvements in Machine Intelligence and a National Forest Service project supporting machine intelligence tools for ecological mapping and surveying. He has published over 30 technical articles and made many presentations at a diversity of technical conferences.

Richard F. Freund has been a senior computer scientist at the NRaD facility (formerly the US Navy's Naval Ocean System Center) in San Diego since 1987. His research interests in heterogeneous computing include crossovers, code profiling, network profiling, optimal matching, and analytical benchmarking. Freund is the technical founder of the Heterogeneous Computing project at NRaD and a leader in the application of heterogeneous processing to military command and control. Freund is a subject editor for the Journal of Parallel and Distributed Computing (JPDC). He is also the foundational organizer of the annual Heterogeneous Computing Workshop held in conjunction with the International Parallel Processing Symposium.