Effect of Representation Methods on Time Complexity of Genetic Algorithm based Task Scheduling for Heterogeneous Network Systems

Hwa-Sung Kim
Electronics and Telecommunications Research Institute
161 Kajong-Dong, Yusong-Gu, Taejon, 305-350, Korea
hwkim@etri.re.kr

Abstract

This paper analyzes the time complexity of Genetic Algorithm based Task Scheduling (GATS) which is designed for the scheduling of parallel programs with diverse embedded parallelism types in a heterogeneous network systems. The analysis of time complexity is performed based on two representation methods (REIA, REIS) which are proposed in this paper to encode the scheduling information. And the heterogeneous network systems consist of a set of loosely coupled parallel and vector machines connected via a high-speed network. The objective of heterogeneous network computing is to solve computationally intensive problems that have several types of parallelism, on a suite of high performance and parallel machines in a manner that best utilizes the capabilities of each machine. Therefore, when scheduling in heterogeneous network systems, the matching of the parallelism characteristics between tasks and parallel machines should be carefully handled in order to obtain more speedup. This paper shows how the parallelism type matching affects the time complexity of GATS.

1. Introduction

Parallel and Distributed Computing Systems can be classified into two classes: homogeneous systems and heterogeneous systems. Heterogeneous systems are composed of multiple dissimilar machines which corporate in solving a problem. This paper focuses on a heterogeneous network systems which consist of a set of loosely coupled parallel and vector machines connected via a high speed network as shown in Fig. 1. The objective of heterogeneous network computing is to solve computationally intensive problems, which

have several types of parallelism, on a suite of high performance and parallel machines in a manner that best utilizes the capabilities of each machine [1,2,3,4,5].

Many large-scale scientific applications have more than one type of embedded parallelism, such as SIMD, MIMD and vector parallelism types, in its various code segments. Since it is unlikely that a single parallel machine would execute this mixed-type of computations with the maximum possible speedup, a homogeneous system can not achieve the optimal speedup for these applications. Therefore, it is more efficient to allocate the different segments of a program with different types of parallelism to various parallel machines that can execute the assigned segments with the optimal speedup. When the tasks are allocated to the parallel machines, each task should be allocated to an individual machine so that the parallelism type of the task and the machine can be matched as close as possible. In each machine, all the tasks assigned to it are scheduled in order to minimize the execution time. In this case, however, the network overhead required for using the various machines should not offset the advantage obtained by assigning the segments of the program to the machines with matching parallelism type. Unlike homogeneous systems, a super-linear speedup can then be achieved when using a heterogeneous network systems [1].

In recent years, the adaptive search methods, such as genetic algorithms and simulated annealing algorithms, have become popular as an effective means of solving combinatorial optimization problems because of their general applicability. Genetic Algorithms introduced by Holland [6] are stochastic search algorithms based on principles of population genetics. They have been used for various optimization problems such as the traveling salesman problem [7] and classifier systems [8]. Even though the list scheduling algorithms [9] have been a dominant method as a polynomial time heuristic for scheduling parallel programs

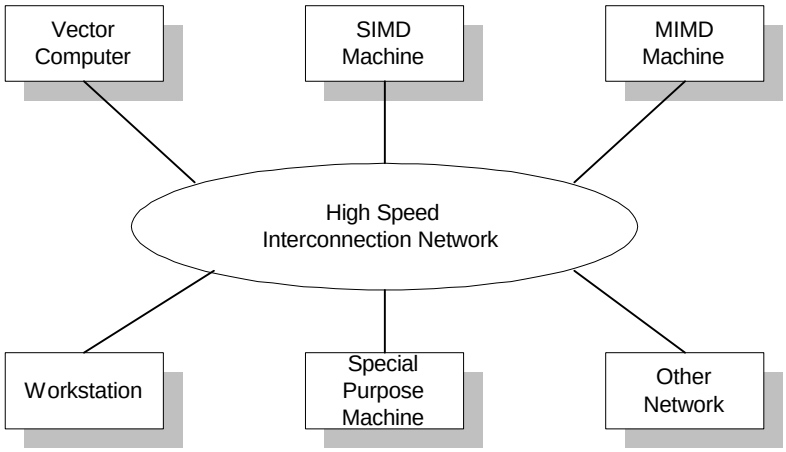


Figure1. Example of Heterogeneous Network Systems

into multi-processors or multi-computers, there have been many attempts to apply Genetic Algorithms in task scheduling for the homogeneous parallel systems [10, 11] and heterogeneous parallel systems [12]. In this paper, we first propose two representation methods to encode the scheduling information for GATS in heterogeneous network systems. Using these representation methods, the time complexity of GATS is analyzed.

2. Formulation of the Problem

The heterogeneous network systems used in this paper is a point-to-point network of multiple high performance and parallel machines, each of which has a specific parallelism type. The heterogeneous network system is represented by a *Network Graph* as defined below.

Definition 1: A heterogeneous network H is represented by an undirected network graph $G_H = <V_H, E_H>$, where $V_H = \{M_{i,k}, i \in \{1, 2, \ldots, m\}, k \in \{1=\text{SISD}, 2=\text{SIMD}, 3=\text{MIMD}, 4=\text{vector}, \ldots\}\}$. For the convenience, the machine is sometimes denoted by M_i (dropping the integer k) when the parallelism type of the task is not important in the discussion. Each machine $M_{i,k}$ in V_H is associated with a tuple - $[k, R_{i,k}]$ where k is the parallelism type of machine M_i and $R_{i,k}$ is the relative performance of machine $M_{i,k}$ to the fastest machine of type k (if machine $M_{i,k}$ is the fastest among the machines of parallelism type k, then $R_{i,k} = 1$ otherwise $R_{i,k} > 1$). E_H is a set of edges; $E_H \subseteq V_H \times V_H$ such that $(M_i, M_j) \in E_H$ iff M_i is connected to M_j through a direct link. Each edge is associated with an integer representing the communication cost that is incurred when sending a unit of information between the two machines, i.e. this integer is an estimate of the transfer rate and type conversion overhead between the machines neglecting the setup time. Fig. 2(a) shows an example of a network graph. It is assumed that each machine in the heterogeneous network may execute at most one task at a time.

