Probabilistic parameters of conditional task graphs

Sergiu Carpov*,†, Jacques Carlier†, Dritan Nace† and Renaud Sirdey*
* *CEA, LIST,*
Embedded Real Time Systems Laboratory,
Point Courrier 94, 91191 Gif-sur-Yvette Cedex, France.
† *UMR CNRS 6599 Heudiasyc,*
Université de Technologie de Compiègne,
Centre de recherches de Royallieu,
BP 20529, 60205 Compiègne Cedex, France.

***Abstract*—This paper deals with the problem of determination of probabilistic parameters for tasks in a series-parallel conditional task graph. Such problematic is encountered in the context of parallel computing when dealing with conditional precedence constrained parallel tasks on a multi-core machine. The conditional task graph was introduced in order to express conditional precedence constraints and thus to model conditional execution in an application, which is not possible with a conventional task graph. We focus here in the calculation of two probabilistic parameters: the heads (release dates) and the tails (delivery times). An algorithm for computing these parameters is proposed. Although it has a pseudo-polynomial time complexity, the execution time of the algorithm can be further reduced at the price of less precision in the results.**

***Keywords*-Parallel computing; conditional task graph; probabilistic release/delivery times; multiprocessor scheduling**

I. Introduction and motivation

A task graph model, also known as directed acyclic graph (DAG), is used to represent algorithms which have to be executed on a parallel computing system. The benefit of the task graph modeling is that it allows to explicitly express the parallelism present in an algorithm by means of dependencies (precedence relations) between the tasks. A multitude of methods have been proposed in the literature to deal with the DAG scheduling on multiprocessor systems having as objective the completion time minimization, refer to [1] for a review. The above methods use different parameters which are defined for the DAG's tasks. Two important parameters are the release date (head or top level) and the delivery time (tail or bottom level) of a task. These parameters are used in task selection rules of list scheduling algorithms. For example, in the work [2] is described the selection rule CP/MISF (critical path/most immediate successors first) which prioritizes the tasks having the largest delivery time.

Another important parameter for a task graph is the minimal execution time (or the critical path) which is the completion time obtained when no constraint is imposed on the number of available processors (i.e. when the number of processors is considered to be unlimited). The minimal execution time is a lower bound for the completion time of the general multiprocessor scheduling problem and it is used in tree search algorithms (e.g. branch and bound methods) to reduce the search space.

The task graph model lacks of expressivity which limits its use in many practical situations. A first drawback is the hypothesis that task execution times are constant. In reality, they are variable and depend on several factors (caching, branch prediction mechanisms). A second disadvantage is the absence of tools for modeling conditional branches which are widely employed in programs. A conditional branch is a special task, such that only one of its successors is executed (in function of a condition depending on the input data for example). The case of variable task durations is known in the literature on probabilistic PERT scheduling [3] and on stochastic DAG scheduling [4], [5], [6]. The literature on task graphs with conditional branches is scarce and mainly consists in methods for allocating and scheduling them onto multiprocessor systems [7], [8]. The goal of this study is to define probabilistic parameters for task graphs with conditional branches.

A conditional task graph (CTG) is a directed acyclic graph $G = (V, A, d, p)$, where V is the set of tasks, $E \subset V \times V$ is the set of dependence relations between the tasks, $d : V \to \mathbb{N}$ is a function that associates to each task $v \in V$ a non-negative duration d_v[1] and $p : E' \to \mathbb{R}$ is a function that associates to a subset of edges E', $E' \subset E$, a probability p_{vu}, $(v, u) \in E'$. For a task v either all its input (output) edges have probabilities, or no probability is defined for any input (output) edge, in the first case we call v a branch-in (branch-out) node and in the later a join (split) node. All the successors, or predecessors, of split, respectively join nodes, are executed. In contrast, when a task v is a branch-in (branch-out) node, only one of its predecessors (successors) u is executed with a probability p_{uv} (p_{vu}). Relation $\sum_{u \in \text{pred}(v)} p_{uv} = 1$ is satisfied for any branch-in node v and equivalently relation $\sum_{u \in \text{succ}(v)} p_{vu} = 1$ for any branch-out node v. We shall note that a task can be a join/split and a branch in/out node at the same time. Figure

[1]Without loss of generality we suppose that the execution times are integers.

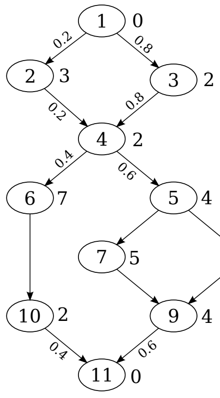

Figure 1: A conditional task graph. The values on the edges represent the probabilities of branch-in, branch-out nodes and the values next to vertices are the execution times.

1 illustrates a CTG build up from 11 vertices, with 2 branch-out (vertices 1 and 4) and 2 branch-in (vertices 4 and 11) nodes. The next node that will be executed after the branch-out node 1 is either the node 2 with a probability of 20% or the node 3 with a probability of 80%.

In this work we propose an algorithm for finding probabilistic release dates and delivery times in series-parallel CTGs. We shall note that the usual algorithm for finding release dates and delivery times in DAG cannot be applied, at least directly, because the statistical correlation between the random variables is not considered. On the other hand, we notice that restricting to the use of constant task execution times limits the scope of the model. Hence, our goal is to propose an an algorithm, which can take into consideration task execution times described by probability distributions. The probabilistic release dates and delivery times can be used in a list scheduling algorithm in order to compute priority rules that will better consider the CTG structure and execution specificities.

