Comparison of Genetic Representation Schemes for Scheduling Soft Real-Time Parallel Applications

Yoginder S. Dandass
Mississippi State University
Box 9637
Mississippi State, MS 39762, USA
+1(662)325-7502
yogi@cse.msstate.edu

Amit C. Bugde
Mississippi State University
Box 9637
Mississippi State, MS 39762, USA
+1(662)325-8274
acb157@msstate.edu

ABSTRACT

This paper presents a hybrid technique that combines List Scheduling (LS) with Genetic Algorithms (GA) for constructing non-preemptive schedules for soft real-time parallel applications represented as directed acyclic graphs (DAGs). The execution time requirements of the applications' tasks are assumed to be stochastic and are represented as probability distribution functions. The performance in terms of schedule lengths for three different genetic representation schemes are evaluated and compared for a number of different DAGs.

The approaches presented here produce shorter schedules than HLFET, a popular LS approach for all of the sample problems. Of the three genetic representation schemes investigated, PosCT, the technique that allows the GA to learn which tasks to delay in order to allow other tasks to complete produced the shortest schedules for a majority of the sample DAGs.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search---Scheduling; C.3 [Special-purpose and application-based systems]: Real-time and embedded systems.

General Terms: Algorithms, Performance, Design.

Keywords: Soft real-time scheduling, genetic algorithms, genetic list scheduling.

1. INTRODUCTION

Advanced architecture processors provide features such as caches and branch prediction that result in improved, but variable, execution time of software. Hard real-time systems require tasks to complete within timing constraints. Consequently, hard real-time systems are typically designed conservatively through the use of tasks' worst-case execution times (WCET) in order to compute deterministic schedules that guarantee the tasks' execution within given time constraints. This use of pessimistic execution time assumptions provides real-time guarantees at the cost of decreased application performance and resource utilization.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

Soft real-time systems, conversely, can tolerate applications missing occasional deadlines. This affords considerable flexibility in scheduling policies and allows the need for meeting time constraints to be balanced with the need for improved performance. Such systems can improve resource utilization and performance by making scheduling decisions based on the premise that, in a given interval of time, it is unlikely that all successively activated tasks in an application will require their full WCET to complete. However, reserving less than the absolute maximum required time means that the deadlines will be missed occasionally when successive tasks require more time than was collectively reserved for them in the schedule.

There are a variety of systems in practice that tolerate occasional deadline misses. For example, in multimedia systems, video frames are decoded and displayed at a fixed rate. If the system misses a frame-decoding deadline, then either a partial frame is displayed or the frame is skipped entirely. Therefore, viewers will tolerate the slight degradation in video quality resulting from an occasional deadline miss. In another example, the over-sampling of environmental parameters allows computer-driven automatic control systems to function correctly as long as several back-to-back deadlines are not missed.

This paper focuses on constructing schedules for parallel soft real-time applications where the ability to tradeoff the probability of missing deadlines for reduced schedule lengths is required. The authors extend and improve on the previous work of a number of researchers in using genetic list scheduling (GLS) algorithms for scheduling real-time and non-real-time parallel applications that are represented as directed acyclic graphs (DAGs). While the schedules produced by the previous GLS-based techniques were of higher quality than those produced using non GLS approaches, further analysis of those schedules revealed several opportunities for further optimization. The techniques reported in this paper improve on the GLS algorithm from the previous research resulting in further reductions in schedule lengths.

The remainder of this paper is organized as follows: Section 2 provides a detailed problem definition, describes the parallel processing environment, and identifies the assumptions made regarding the communication infrastructure. Section 3 provides a brief overview of existing List Scheduling (LS) and GLS techniques. Section 4 describes the GLS approach used in this research. Section 5 presents experimental results and analysis. Key contributions and avenues for further research are highlighted in Section 6.

2. PROBLEM STATEMENT

The problem under study is as follows: *given a directed acyclic graph (DAG) representing a parallel soft real-time application, devise a scheduling algorithm that minimizes schedule lengths while simultaneously enabling the predictable tradeoff of quality-of-service for improved resource utilization*. A DAG $G = \{V, E\}$ consists of a set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices and a set $E = \{e_1, e_2, \ldots, e_k\}$ of k directed edges connecting the vertices. The vertices represent computational tasks in a parallel application and the edges represent communication and precedence relations between the tasks. The ordered pair $e_i = (v_{src}, v_{dest})$ indicates that the direction of edge e_i is from vertex v_{src} to v_{dest}.

Figure 1 depicts a hypothetical DAG. Edges e_1, e_2, e_3, and e_4 are designated as (v_0, v_3), (v_1, v_3), (v_2, v_3), and (v_3, v_4), respectively. Task execution time probabilities are depicted at the right of each vertex and edge. For example, when vertex v_0 is executed, it can take 4, 5, or 6 time units to complete with equal probability. The weight of vertices and edges, determined through analytical or empirical analysis, are assumed to be *independent random variables* and are represented as discrete probability distribution functions. These probability distribution functions (PDFs) can essentially be viewed as histograms representing the relative frequency with which different execution times are observed.