On the other hand, a parallel program consists of a number of cooperating and communicating tasks, each of which is characterized by a parallelism type such as SISD, SIMD, MIMD, or a vector type. The behavior of the program is described by a *Task Graph* as defined below.

Definition 2: A program P is represented by a directed, acyclic task graph $G_P = <V_P, E_P>$ where $V_P = \{t_{i,k}, i \in \{1, 2, \ldots, n\}, k \in \{1=\text{SISD}, 2=\text{SIMD}, 3=\text{MIMD}, 4=\text{vector}, \ldots\}\}$ is a partition of P consisting of n tasks each of which has a parallelism type k. For the convenience, the task is sometimes only denoted by t_i (dropping the integer k). Each task

$t_{i,k}$ in V_P is associated with a tuple - [k, ($D_{k,1}$,... $D_{k,l}$, ..., $D_{k,L}$), $O_{i,k}$] where $D_{k,l}$ is a coefficient that represents the type mismatch penalty between task $t_{i,k}$ and machine $M_{j,l}$ (if $k = l$, then $D_{k,l}$ =1 otherwise $D_{k,l} > 1$), L is the number of available machine types in the heterogeneous network under consideration. $O_{i,k}$ is the optimal execution time of $t_{i,k}$ on the its best matching machine. E_P is a set of edges; $E_P \subseteq V_P \times V_P$ such that $(t_i, t_j) \in E_P$ if and only if t_i must execute before t_j due to data dependency and/or communications between the two tasks. Each edge is associated with an integer representing the amount of information to be transferred between two tasks. Fig. 2(b) shows an example of a task graph. It is

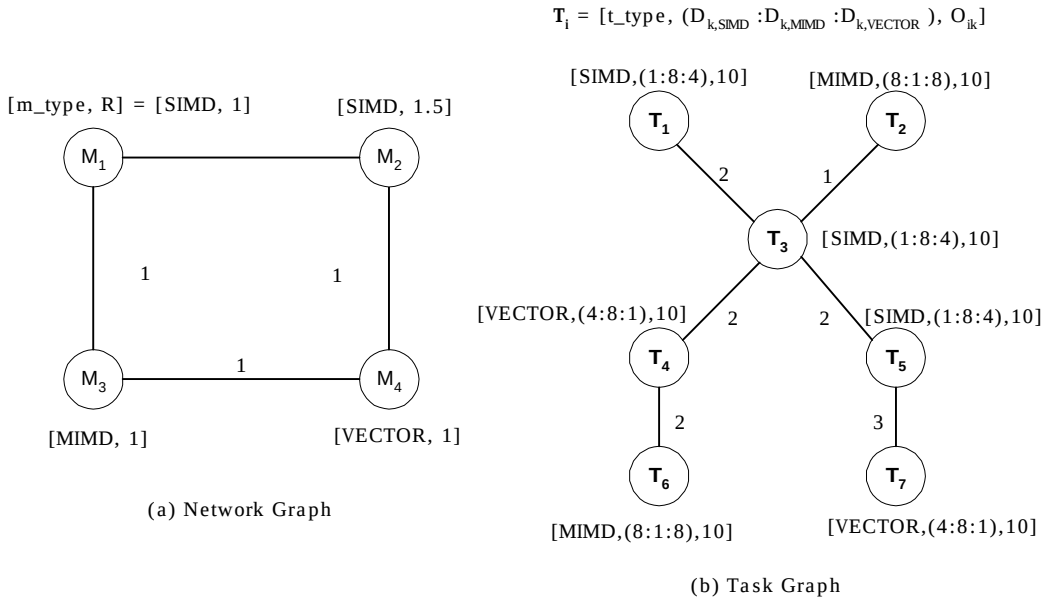


Figure 2. Example of Network and Task Graph

assumed that the execution of tasks in the machines is nonpreemtive, i.e. once the task begins execution, it executes until its completion.

After mapping the tasks of a program P to the machines in a heterogeneous network H, another task graph called an *Allocated Task Graph* is obtained. In an allocated task graph, each task is assigned a single value, which is its execution time on a specific machine, and each edge is assigned the actual communication cost between two tasks. Let $T(t_{i,k}, M_{j,l})$ be the execution time of task $t_{i,k}$ on machine $M_{j,l}$. We can describe $T(t_{i,k}, M_{j,l})$ by the product: $D_{k,l} \cdot R_{j,l} \cdot O_{i,k}$. If the parallelism types of a task does not correspond to that of a machine, the execution time will be $D_{k,l}$ times the optimal execution time $O_{i,k}$. On the other hand, DHSS can be composed of machines with various types and various performance even in same type. Therefore, a task can be assigned to the machine, which is not the optimal

choice even though their types match. $R_{j,l}$ incorporates the non-optimal machine choice although type matching succeeds. R_{jl} should be 1 if a parallel machine M_{jl} on which task t_{ik} is allocated is the fastest machine among the machines with parallelism type l. $O_{i,k}$ represents the optimal mapping of task t_i of parallelism type k to the best machine of type k.

Let $C(t_i, M_{j,l}, t_{i\phi} M_{j\phi l\phi})$ be the actual communication cost between a task $t_{i,k}$ allocated to machine $M_{j,l}$ and any its parent task $t_{i\phi k\phi}$ allocated to machine $M_{j\phi l\phi}$ The execution of parallel tasks using various machines accompanies the communication overhead. Moreover in heterogeneous network systems, more speedup can be achieved by executing the task nodes which lie on sequential path on different machines. This will introduce the communication overhead. The communication overhead is decided by the amount of the data $A_{i,i\phi}$ transferred between any two tasks t_i and $t_{i\phi}$ the transmission rate $H_{j,j\phi}$ between any two machines M_j and $M_{j\phi}$, and the conversion time of the data representation and the synchronization time $F_{l,l\phi}$ between any two machines of type l and $l\phi$

After all the tasks that are mapped to the same machine are totally ordered, the allocated task graph is changed into a *Scheduled Task Graph* in which the mapping of tasks to machines in the network and any execution order between tasks are fixed. In a scheduled task graph, additional edges to represent the execution order between tasks assigned to the same machine can be added and the transitive edges are removed. The scheduling problem can be described as finding a mapping function Π of the tasks of a program P into the machines of a network H, i.e. Π: $V_P \rightarrow V_H$, and the execution order of all the tasks that are mapped to the same machine such that schedule length is minimized. The schedule length is equal to the summation of execution times $T(t_{i,k}, M_{j,l})$ and the communication costs $C(t_i, M_{j,l}, t_{i\phi} M_{j\phi l\phi})$ along the critical path in a scheduled task graph. The critical path in a scheduled task graph is the longest path between any starting task and any ending task in the graph. In this case, the communication time between two tasks executing on the same machine is assumed to be zero time unit. Especially in the heterogeneous network systems, the parallelism type matching heavily affects the schedule length [16].

3. Genetic Algorithm based Task Scheduling (GATS)

GATS starts with an initial set of schedules (solutions to the scheduling problem) and then improves upon these schedules using an iterative process. Each schedule is represented by a *string*. A set of strings that GATS operates upon is called a *population*. During each iteration, called a generation, the current population is evaluated by rating the strings using a measure of fitness. Based on the fitness values and according to a *selection strategy*, two strings are chosen at a time to act as parents. A number of *genetic operators* are

then applied to the parents to create new individuals by combining the features of both parents. This process continues until a population is generated which is the same size as the previous generation. At the end of each iteration, transformations and replacement strategies are applied to the produced population and the previous population to create the new population [18]. In GATS, the elitist strategy can be used to preserve the best string obtained so far when replacing the old population by a new one. The elitist strategy complements the disruptive effects of the crossover and the mutation operations. The scaling strategy is also used to regulate the level of competition between individual strings in a population. GATS terminates whenever a termination condition is met (either the specified maximum number of generations is reached or no improvement in the fitness of a population is detected for a number of generations).

A genetic algorithm tries to maximize an objective function, whose value is called the fitness of the schedule, through generations. For example, the reproduction operator produces strings for the next generation according to their fitness values. Strings with higher fitness values have a higher probability of contributing one or more offspring in the next generation. In this paper, the fitness of a string is defined as the difference between the schedule length of that string and the maximum schedule length of the strings in a population as follows :

$$Fitness(s) = K - schedule\_length(s) \quad \text{where } K = \max_{s \in S} schedule\_length(s)$$

The smaller the schedule length, the fitter the string. Therefore, our task is to find a global maximum for the fitness function $F : S \rightarrow R^+$ where S is a set of strings which have their own schedule length, and R^+ denotes the positive real numbers or fitness.

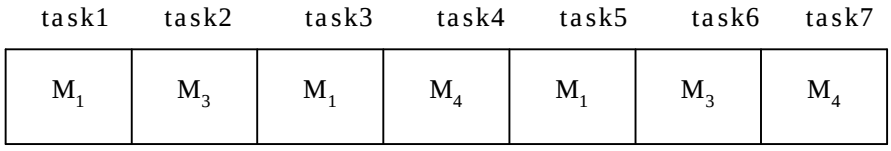
4. Representation of Scheduling Information

For the genetic algorithms to be used for the scheduling problem in heterogeneous network systems, the scheduling information by which the schedule length can be computed should be encoded using a binary or decimal string. In this paper, two representation formats are proposed for encoding the scheduling information: the *Representation by Encoding an Instance of Allocation (REIA)* and the *Representation by Encoding an Instance of Schedule (REIS)*. Given a task graph derived by partitioning the parallel program with diverse embedded parallelism types, both the execution time of each task node and the communication cost between any communicating tasks are not known. They can be fixed after the tasks are allocated to the specific machines. In addition to these parameters, the execution order among the tasks allocated to the same machine (*local schedule*) should

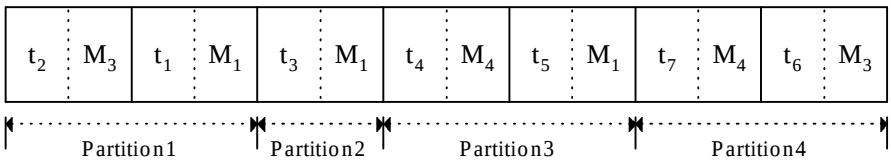
also be fixed in order for the schedule length to be calculated. Each string encoded by REIA represents one instance of allocation of the task graph into the set of parallel and vector machines. Each string encoded by REIS represents both the allocation of tasks and their local schedule, which is an instance of a *global schedule*. The schedule length of both representations is calculated using a cost function.