Independence between PDFs implies that the variations in vertex and edge weights are unrelated (*i.e.*, the variations in execution times are caused by architectural features such as caches and speculative execution and not because of variations in the size of the data being processed). Therefore, the problems being addressed here are primarily in the domain of real-time signal and image processing applications where successive "frames" of data gathered by sensors are processed repeatedly by a dedicated system.

It is assumed, without loss of generality, that the vertex and edge weights are specified as integer values (*i.e.*, by selecting an appropriate scale for the time units). Furthermore, it is assumed that the weight probabilities are non-zero only over a finite range of weight values. This is a valid assumption because real-time systems are designed to have as little variance in execution time as possible. These assumptions imply that vertices and edges have weight values only within a well-defined range of integers.

The scheduling technique developed in this research prevents the real-time tasks from being preempted at arbitrary instances of time. Instead, tasks are preempted only at vertex or edge boundaries. Therefore, in applications where rapid task switching is required, large tasks must be partitioned into strings of vertices or edges. For example, a loop in a task can be unrolled such that individual loop iterations are represented by distinct vertices. This restriction on preemption enables a more accurate determination of individual tasks' execution times because the disruptive effects of interrupt handling and task switching on instruction pipelines and caches are isolated to task boundaries.

A *homogeneous* parallel machine is assumed for executing the parallel application. The use of identical processors implies that the time taken to execute a computational task is the same on any processor. Also, a uniform point-to-point network capacity is assumed over the entire parallel system. This implies that the time needed to complete a particular communication operation is the same over any combination of distinct source and destination processors.

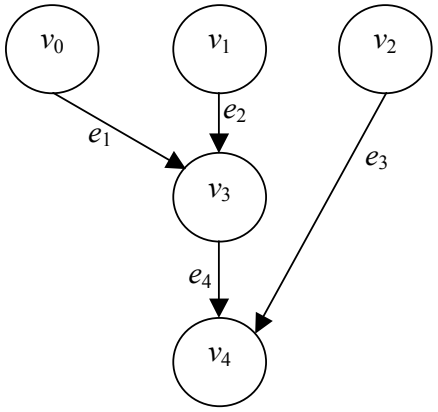

Figure 1. A hypothetical DAG (adapted from [7])

Table I. PDFs for the DAG in Fig 1 (adapted from [7])

Task				
v_0	**Weight:**	**4**	**5**	**6**
	Probability:	⅓	⅓	⅓
v_1	**Weight:**	**7**	**8**	**9**
	Probability:	⅓	⅓	⅓
v_2	**Weight:**	**1**	**2**	**3**
	Probability:	⅓	⅓	⅓
v_3	**Weight:**	**1**	**2**	
	Probability:	½	½	
v_4	**Weight:**	**2**	**3**	**4**
	Probability:	⅓	⅓	⅓
(v_0, v_3)	**Weight:**	**1**	**2**	**3**
	Probability:	⅓	⅓	⅓
(v_1, v_3)	**Weight:**	**7**	**8**	**9**
	Probability:	⅓	⅓	⅓
(v_2, v_4)	**Weight:**	**1**	**2**	**3**
	Probability:	½	9/20	1/20
(v_3, v_4)	**Weight:**	**7**	**8**	**9**
	Probability:	⅓	⅓	⅓

The time required to complete the execution of vertex v is a function of v's weight and the processor, p, on which it is to be executed. In homogeneous systems, the choice of processor to execute a task is assumed to have no impact on execution time. Therefore, the vertex weight directly represents execution time. The time required to complete a communication operation between two vertices depends on the weight of the corresponding edge, $w(e)$, and the processors on which the source and destination vertices are scheduled. In particular, the cost of communication operations between vertices scheduled on the same processor is assumed to be negligible. Therefore, the time required to complete the communication operation corresponding to edge e is computed as follows:

$$t(e, p_s, p_d) = \begin{cases} 0 \text{ when } p_s = p_d \\ w(e) \text{ otherwise} \end{cases}, \quad (1)$$

where p_s and p_d are the source and destination processors, for e, respectively.

Each processor is assumed to have a pair of simplex interfaces to the homogeneous virtual point-to-point network (one for incoming and one for outgoing data). Therefore, while full duplex operations are permitted, only a single incoming operation

and a single outgoing communication operation can occur at each processor-network interface at a time. However, the switched network fabric is assumed to be contention free (*i.e.*, there is sufficient network capacity available or packets can be sent along alternate routes in order to prevent congestion).

Non-preemptively scheduling DAGs with the goal of minimizing schedule lengths is known to be NP-hard in general [5]. Therefore, a number of heuristic approaches have been proposed that can produce near-optimal schedules in polynomial time [3, 11, and 16]. This paper presents a GLS-based offline scheduling approach for parallel soft real-time applications with precedence constraints. Following are the two main objectives of the scheduling algorithm investigated in this research:

1. Assign tasks in the DAG onto a parallel machine so as to minimize schedule length while utilizing as few processors as possible. Of two schedules with identical lengths, the schedule requiring fewer processors is preferred.
2. Compute the completion time PDF of the application. This PDF provides a means for precisely determining how to trade off the probability of meeting deadlines for reduced application execution time.