4.1 Representation by REIA

In the case of REIA, each string contains only information about the allocation of the tasks to the set of parallel machines. The local scheduling order among the tasks allocated to the same machine should be decided at the time when the strings are evaluated to obtain the schedule length using a cost function. In order for a string to be constructed using REIA, each task in a task graph should be labeled and assigned to one segment in the string in sequence. For example, the labeling of tasks is done in breadth-first fashion in Fig. 2(b). Fig. 3(a) shows an example of a string represented by REIA for the task graph in Fig. 2(b), in which the allocation information is given arbitrarily. The string in Fig. 3(a) consists of 7 segments because there are 7 task nodes in the task graph, and each segment



(a) Example of REIA



(b) Example of REIS

Figure 3. Example of String Representation

is assigned to one task permanently. As illustrated in Fig. 3(a), the encoded value of each segment in a string represents the machine identifier of any parallel or vector machine in the network, which means that the task representing that segment is allocated to the parallel or vector machine whose machine identifier is encoded in that segment. For example, in Fig. 3(a), task 1 which is assigned to the first segment, is allocated to machine M_1 and task 2 which is assigned to the second segment, is allocated to machine M_3 etc. Each machine

in the network is assigned a unique machine identifier regardless of its parallelism type. In REIA, each segment in a string consists of $\lceil \log_2 m \rceil$ bits, and $\lceil \log_2 m \rceil * n$ bits are needed for the representation of a REIA string, where m is the number of machines in the network. The REIA string should fulfill the following condition to represent a legal allocation information:

Condition for Legal REIA String: If the machine identifiers in each segment of the REIA string are legal, the corresponding REIA string is also legal.

4.2 Representation by REIS

In order to use a REIS representation with GATS, a task graph needs to be partitioned into a set of subgroups according to their precedence relations. This process of the precedence partitions was originally adopted in the Optimal Preemptive Scheduling (PS) algorithm by Muntz and Coffman [13]. It was also used later by Hou *et al.* [11] and Iverson *et al.*[14]. During this process, the nodes in a task graph are partitioned into a sequence of disjoint subsets according to the height information of the nodes such that all nodes in a subset are independent of the others. All nodes in the same subset are candidates for parallel execution or group scheduling.

The precedence partition process can be done in two ways: *latest precedence partition* and *earliest precedence partition*. The former assigns height 1 to every task node, which has no immediate successor. The height as

signment of this case is done in a manner that defers a task initiation to the latest possible time. On the contrary, The latter assigns depth 1 to every task node, which has no immediate predecessor, as in Eq. (1). The height value of each task using the earliest precedence partition is calculated using Eq. (2). The height assignment of the latter case is done in a manner that assigns the task to any available machine as early as possible if its precedence constraints are satisfied [15]. In this paper, the earliest precedence partition is adopted to represent any scheduling information using REIS. As an example, the task graph in Fig. 2(b) is partitioned into the subsets {1,2}, {3}, {4,5} and {6,7} by the earliest precedence partition.

$$\text{depth}(t_i) = \begin{cases} \text{if } t_i \text{ has no parent} & 1 \\ \text{else} & 1 + \max\limits_{t_j \in \text{parent}(t_i)} \text{depth}(t_j) \end{cases} \quad (1)$$

$$\text{height}(t_i) = \max_{1 \le j \le n} \text{depth}(t_j) - \text{depth}(t_i) + 1 \quad (2)$$

Since REIS string includes both the allocation and local scheduling information directly in the strings, the representation of scheduling information using REIS is more natural than REIA. Each REIS string consists of *n* segments and the encoded value of each segment in a string represents *(task_id, allocated_machine_id)* pair. The *task_id* is a unique task identifier and the *allocated_machine_id* is a unique machine identifier of any parallel or vector machine in the network. Each segment in a string consists of $\lceil \log_2 n \rceil + \lceil \log_2 m \rceil$bits. And $(\lceil \log_2 n \rceil + \lceil \log_2 m \rceil)^* n$ bits, which is larger than the length of strings in REIA, are needed for the representation of a REIS string. The legality condition for a REIS string is defined as follows. Any REIS string, which fulfills the following four conditions is a legal string that represents a schedule of a task graph.

Condition for Legal REIS String:: 1) Both the task identifier and the machine identifier in each segment of the REIS string should be legal. 2) All the tasks of the task graph should be included in the string. 3) Each task should be present only once in a string. 4) All tasks in the same partition should be positioned in sequence in the string and each partition should be positioned in descending order according to their height value.

If all of encoded values of the *n* segments are legal (condition 1) and each segment is assigned to exactly one task (conditions 2 & 3), then every task of a task graph will be present with its legal allocation information. Also, the tasks should be totally ordered in sequence in the string. The tasks in the same partition have the same height and the relative positions of tasks in that partition imply the execution order of tasks. However, the relative positions of tasks in a partition are meaningless if the machine identifiers for the tasks are different because those tasks do not have any precedence constraints. Further, since they can not intermix with other tasks with different heights, the following relation should exist when t_i is positioned before t_j in REIS string (condition 4): height(t_i) = height(t_j) if t_i, t_j belong to the same partition, height(t_i) > height(t_j) if t_i, t_j belong to different partitions. This condition makes the representation obey the precedence relations between tasks.

Fig. 3(b) shows an example of the REIS representation for the task graph shown in Fig. 2(b). This task graph has 4 partitions - {t_1, t_2}, {t_3}, {t_4, t_5} and {t_6, t_7} - using the earliest precedence partition as shown in Fig. 3(b). Each partition is assigned the same number of segments as the number of tasks in the partition. In Fig. 3(b), partition 1 is assigned 2 segments while partition 2 is assigned 1 segments. (t_2, M_3) and (t_1, M_1) are encoded in the first two segments, which implies that task 1 is allocated to machine 1, task 2 is allocated to machine 3 and task 2 is executed before task 1 if they are allocated to the same machine. But in this case, the execution order is meaningless because the two tasks are allo-

cated to different machines. The next segment has (t_3, M_1), which implies that task 3 is allocated to machine 1.