3. RELATED WORK

Ahmad and Kwok [16], Grajcar [13], and Dandass [6, 7] have shown GLS to be an effective technique for constructing non-preemptive schedules for DAGs with fixed execution times. Fundamentally, GLS is a combination of LS and a Genetic Algorithm (GA). LS is an iterative algorithm for schedule construction. During each iteration, a list of *ready vertices* is constructed and prioritized. A ready vertex is an unscheduled vertex whose precedence constraints have been met (*i.e.*, a ready vertex either has no preceding vertices in the DAG or all of its predecessor vertices have been previously scheduled and have completed execution). Prioritization essentially consists of ordering the ready vertices according to some criteria. The highest priority ready vertex is then scheduled on the processor that allows the earliest execution of the selected vertex. Note that scheduling a vertex, v, implies that all edges incident on v must also be scheduled and have to complete before v can begin execution.

The ready vertex prioritization scheme is the heuristic aspect of LS and a number of different prioritization criteria for LS have been proposed in the literature.

The *Dynamic Critical Path* (DCP) heuristic [15] computes the *critical path* (CP) of the partial schedule and selects the ready vertex with the least mobility for scheduling first. The CP of a DAG is the longest path in the DAG from an entry vertex to a terminal vertex. DCP outperforms the other LS heuristics for a majority of the DAGs tested. However, because of its complexity, DCP also requires long execution times. The *Mobility Directed* (MD) heuristic [19] does not fix the starting times of scheduled vertices until all nodes have been scheduled. In MD, vertices are scheduled in increasing order of their *relative mobility*. The relative mobility attribute of a vertex provides an indication of the amount by which the vertex can be delayed and still meet precedence constraints.

Highest Level First with Estimated Times (HLFET) [2] is a simple (and consequently fast) LS heuristic in which ready vertices are scheduled according to non-decreasing order of the longest path between the ready vertex and a *terminal* vertex in the DAG. A terminal vertex is one that has no outgoing edges. The *Earliest Time First* (ETF) LS [14] uses a greedy heuristic that prioritizes vertices according to their earliest start times. The algorithm exhaustively examines all ready vertex and processor pairs and schedules the ready vertex that can start the earliest on the processor that allows the earliest start time.

In GLS, a GA is used for prioritizing the vertices and edges. Kwok and Ahmad's GLS algorithm produces better schedules than the DCP algorithm for several test case DAGs and takes less time than the DCP algorithm [16]. Grajcar's GLS algorithm [13] produces schedules for a heterogeneous machine. Grajcar also points out certain DAG structures for which LS and GLS algorithms cannot find optimal schedules. Dandass [6] has reported the results of applying GLS towards scheduling DAGs with multicast communication in resource limited point-to-point network environments. Dandass has also applied GLS towards constructing soft real-time schedules using DAGs [7].

A variety of techniques relying on preemption have been developed for scheduling hard and soft real-time systems with fixed execution time requirements. A survey of these techniques can be found in [17]. Recently, research in scheduling tasks with stochastic execution times has gained attention. Proposed techniques include *Statistical Rate Monotonic Scheduling* (SRMS) [4], *Probabilistic Time-Demand Analysis* (PTDA) [18], *Stochastic Time Demand Analysis* (STDA) [10] and *Bandwidth Reservation* [1]. These techniques typically assume that tasks can be preempted and that the cost of switching tasks is negligible. Dogan and Ozguner [9] have developed techniques for stochastic scheduling of tasks in heterogeneous distributed computing systems. Their simulation studies showed that using stochastic scheduling provided better results in heterogeneous environments as compared to deterministic scheduling.

However, the stochastic scheduling algorithms described above cannot be used for scheduling DAGs in parallel environments because these algorithms are either intended for fully preemptive systems, or for uniprocessor environments, or do not account for interprocess communication and precedence constraints.

4. APPROACH

Figure 2 provides an overview of the GLS algorithm used in this paper. This is a *steady-state* algorithm in that new chromosomes derived from genetic operators immediately replace members of the current population.

4.1 Genetic Representation

Most existing LS and GLS algorithms focus on prioritizing vertices in the ready list and can schedule the incoming edges of a vertex in arbitrary order because communication contention is ignored. However, when communication contention is allowed, the order in which edges are scheduled also impacts schedule length. Therefore, the genetic representation in this research is used for prioritizing the DAG's vertices and, equally importantly, its edges. For this paper, two new priority-encoding schemes, PosCT, and PriNT were implemented and compared to the PosNT scheme previously used by Dandass [6, 7]. The three schemes are described in detail below.

4.1.1 Positional with No Thresholds

In the *Positional with No Thresholds* (PosNT) encoding scheme, each chromosome in the GLS has two vectors of genes. The vertex gene vector contains a gene for each vertex in the DAG and the edge gene vector contains a gene for each edge in the DAG (*i.e.*, there are $|V| + |E|$ genes in each chromosome). Each gene is a 32-bit value identifying the corresponding task (vertex or edge) in the DAG. The position of the vertex and edge genes in their respective vectors determines the priority of the corresponding vertices and edges used by the list scheduler. For example, consider two ready vertices v_x and v_y appearing at indices i_x and i_y, respectively in the vertex gene vector. If $i_x < i_y$, the pointer for v_x appears before the pointer for v_y in the vertex gene vector, and therefore, v_x is given higher priority. Edge priorities are similarly determined by ordering of edge genes in the edge gene vector. The PosNT-based GA searches for an optimal ordering of vertices and edges in the chromosomes.