Any string represented by REIS, which violates any one of the four conditions does not represent a legal schedule. In the case of REIA, the second and third conditions are satisfied automatically by the nature of the representation, and the fourth condition is meaningless because the precedence information is not included in the representation. When implementing GATS using REIS, the method for generating initial solutions and the genetic operators should be carefully designed so that above mentioned four conditions are satisfied.

4.3 String Space and Solution Space

Using binary strings, the size of the *string space* of REIA for scheduling *n* tasks onto *m* parallel machines in heterogeneous network is given in Eq. (3). There are many illegal strings, however, in this string space. If GATS is implemented efficiently, the illegal strings will not be introduced while running these algorithms. The *legal string space* size is given in Eq. (4). Given the priorities of the tasks, which are used to decide the execution order between tasks in a machine, the scheduling problem using REIA can be stated as finding the optimal allocation of the tasks onto the parallel machines.

$$\text{Size of string space for REIA: } (2^{\lceil \log_2 m \rceil})^n \tag{3}$$

$$\text{Size of legal string space for REIA: } m^n \tag{4}$$

The size of the string space of REIS and the size of the legal string space is given in Eq. (5) and (6) respectively, where H represents the number of partitions in the task graph and $|partition(h)|$ represents the size of the partition h.

$$\text{Size of string space for REIS: } (2^{(\lceil \log_2 n \rceil + \lceil \log_2 m \rceil)})^n \tag{5}$$

$$\text{Size of legal string space for REIS : } \prod_{h=1}^{H} |partition(h)| \cdot m^n \tag{6}$$

On the other hand, the *solution space* for scheduling *n* tasks onto *m* parallel machines is actually slightly larger than the legal string space of REIS because some tasks in different partitions can be independent and can run concurrently. The difference between the size of the legal string space of REIS and the solution space is represented by symbol Δ_1. Therefore the size of the solution space can be represented by

$$\prod_{h=1}^{H} |partition(h)|! \cdot m^n + \Delta_1 \quad (7)$$

Comparing Eq. (4) and (6), GATS using REIA might converge faster than those using REIS because of their smaller string space than REIS.

4.4 Improving Search Efficiency by Restricting String Space

An exhaustive search of the solution space is needed in order to guarantee that a global minima is reached. However, such a search can not usually be achieved with a polynomial time complexity. In reality, the approximation search methods, which try to achieve near-optimal solutions with a polynomial time complexity, are more reasonable and are preferably used. Genetic algorithms are known to be one of such kinds of search methods. However, one of the disadvantages of the genetic algorithms is their slow convergence to the final solution compared to heuristic algorithms. When scheduling in heterogeneous network systems, important domain specific knowledge can be incorporated in the algorithms to reduce the solution space and eventually to expedite the convergence time.

In homogeneous systems, minimizing the inter-processor communication cost and balancing the machine load are two important factors in the mapping strategy. But in heterogeneous network systems, matching the parallelism types is much more important than these two factors. If the superlinear speedup is to be achieved using heterogeneous network systems, most of the tasks need to be assigned to their best matching machines in terms of the parallelism type [16]. In [12], this domain specific knowledge is incorporated in GATS to expedite the convergence to a good solution. However, it should be noted that such a type match might not always lead to an optimal solution. For example, allocating a task to a non-matching parallelism type machine may improve the overall objective function. In this context, some techniques should be incorporated to introduce the intermittent allocation of a task to a non-matching parallelism type machine [12].

If the allocation is restricted to the machines with the same parallelism type, the size of the legal string space of REIA is reduced to the size described in Eq. (8). Also, the size of the legal string space of REIS is reduced to Eq. (9), where $m_r(t_i)$ is smaller than m for all i and represents the number of machines having the same parallelism type ρ as task t_i. Further, the size of the solution space is reduced to the size described by Eq. (10). The difference between the size of the restricted legal string space of REIS and the restricted solution space is represented by symbol Δ_2.

$$\text{Size of restricted legal string space for REIA} \quad : \quad \prod_{i=1}^{n} m_r(t_i)$$

(8)

Size of restricted legal string space for REIS : $\prod_{h=1}^{H} |partition(h)|! \cdot \prod_{i=1}^{n} m_r(t_i)$ (9)

Size of restricted solution space : $\prod_{h=1}^{H} |partition(h)|! \cdot \prod_{i=1}^{n} m_r(t_i) + \Delta_2$ (10)

The size of the restricted legal string space of REIA given in Eq. (8) is smaller than the legal string space size in Eq. (4).

$$\prod_{i=1}^{n} m_r(t_i) \le (max\, m_r)^n \le m^n \quad (11)$$

The size of the restricted legal string space of REIS given in Eq. (9) is also smaller than the legal string space size in Eq. (6).

$$\prod_{h=1}^{H} |partition(h)|! \cdot \prod_{i=1}^{n} m_r(t_i) \le \prod_{h=1}^{H} |partition(h)|! \cdot (max\, m_r)^n \le \prod_{h=1}^{H} |partition(h)|! \cdot m^n \quad (12)$$

Eq. (11) and (12) imply that GATS using restricted string space of either REIA or REIS might converge faster than those using non-restricted string space.