4.1.2 Positional with Customized Thresholds

In the *Positional with Customized Thresholds* (PosCT) encoding scheme, each chromosome has three vectors of genes and there are a total of 2 * ($|V| + |E|$) genes in a chromosome. The vertex gene vector and the edge gene vector are identical in structure and function as in the PosNT representation described previously. In addition to the positional genes, PosCT also contains genes in an *overlap threshold gene vector*. This vector contains one threshold gene for every vertex and edge in the DAG. The threshold gene specifies the overlap threshold value for the corresponding task represented as an 8-bit unsigned integer. It is a fractional value in the interval [0, 1] computed by dividing the gene value by 255. The overlap threshold is used to determine if a task (vertex or edge), T_c, to be scheduled on processor P_s such that another task, T_s, already in the schedule for P_s, is delayed in order to allow task T_c to execute first (see section 4.6 for additional details).

Unlike the positional genes, the threshold genes occur at fixed locations in the gene vector (*i.e.*, the threshold for vertex v_x is located at position x in the overlap threshold gene vector and the overlap threshold for edge e_y is located at position $|V|+y$). In addition to searching for optimal vertex and edge positions, the PosCT-based GA also searches for optimal threshold assignments for tasks.

4.1.3 Priority with No Thresholds

In the *Priority with Fixed Thresholds* (PriNT) encoding scheme, each chromosome in the GLS has one vector of genes that directly represent the relative priorities of the vertices and edges. There are $|V| + |E|$ genes in each chromosome. Each priority gene is a 32-bit integer that encodes the priority of the corresponding edge or vertex. Unlike the positional genes in PosNT, priority genes for specific vertices and edges occur at fixed locations in the gene vector (*i.e.*, the priority gene for vertex v_x is located at position x in the priority gene vector and the priority gene for edge e_y is located at position $|V|+y$). The PriNT-based GA searches for an optimal prioritization of vertices and edges in the chromosomes.

4.2 The Selection Operator

In order to reduce the occurrence of premature convergence, the ability of high quality chromosomes to dominate the population is reduced by using the following fitness function proposed by Grajcar [13]:

$$\varphi(c) = \frac{rank(c)}{|offspring(c)| + 1}, \quad (2)$$

Where, the rank of chromosome c is the number of chromosomes in population Ω that produce poorer schedules than c. The chromosome with the largest fitness value in a random subset from Ω is selected for reproduction. Similarly, the chromosome with the least fitness value from another random subset from Ω is selected for replacement. In the experiments reported here, selection subset sizes ranging from 2% to 10% of $|\Omega|$ worked well for populations ranging from 200 to 1000 chromosomes.

```
Generate the initial population of chromosomes;
Use LS to construct schedules for each chromosome in order to
    assign fitness values;
While the termination criteria are not satisfied, loop
{
    Randomly select a genetic operator (crossover or mutation);
    Select chromosome(s) from the local population and apply
        the operator to produce the offspring chromosome;
    Use LS to construct a schedule in order to evaluate the
        offspring chromosome;
    Select chromosome from the local population to be replaced
        by the offspring chromosome;
}
Use the fittest chromosome to construct the solution schedule;
```

Figure 2. Overview of the GLS Algorithm

4.3 Recombination Operators

Three different crossover operators, *standard crossover* (SX), *ordered crossover* (OX) [8] and *vector crossover* (VX) are used in this GLS algorithm. SX is used for recombining the genes in the priority vectors in PriNT and the overlap threshold vectors in the PosCT representation schemes. In SX, the chromosome is treated as a sequence of bit positions and a *crossover* bit position in the vector is randomly selected. The offspring chromosome's genes are composed from a copy of the first parent's gene sequence prior to the crossover point appended to the copy of the second parent's gene sequence beginning at the crossover point.

OX is used for recombining the genes in the positional vertex and edge vectors in the PosNT and PosCT representation schemes. In OX, a single crossover point is randomly selected in the vertex vector. The sequence of genes prior to the crossover point is copied from the first parent to the front of the vertex vector in the offspring chromosome. The remaining genes in the first parent (*i.e.*, following the crossover point) are copied into the offspring gene in the order they appear in the second parent. The same operation is also performed on the edge and threshold vectors to construct the offspring chromosome.

In VX, the first parent contributes a complete copy of its vertex vector and the second parent contributes a complete copy of its edge vector to construct the offspring chromosome.

A mutation operator is also used in this research. The mutation operator swaps the location of a pair of genes within the vertex position, edge position, priority, and threshold vectors (where applicable).

4.4 Fitness and Schedule Construction

Fitness of a chromosome is a function of the length of the schedule produced by the chromosome. The authors of this paper have adapted the schedule construction techniques developed by Dandass [6, 7] for use with the new PosCT and PriNT representation schemes. Because of space restrictions, a detailed description of the schedule construction algorithm is not provided here. Essentially, the algorithm looks for idle timeslots in the current partial schedule in which the highest priority ready vertex, v_r, and its incident edges can be scheduled. The algorithm uses an iterative procedure, temporarily scheduling v_r on the first processor in the parallel machine, M, recording the completion time of v_r, and then reversing this temporary scheduling operation before scheduling v_r on the next processor in M. After attempting the scheduling operation on each processor in M, the algorithm greedily selects the processor that allowed the earliest completion of v_r and permanently schedules v_r on that processor. Before v_r is scheduled, its incident edges must be scheduled. Scheduling an edge requires the algorithm to find overlapping time slots during which the sending processor's outgoing network link and the receiving processor's incoming link are simultaneously idle.

When tasks have fixed execution times, the starting and completion times of scheduled tasks are specified as fixed values. However, the starting and completion times of tasks with stochastic execution time requirements need to be specified as PDFs. Figure 3 depicts a schedule (in Gantt-chart form) for the DAG in Figure 1. In this figure, the shaded rectangular regions indicate the times when the vertices and edges may potentially be executing. For example, v_0 begins executing on processor p_0 at time instance 1, and completes at the end of time instances 4, 5, or 6 with a probability of 1/3 each. Similarly, edge (v_0, v_3) begins execution immediately after v_0 completes at time instances 5, 6, or 7 with a probability of 1/3 each. The edge completes execution at time instances 5, 6, 7, 8, or 9 with probabilities 1/9, 2/9, 3/9, 2/9, and 1/9, respectively.

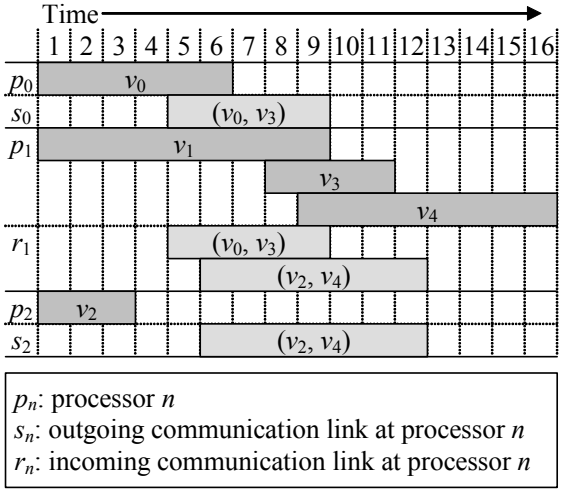

Figure 3. Stochastic Schedule for the DAG in Figure 1

When scheduling vertices with fixed time requirements, the completion time of a vertex is computed by summing its start time and execution times. From fundamental probability theory, the completion time PDF of a vertex with stochastic execution time requirements is computed by the convolution of its start time and execution time PDFs. Convolution of discrete PDFs $s(x)$ and $w(x)$ is defined as follows:

$$f(X) = \sum_{t=l_s}^{u_s} s(t) \cdot w(X - t + 1) \,, \tag{3}$$

where $[l_s, u_s]$ and $[l_w, u_w]$ are the intervals over which $s(x)$ and $w(x)$ are non-zero, respectively, and $X \in [l_s+l_w-1, u_s+u_w-1]$.

Often, a ready vertex v_r is to be scheduled on a processor after a previously scheduled vertex v_p has completed and a previously scheduled incident edge e_r on v_r has also completed. For example, in Figure 3, v_3 can start executing only after vertex v_1 and edge (v_0, v_3) complete. Note that edge (v_1, v_3) has an effective weight of 0 because v_1 and v_3 are scheduled on the same processor, and therefore, does not factor in v_3's start time computation. In such situations, the start time PDF for the vertex is determined from the maximum of the completion PDFs of the two preceding tasks. The maximum of two independent PDFs π_1 and π_2 defined over intervals $[l_1, u_1]$ and $[l_2, u_2]$, respectively, is computed as follows:

$$\begin{aligned} &\forall\, x \in [max(l_1,l_2),\ max(u_1, u_2)], \\ &\pi_{\max(\pi 1,\, \pi 2)}(x) = \pi_1(x)\pi_2(x) + \pi_1(x)\Pi_2(x-1) + \Pi_1(x-1)\pi_2(x), \end{aligned} \tag{4}$$

where Π_1 and Π_2 are the cumulative distribution functions (CDFs) corresponding to the PDFs π_1 and π_2, respectively.

An edge is scheduled in a common time slot in the processor-to-network links at the source and destination processors that starts after the source vertex has completed. Therefore, the start time of the edge is computed from the maximum of the completion time PDFs of the source vertex, the previously scheduled edge (if any) in the source processor-to-network link, and the previously scheduled edge (if any) in the destination processor-to-network link. For example, suppose that edge e is to be scheduled after vertex v completes and that the source processor-to-network link has edge e_{src} scheduled to complete after e can begin executing. Similarly, assume that the destination processor-to-network link has edge e_{dest} scheduled to complete after e can begin executing. In this case, the starting PDF of e can be computed from the maximum of the completion time PDFs of v, e_{src}, and e_{dest}. The maximum of three independent PDFs π_1, π_2 and π_3 can be computed as follows:

$$\pi_{max(\pi 1,\, \pi 2,\, \pi 3)} = max[max(\pi_1, \pi_2), \pi_3]. \tag{5}$$

It is important to note that equations (3) and (4) only apply to independent PDFs. Therefore, situations with dependent PDFs must be handled separately. In the example above, if the previously scheduled edges in the source and destination processor-to-network links are the same edge (*i.e.*, $e_{src} = e_{dest}$), then the starting time PDF of e must be computed from the maximum of the completion time PDFs of v and e_{src} only; taking the maximum of v, e_{src}, and e_{dest} in this situation will be erroneous.