4.5 Calculation of Estimated Schedule Length

REIA string includes only the allocation information but not the execution order of all the tasks that are allocated to the same machine and the execution order in each machine can be obtained in two different ways. In the first method, called *random scheduling*, the execution order between tasks is decided randomly when a conflict arises in a machine (i.e. two or more independent tasks can be scheduled at the same time interval). In the second method, called *critical path scheduling*, each task t_i in the task graph is initially assigned a weight that is equal to the sum of the expected execution time of all the tasks that exist in the longest path from t_i to one leaf of the acyclic graph. When a conflict arises, this weight is used to decide the execution order. When the schedules are encoded using REIA, the time complexity for calculating the schedule length is $O(n^3)$. It takes $O(n)$ time to calculate the finishing time of a task when it runs on specific machine because a task can have an arbitrary number of predecessors in the arbitrary task graph. And it takes $O(n)$ time to decide which task should be run next among the ready tasks. Finally there are n tasks in a task graph.

On the contrary, the string of REIS describes a schedule itself, which eliminates the extra overhead to decide the execution order between tasks allocated to the same machine.

The time complexity for calculating the schedule length when the schedule is encoded using REIS, is $O(n^2)$. Therefore REIS takes less time to calculate the estimated schedule length than REIA.

5. Convergence Time Analysis

In this section, the time complexity of GATS is described in terms of the number of generations for the convergence using the schema theorem. The time complexity is analyzed based on the work done in [17]. In the following analysis, no mutation or crossover operations are applied.

5.1 Convergence Time Analysis for REIA

Let $P_j(i,M_1)$ represent the proportion of segments set to an arbitrary machine identifier M_1 at generation i for a particular segment position j in a string, which is interpreted as: task t_j is assigned to machine M_1 at generation i. In the following discussion, $P_j(i,M_1)$ is denoted by $P(i,M_1)$ for the convenience. The value of a particular segment position in a string is defined over the alphabet set $\{M_1, M_2, \ldots, M_m\}$, in the case of the nonrestricted string space, where M_i is a machine identifier and m is the number of machines in the network. In the case of the restricted string space, the alphabet set becomes $\{M_1, M_2, \ldots, M_{m_\Theta}\}$, where m_Θ is defined as Eq. (13). $m_r(t_i)$ represents the number of machines having the same parallelism type as task t_i of parallelism type ρ.

$$\prod_{i=1}^{n} m_r(t_i) = (m_\Theta)^n \tag{13}$$

In the following, only the nonrestricted case is discussed. The restricted case can be discussed in a similar fashion. Let $f(j, M_1)$ represents the average fitness value of all strings whose j^{th} segment have the value M_1. The proportion of the strings whose j^{th} segment has value M_1 at generation $g+1$ can be obtained using the schema theorem as follows [17].

$$P(g+1, M_1) = \frac{f(j,M_1)\cdot P(g,M_1)}{f(j,M_1)\cdot P(g,M_1) + f(j,M_2)\cdot P(g,M_2) + \cdots + f(j,M_m)\cdot P(g,M_m)} \tag{14}$$

where $P(g,M_1) + P(g,M_2) + \cdots + P(g,M_m) = 1$ and

$$1 - P(g,M_1) = P(g,M_2) + P(g,M_3) + \cdots + P(g,M_m) \tag{15}$$

If we define $f(j,\overline{M_1})$ as follows,

$$f(j,\overline{M_1})\cdot(P(g,M_2)+\cdots+P(g,M_m)) = f(j,M_2)\cdot P(g,M_2)+\cdots+f(j,M_m)\cdot P(g,M_m) \quad (16)$$

then the following Eq. (17) can be obtained by substituting Eq. (16) into Eq. (14) and Eq. (18) can be obtained by substituting Eq. (15) into Eq. (17).

$$P(g+1, M_1) = \frac{f(j,M_1)\cdot P(g,M_1)}{f(j,M_1)\cdot P(g,M_1)+f(j,\overline{M_1})\cdot(P(g,M_2)+\cdots+P(g,M_m))} \quad (17)$$

$$= \frac{f(j,M_1)\cdot P(g,M_1)}{f(j,M_1)\cdot P(g,M_1)+f(j,\overline{M_1})\cdot(1-P(g,M_1))} \quad (18)$$

Here, the fitness ratio r is defined as $f(j,M_1) / f(j,\overline{M_1})$ which will be assumed to be constant through the generations. Using the notation r, Eq. (18) is transformed to

$$P(g+1, M_1) = \frac{r\cdot P(g,M_1)}{1+(r-1)\cdot P(g,M_1)} \quad (19)$$

The recurrence relation (19) is solved to produce (20) where $P(0,M_1)$ is the proportion of segments set to an arbitrary value M_1 at generation 0 for a particular segment position j in a string

$$P(g, M_1) = \frac{r^g\cdot P(0,M_1)}{(1-P(0,M_1))+r^g\cdot P(0,M_1)} \quad (20)$$

Eq. (20) can be rewritten as follows, where g_c represents the number of generations, which is needed for the genetic algorithm to converge.

$$g_c = \frac{\ln[P(g_c,M_1)\cdot(1-P(0,M_1))/P(0,M_1)\cdot(1-P(g_c,M_1))]}{\ln(r)} \quad (21)$$

worst case analysis

For the worst case analysis, $P(0, M_1)$ is given as $1/PS$ where PS is the population size, and $P(g_c,M_1)$ is given by 1- γ where γ is a tolerance parameter. Here, γ is set to $1/PS$. Using these two assumptions, the worst case for g_c is

$$g_c = \frac{\ln[(PS-1)^2]}{\ln r} \quad (22)$$

Therefore, the time complexity for the convergence of GATS using REIA in the worst case is $O(n^3 \cdot PS \cdot \ln PS)$ which is same for both the nonrestricted and restricted string space cases. $O(n^3)$ is the time to calculate the cost or the fitness of a solution (a string). This calculation is needed *PS* times in a generation.