4.5 Simplifying Heuristic

A key feature contributing to the effectiveness of GAs is the ability to evaluate chromosomes quickly, thereby enabling the GA to rapidly explore large portions of the search space. However, the PDF manipulation operations described above can be computationally costly, especially for PDFs defined over large intervals. Furthermore, the convolution of PDFs π_1 and π_2 defined

over intervals $[l_1, u_1]$ and $[l_2, u_2]$, respectively, results in a PDF defined over interval $[l_1+l_2-1, u_1+u_2-1]$. The resulting PDF is nearly as wide as the sum of the widths of the original PDFs. Therefore, the starting and completion PDFs for tasks towards the end of the schedule are typically much wider than the tasks' weight PDFs. The problem with the high computational cost of PDF manipulations is further exacerbated because during each LS iteration, a vertex and its incident edges are temporarily scheduled on every processor in order to determine the best processor. This implies that in each LS iteration, a majority of PDF computations are discarded before the vertex and its associated edges are permanently scheduled.

In order to reduce the number of PDF manipulations required to evaluate a chromosome, the GLS used in this research employs a two-phase scheduling approach. During the first phase, the expected values of the vertex and edge weight PDFs are used as fixed value estimates. These expected values are used to construct a preliminary fixed execution time schedule from the chromosome using the standard LS procedure. This preliminary schedule specifies the processor assignment and the ordering of vertices and edges to be used in the second phase. In the second phase, the PDF operations are used to convert the preliminary schedule into a stochastic schedule. Because the scheduling decisions have already been made previously, only those PDF operations required for computing the stochastic starting and ending time values of the vertices and edges in the schedule are performed.

4.6 Thresholds and PosCT

Every processor and communication link has a list of *idle* time-slots in which vertices can be scheduled. A ready task is assigned to a sub-interval within an idle slot, resulting in the fragmentation of the idle slot into smaller idle slots. There are occasions when a ready task's, T_R, ready time is less than or equal to the idle slot's start time, however, the idle slot, S_I, is not sufficiently large in order to allow T_R to complete (*i.e.*, T_R can be assigned to begin within S_I but the previously scheduled task, T_S, that appears at the end of S_I is scheduled to begin before T_R will complete). In the PosNT approach (*i.e.*, the no threshold approach used by Dandass [7]), T_R is inserted into S_I only if T_R does not overlap T_S. If there is overlap then T_R is scheduled in another interval that occurs after S_I. However, inspection of the schedules produced by PosNT revealed several instances in which delaying T_S by a small amount of time would have reduced overall schedule lengths. This is because allocating T_R in a later time slot resulted in a significant delay of tasks dependent on the completion of T_R, as compared with the delay incurred by T_S and its dependent tasks if T_R was allowed to complete before T_S began.

Figure 4 depicts the schedule for the DAG in figure 1 in which edge (v_2, v_4) is allowed to execute before (v_0, v_3). This results in a schedule that is shorter than the schedule shown in figure 3 by one time unit (the schedule in figure 3 was constructed using the PosNT approach).

However, arbitrarily delaying tasks do not always produce shorted schedules and can also perturb the scheduling power of the GLS algorithm. Therefore, previously allocated tasks should only be delayed by relatively small amounts as determined by the GA. In order to compute when T_S should be delayed, T_R is tentatively assigned to begin in S_I and the completion time of T_R is computed. Next the amount of overlap, δ, between tasks T_S and T_R is computed as follows:

$$\delta(T_S, T_R) = F(T_R) - S(T_S), \quad (6)$$

where $F(T_R)$ is the completion time of T_R and $S(T_S)$ is the start time of T_S. If $\delta \leq 0$ the two tasks do not overlap and task T_R can be scheduled in S_I without further consideration. If an overlap exists the overlap ratio, ω, is computed as follows:

$$\omega(T_S, T_R) = \frac{\delta(T_S, T_R)}{t(T_S)}, \quad (7)$$

where $t(T_S)$ is the expected value of the weight of task T_S. (see section 4.5).

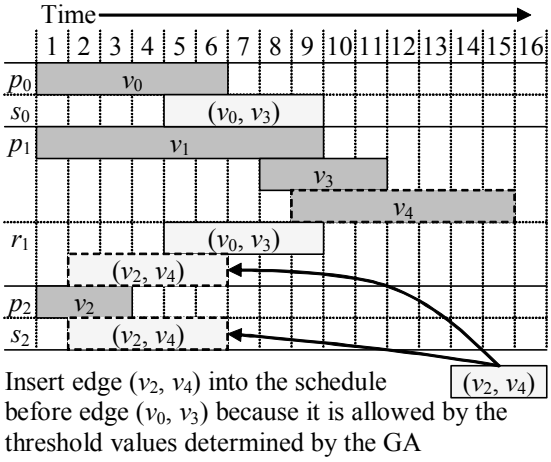

Figure 4: Shorter Schedule Produced by PosCT

If $\omega(T_S, T_R)$ is less than or equal to the PosCT overlap threshold gene value for T_S, then T_R is scheduled in S_I and T_S is delayed; if $\omega(T_S, T_R)$ is greater than the threshold value, then T_R is not scheduled in S_I and algorithm looks for the next available slot.