average case analysis

For the average case, $P(0, M_1)$ is given by $1/m$ for nonrestricted string space case, $1/m_\Theta$ for the restricted string space case. $P(g_c,M_1)$ is given by $1-1/PS$ for both cases. Using these two terms, g_c for the average case is obtained as

$$g_c = \frac{\ln[(PS-1)(m-1)]}{\ln r} \quad \text{--------} \quad \text{(nonrestrcited)} \tag{23}$$

$$g_c = \frac{\ln[(PS-1)(m_\Theta-1)]}{\ln r} \quad \text{--------} \quad \text{(restricted)} \tag{24}$$

Therefore, the time complexity for the convergence of GATS using REIA on the average is $O(n^3 \cdot PS \cdot \ln(PS \cdot m))$ for the nonrestricted string space case, and $O(n^3 \cdot PS \cdot \ln(PS \cdot m_\Theta))$ for the restricted string space case. Because $m_\Theta < m$, a genetic search in the restricted string space tends to converge faster than that of the nonrestricted string space.

5. 2 Convergence Time Analysis for REIS

For REIS, let $P_j(i,t_1,M_1)$ represent the proportion of segments set to an arbitrary value (t_1,M_1) at generation *i* for a particular segment position *j* in a string. $P_j(i,t_1,M_1)$ is interpreted as: at generation *i*, task t_1 is assigned to machine M_1, and t_1 is executed after all the tasks located in $0^{th} \sim (j-1)^{th}$ segment position but it is executed before all the tasks located in $(j+1)^{th} \sim (n-1)^{th}$ segment position. In the following discussion, $P_j(i,M_1)$ is denoted by $P(i,M_1)$ for the convenience. In the case of nonrestricted string space, the value of a particular segment position in a string is defined over the combinations of the alphabet set $\{t_{h1}, t_{h2}, \ldots, t_{|partition(h)|}\}$ and $\{M_1, M_2, \ldots, M_m\}$. t_{hi} is a task identifier with height *h* in a task graph and $|partition(h)|$ represents the number of tasks in a partition *h* such that $partition(h) = \{ t_i \mid height(t_i) = h, 1 \le i \le n \}$. *n* is the number of task nodes in a task graph. M_k is a machine identifier and *m* is the number of machines in the network. In the case of re-

stricted string space, the alphabet set becomes $\{t_{h1}, t_{h2}, \ldots, t_{|partition(h)|}\}$ and $\{M_1, M_2, \ldots, M m_\Theta\}$, where m_Θ was defined in Eq. (13).

Let $f(j,t_{h1},M_1)$ represent the average fitness value of all strings whose j^{th} segment have a value (t_{h1},M_1). The proportion of the strings whose j^{th} segment have a value (t_{h1},M_1) at generation $g+1$ can be obtained by schema theorem. Eq. (25) can be obtained using a similar method as REIA.

$$P(g+1,t_{h1},M_1) = \frac{r \cdot P(g,t_{h1},M_1)}{1+(r-1)\cdot P(g,t_{h1},M_1)} \tag{25}$$

The recurrence relation in Eq. (25) is solved to produce Eq. (26), where $P(0,t_{h1},M_1)$ is the proportion of segment set to an arbitrary value M_1 at generation 0 for a particular allele position j in a string.

$$P(g,t_{h1},M_1) = \frac{r^g \cdot P(0,t_{h1},M_1)}{(1-P(0,t_{h1},M_1))+r^g \cdot P(0,t_{h1},M_1)} \tag{26}$$

Eq. (26) can be rewritten as follows, where g_c represents the number of generations which are needed for the genetic algorithm to converge.

$$g_c = \frac{\ln[P(g_c,t_{h1},M_1)\cdot(1-P(0,t_{h1},M_1))/P(0,t_{h1},M_1)\cdot(1-P(g_c,t_{h1},M_1))]}{\ln(r)} \tag{27}$$

worst case analysis

For the worst case analysis, $P(0,t_{h1},M_1)$ is given as $1/PS$ where PS is the population size, and $P(g_c,t_{h1},M_1)$ is given as 1- γ where γ is a tolerance parameter. Here, γ is set to $1/PS$. Using these two assumptions, g_c for the worst case is obtained as

$$g_c = \frac{\ln[(PS-1)^2]}{\ln r} \tag{28}$$

Therefore, the time complexity for the convergence of GATS using REIS in the worst case is $O(n^2 \cdot PS \cdot \ln PS)$, and is the same for both the nonrestricted and the restricted string space cases. $O(n^2)$ is the time to calculate the cost or fitness of a solution (a string) and its execution is needed PS times in a generation.

average case analysis

For the average case analysis, $P(0,t_{h1},M_1)$ is given as $1/(n_h m)$ for the nonrestricted string space case, and as $1/(n_h m_\Theta)$ for the restricted string space case where n_h is defined as

$$n_h = (\sum_{h=1}^{H} / \, partition(h) /) / \mathrm{H} \tag{29}$$

$|partition(h)|$ represents the number of tasks in a partition h and H is the number of partitions in a task graph.

$P(g_c,t_{h1},M_1)$ is given by $1\text{-}1/PS$ for both cases. Using these assumptions, g_c for the average case is obtained as follows:

$$g_c = \frac{\ln[(PS-1)(n_h m-1)]}{\ln r} = \frac{\ln(PS \cdot m \cdot n_h)}{\ln r} \quad \text{(nonrestricted)} \tag{30}$$

$$g_c = \frac{\ln[(PS-1)(n_h m_\Theta-1)]}{\ln r} = \frac{\ln(PS \cdot m_\Theta \cdot n_h)}{\ln r} \quad \text{(restricted)} \tag{31}$$

Therefore, the time complexity for the convergence of GATS using REIS on the average is $O(n^2 \cdot PS \cdot \ln(PS \cdot m \cdot n_h))$ for the nonrestricted string space case, and $O(n^2 \cdot PS \cdot \ln(PS \cdot m_\Theta \cdot n_h))$ for the restricted string space case. Because $m_\Theta < m$, a genetic search in the restricted string space tends to converge faster than that in the nonrestricted string space. However, there is no difference in the worst case analysis. Further, as can be seen in Table I which lists the time complexities for all cases, REIA requires less generations than REIS to converge, because it has a smaller string space than REIS. However, REIA requires more computation overhead to calculate the schedule length of the string than REIS.