The delay in the start time of T_S also delays the start times of any previously scheduled tasks that depend on the completion of T_S. Therefore, a delay in the start time of T_S causes a "ripple" effect of delays in the partial schedule.

5. EXPERIMENTAL RESULTS

A number of DAGs were constructed in order to evaluate the three different GLS representation schemes. The structure of each DAG was one of *Simple Fork-Join* (SFJ), *Out Tree* (OUT), *Hierarchical Fork-Join* (HFJ), *Mean Value Analysis* (MVA), and *Random* (RND) as depicted in figure 5. The DAGs with RND structure have no predetermined branching pattern. All of the vertex and edge weights of the DAGs were also assigned PDFs derived from *beta*, *exponential*, and *randomized* distributions. The randomized PDFs have an irregular landscape of probability values as opposed to the relatively smooth curves of the beta and exponential distributions.

The DAGs were also given one of five different *computation-to-communication ratio* (CCR) values of 0.5, 0.6, 1.0, 1.5, and 2.0. CCR is the ratio of the average vertex weight and the average edge weight in the DAG. The various combinations of structure, probability distributions, and CCRs resulted in 225 different DAGs. Each DAG had approximately 500 vertices and edges, combined.

The results reported below were obtained by executing parallel island-model [12] implementations of the GLSs using 8 processes. Each process maintained an independent population of 1,000 chromosomes and computed 24,000 iterations. The processes exchanged the fittest chromosomes with each other at every 1,000th iteration beginning with the 12,000th iteration and at every 100th iteration after the 23,000th iteration.

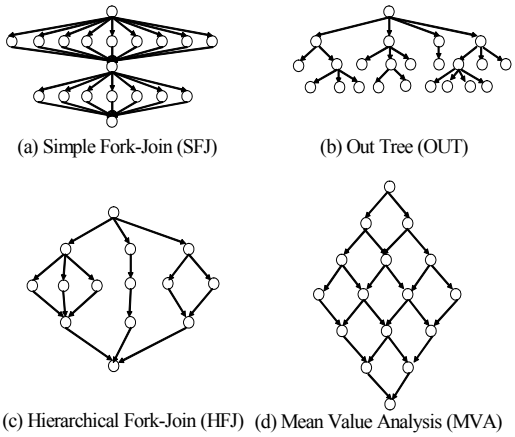

Figure 5: Example DAG Shapes (adapted from [7])

Table II shows the relative performance of the three representation schemes versus each other, broken out by DAG shapes. For example, the topmost row indicates that of the 45 HFJ DAGs:

- PosNT produced shorter schedules for 27 DAGs as compared with PriNT, and conversely, PriNT produced shorter schedules for 18 of the 45 DAGs.
- PosNT produced shorter schedules for 18 DAGs while PosCT produced shorter schedules for the remaining 27 DAGs.
- PriNT produced shorter schedules for 15 DAGs while PosCT produced shorter schedules for the remaining 30 DAGs.

The aggregate results over all DAGs summarized in the bottom row of the table clearly show that PosNT and PriNT have similar performance, whereas PosCT outperformed PosNT and PriNT by a ratio of 2:1. Also, all three representation techniques produced shorter schedules than the HLFET LS technique.

Table II. Pairwise Comparison of PosNT, PriNT, and PosCT

Shape	PosNT	PriNT	PosNT	PosCT	PriNT	PosCT
HFJ	27	18	18	27	15	30
MVA	23	22	17	28	18	27
RND	20	25	10	35	11	34
OUT	22	23	20	25	22	23
SFJ	21	24	10	35	6	39
All	113	112	75	150	72	153

Table III shows the collective performance of the three representation schemes and reinforces the superiority of the PosCT approach as compared with PosNT and PriNT. PosCT produced shorter schedules for 128 of the 255 DAGs (slightly over 50%) whereas the remaining 97 DAGs were split nearly evenly between PosNT and PriNT. These results imply that when restricted time and computational resources mandate the use of a single GLS implementation, then PosCT is a clear choice for the genetic representation scheme. However, when computational resources are unrestricted, then it is better to use GLS implementations with all three representation schemes and to pick the shortest resulting schedule.

Once a schedule for a DAG is constructed, the completion PDF, π_c for the *terminal* vertex (*i.e.*, the vertex with no outgoing edges) in the DAG (*e.g.*, v_4 in figure 1) also represents the PDF of the execution time for the entire DAG. Recall that π_c is essentially computed by summing the weight PDFs of many of the vertices and edges in the DAG. Therefore, according to the Central Limit theorem, for large DAGS, π_c has the shape of a Gaussian curve even if the individual weight PDFs that were summed to compute π_c are themselves not normal. Furthermore, π_c has non-zero probability values within the time interval $[l_c, u_c]$, where l_c and u_c represent the best-case and worst case execution time for the PDF (*e.g.*, [9, 15] in figure 4). Given π_c, the CDF, $\Pi_c(x)$ where $l_c \le x \le u_c$, provides the probability that the schedule will complete at or before time x. This can used to predictably tradeoff the probability of meeting completion deadline against the time for which resources are reserved for DAG execution.

Table III. Comparison of PosNT, PriNT, and PosCT

Shape	PosNT	PriNT	PosCT	Total
HFJ	16	7	22	45
MVA	10	16	19	45
RND	5	9	31	45
OUT	11	12	22	45
SFJ	9	2	34	45
All	51	46	128	225

6. CONCLUSIONS AND FUTURE WORK

This paper presents an effective genetic list scheduling technique for constructing non-preemptive schedules for soft real-time parallel applications. It is assumed that the applications are expressed in the form of fine-grained DAGs and that the variable weights of the computation and communication tasks in the DAGs are expressed in the form of probability distribution functions.

Three different genetic representation schemes were investigated. In order to study the efficacy of the genetic representation schemes, schedules were created for 225 different DAGs with a variety of structural characteristics. Of these, the PosCT scheme enabled the GA to determine when to delay the execution of certain previously scheduled tasks in order to allow other ready tasks to execute. This ability resulted in shorter schedules for a majority of DAGs as compared with the schedules produced using the other representation schemes.

Ongoing research is focused on improving and extending the PosCT-based GLS approach. Currently, PosCT overlap thresholds are compared with the ratio of overlap versus task weights. It may be more appropriate to compare overlap thresholds to the *probability* that a ready task to be inserted into a slot delays (because of overlap) a previously scheduled task. In this case, insertion of the ready task into the slot will be permitted only if this probability is less than the GA-determined threshold. This extension will not allow the use of the simplifying heuristic and the two phased approach. Therefore, a central idea to be explored in this future research is to determine whether the detailed start and completion time PDFs of the vertices and edges

in the partial schedules can be exploited to construct even better schedules than possible using the current two-phase approach. It is expected that larger-scale parallelism will also need to be explored in order to account for the significant increases in computation time required to evaluate chromosomes using the single-phase approach.

7. REFERENCES

[1] Abeni, L., and Buttazzo, G. QoS Guarantee Using Probabilistic Deadlines. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*. 1999, 242-249.

[2] Adam, T. L., Chandy, K. M., and Dickson, J. R. A Comparison of List Schedules for Parallel Processing Systems. *Communications of the ACM, Vol. 17*. 1974, 685-690.

[3] Ahmad, I., Kwok, Y., and Wu, M. Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks.* 1996, 207-213.

[4] Atlas, A., and Bestavros, A. Statistical Rate Monotonic Scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium.* 1998, 123-132.

[5] Coffman, E. G. *Computer and Job-Shop Scheduling Theory*, John Wiley and Sons, New York, 1976.

[6] Dandass, Y. S. A Genetic Algorithm for Scheduling Acyclic Digraph in the presence of Communication Contention. In *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications.* 2003, 223-230.

[7] Dandass, Y. S. Genetic List Scheduling for Soft Real-Time Parallel Applications. *IEEE Congress on Evolutionary Computation.* June 2004, 1164-1171.

[8] Davis, L. Applying Adaptive Algorithms to Epistatic Domains. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence.* 1985, 162-164.

[9] Dogan, A., and Ozguner, F. Stochastic Scheduling of a Meta-task in Heterogeneous Distributed Computing. *IEEE International Conference on Parallel Processing Workshops.* 2001, 369–374.

[10] Gardner, M. K. *Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems*, Ph.D. Thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1999.

[11] Gerasoulis, A., and Yang, T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors. *Journal of Parallel and Distributed Computing, Vol. 16*, *No. 4*. 1992, 276-291.

[12] Gordon, V. S., and Whitley, D. *Serial and Parallel Genetic Algorithms as Function Optimizers*. Technical Report CS-93-114, Colorado State University, 1993.

[13] Grajcar, M. Strengths and Weaknesses of Genetic List Scheduling for Heterogeneous Systems. In *Proceedings of the 2nd International Conference on Application of Concurrency to System Design, IACSD,* 2001.

[14] Hwang, J. J., Chow, Y. C., Anger, F. D., and Lee, C. Y. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM Journal of Computing, Vol. 18, No. 2*. 1989, 244–257.

[15] Kwok Y., and Ahmad, I. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 5.* 1996, 506-521.

[16] Kwok, Y., and Ahmad, I. Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors using a Parallel Genetic Algorithm. *Journal of Parallel and Distributed Computing*, *Vol. 47, No. 1.* 1997, 58-77.

[17] Liu, J. W. S. *Real-Time Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2000.

[18] Tia, T. S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L. C., and Liu, J. W. S. Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium.* May 1995, 164-173.

[19] Wu, M-Y., and Gajski, D. D. Hypertool: A Programming Aid for Message Passing Systems. *IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 3.* 1990, 330–343.