		REIA	REIS
Nonrestricted	Worst	$O(n^3 \cdot PS \cdot \ln PS)$	$O(n^2 \cdot PS \cdot \ln PS)$
	Average	$O(n^3 \cdot PS \cdot \ln(PS \cdot m))$	$O(n^2 \cdot PS \cdot \ln(PS \cdot m \cdot n_h))$
Restricted	Worst	$O(n^3 \cdot PS \cdot \ln PS)$	$O(n^2 \cdot PS \cdot \ln PS)$
	Average	$O(n^3 \cdot PS \cdot \ln(PS \cdot m_\Theta))$	$O(n^2 \cdot PS \cdot \ln(PS \cdot m_\Theta \cdot n_h))$

Table I. Time complexities of GATS

6. Conclusion

In this paper, we proposed two representation methods (REIA, REIS) for encoding the scheduling information that is used in Genetic Algorithm based Task Scheduling (GATS) for scheduling the parallel programs with diverse embedded parallelism types in heterogeneous network systems. Each representation method affects the time complexity of GATS to converge the final solution. REIA string takes less generation to converge the final solution, but REIS string takes less time to calculate the schedule length of each string, which leads to less overall time complexity than REIA.

When allocating tasks of a parallel program into a heterogeneous network systems, the problem specific knowledge can be incorporated to expedite the convergence of GATS. That is to restrict the string space such that the task can only be allocated to the well matching machine in terms of the parallelism type. In this case, the allocation information in which case a task is allocated to badly matching machine is excluded. In heterogeneous network systems, each task should be allocated to an individual machine such that the parallelism types of the task and the machine are matched as close as possible to obtain more speedup.

7. References

[1] R.F. Freund and D.S. Conwell, Superconcurrency: A form of distributed heterogeneous supercomputing" *Supercomputing Review*, Vol. 3, No. 10, Oct. 1990, pp. 47-50

[2] A. Khokhar, V.K. Prasana, M.E. Shabaan and C. Wang, Heterogeneous Supercomputing: Problems and Issues" *Proc. Workshop on Heterogeneous Processing*, Mar. 1992, pp. 3-12

[3] R.F. Freund, "Optimal Selection Theory for Superconcurrency" *Supercomputing*, Nov. 1989, pp 699-703

[4] M. Wang, S. Kim, M. Nichols, R.F. Freund and H.J. Siegel, Augmenting the optimal selection theory for superconcurrency" *Proc. Workshop on Heterogeneous Processing*, Mar. 1992, pp. 13-21

[5] S. Chen, M.M. Eshaghian, A. Khokhar and M.E. Shabaan, "Selection Theory and Methodology for Heterogeneous Supercomputing" *Proc. Workshop on HeterogeneousProcessing*, March 1993, pp. 15-22

[6] J.H. Holland, "Adaptation in Natural and Artificial Systems", *Univ. of Michigan Press*, 1975

[7] J.J. Grefenstette, "Incorporating Problem Specific Knowledge into Genetic Algorithms", *Genetic Algorithms and Simmulated Annealing*, edited by L. Davis, Morgan Kaufmann, 1987, pp. 42-60

[8] G. Robertson, "Parallel Implementation of Genetic Algorithms in a Classifier System", *Genetic Algorithms and Simulated Annealing*, edited by L. Davis, Morgan Kaufmann, 1987, pp. 129-140

[9] T.L. Adam, K.M. Chandy and J.R.Dickinson," Comparison of List Schedules for Parallel Processing Systems" *Communications of the ACM*, Dec. 1974, pp. 685-690

[10]N. Adar and H. Barada, "Comparing the Efficiency of Various Genetic Algorithms for Task Scheduling", *Int. Conf. on Parallel and Distributed Computing Systems*, 1993, pp. 210-215

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

[12]Hwa Sung Kim, "Genetic Algorithms for Task Scheduling in Distributed Heterogeneous Supercomputing Systems", ICCIMA '98, Monash University, Australia

[13]R.R. Muntz and E.G. Coffman, "Optimal Preemptive Scheduling on Two-Processor System", *IEEE Trans. on Computers*, Vol. 18, No. 11, Nov. 1969, pp 1014-1020

[14]M.A. Iverson, Fusun Ozguner and G. J. Follen, " Parallelizing Existing Applications in a Distributed Heterogeneous Environment", *Proc. Workshop on Heterogeneous Processing*, April 1995, pp. 93-10

[15]K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGRAW-HILL Book Co., 1985

[16]H.S. Kim and H. Barada, "Superlinear Speedup in Distributed Heterogeneous Supercomputing Systems", Proc. 7'th KSEA NRC '96, Rutgers University, New Brunswick, New Jersey, March 1996, pp.139-145

[17]C.A. Ankenbrandt, "An Extension to the Theory of Convergence and a Proof of the Time Complexity of Genetic Algorithms", *Foundations of Genetic Algorithms*, edited by G.J.E. Rawlins, Morgan Kaufmann, San Mateo, California, 1991, pp. 53-68

[18]] D.E Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning, Massachusetts: Addison-Wesley Pub. Co., 1989