The paper is organized as follows: after a brief description of some basic operations on discrete random variables in Section II, the proposed algorithm for calculating release dates and delivery times is introduced in Section III, subsequently in Section IV the results of the algorithm applied on a sample graph are studied and Section V concludes the paper.

II. Discrete random variable operations

Let A be a discrete random variable (DRV) characterized by a finite set of possible values D_A, where each value $x \in D_A$ (also called *realization*) has a positive probability $\Pr(A = x) > 0$ defined. The probability mass function (pmf) of the DRV A is a bijection $f_A : D_A \to [0,1]$ such that $f_A(x) = \Pr(A = x)$ for any $x \in D_A$ and $f_A(x) = 0$ for any other value. We shall note that $\sum_{x \in D_A} f_A(x) = 1$. The cumulative distribution function (cdf) of A is a bijection $F_A : \mathbb{R} \to [0,1]$ such that $F_A(x) = \Pr(A \le x) = \sum_{t \le x} f_A(t)$ for any x. In what follows we consider that the domain of definition D_A contains only natural numbers (integers)[2], i.e. $D_A \subset \mathbb{N}$, and we note by n_A the size of the domain of definition, $n_A = |D_A|$. In the sequel we introduce some basic operations on DRVs that are used in the computations of heads and tails in CTG.

A. Maximum of independent DRVs

Let A and B be two independent DRVs with domains D_A and D_B, cumulative distribution functions respectively F_A and F_B. Let C be another DRV that denotes the maximum of A and B, i.e. $C = \max(A, B)$. The domain of definition of variable C is $D_C \subseteq (D_A \cup D_B) \setminus \{\min(D_A \cup D_B)\}$, respectively its size is $|D_C| \le n_A + n_B - 1$. Using simple mathematical manipulations we can compute the maximum of two independent DRV for a value x:

$$\Pr(\max(A,B) \le x) = \\ \Pr(A \le x \wedge B \le x) = F_A(x) \cdot F_B(x)$$

Thus, the cdf of variable C is given by $F_C(x) = F_A(x) \cdot F_B(x)$ for any x. The algorithmic complexity of computing the maximum of two DRVs is $O(n_A + n_B)$.

Because of the associativity of the maximum operator, the maximum of m independent DRVs, X_i for $i \in 1 \ldots m$, can be computed by applying the maximum operation $m-1$ times between two DRVs. The complexity of this operation is $O\left(\sum_i n_i\right)$ where n_i is the size of the domain of definition of X_i. The size of the domain of the obtained DRV is at most $\sum_i n_i - m + 1$.

B. Sum of independent DRVs

Let A and B be two independent DRVs with domains D_A and D_B, probability mass functions f_A and respectively f_B. Let C be the sum of these DRVs, $C = A + B$. The pmf of the sum of two DRVs is computed by convolving the pmf of these DRVs, which is a well known result from the probability theory. Thus the pmf of C is $f_C(x) = f_A * f_B(x) = \sum_t f_A(t-x) f_B(x)$, where $*$ is the convolution operator, for any $x \in D_C$. The domain of definition of DRV C is $D_C = \{x_a + x_b \mid x_a \in D_A, x_b \in D_B\}$ and its size satisfies $|D_C| \le n_A \cdot n_B$. Only the possible values belonging to the domains of definition are not zero, hence many useless calculations of the convolution sum can be avoided. In such a way the convolution sum computation has a complexity $O(n_A n_B)$.

The sum of three and more independent DRVs is calculated equivalently, the convolution operator being associative. Suppose we have m independent DRVs, X_i for

[2]No precision is lost due to rounding errors as CTGs have integer execution times.

$i \in 1 \dots m$, then the sum of these DRVs has a domain size of a most $\prod_{i=1}^{m} n_i$ where n_i is the domain size of X_i.

The sum between a DRV A and a constant b is equivalent to shifting the pmf of A by b units to the right. Suppose $C = A + b$ then the pmf of C, is $f_C(x + b) = f_A(x)$ for any $x \in D_A$.

C. Combining realizations of a DRV

We call a *composite realization* of a DRV a set of its possible values, that is a collection of values from the domain of definition of the DRV. Suppose that the probabilities of a composite realization A of a DRV C are given by a function $f_A : D_A \to [0, 1]$ where D_A is the set of values of the realization A and $f_A(x)$ is the probability of $x \in D_A$, $f_A(x) = \Pr(C = x \mid C = A)$. Relation $\sum_{x \in D_A} f_A(x) = 1$ is verified.

Let C be a DRV which has two mutually exclusive composite realizations A and B, described by the parameters D_A, f_A and D_B, f_B. Let p_A, $p_A = \Pr(C = A)$, denote the probability that DRV C takes all the values of the realization A and p_B, $p_B = \Pr(C = B)$, of the realization B. Relation $p_A + p_B = 1$ is valid because A and B are mutually exclusive. Knowing the parameters of A and B we want to compute the domain of definition D_C and the pmf f_C of C. The domain of definition is the union of the sets of values of the realizations A and B, $D_C = D_A \cup D_B$. For any $x \in D_C$ we have:

$$\begin{aligned}\Pr(C = x) = \Pr(C = x \wedge C = A) +& \\ \Pr(C = x \wedge C = B) =& \\ \Pr(C = x | C = A) \Pr(C = A) +& \\ \Pr(C = x | C = B) \Pr(C = B)&\end{aligned}$$

Rewriting the last expression we obtain $f_C(x) = p_A f_A(x) + p_B f_B(x)$ for any $x \in D_C$. This computation has a complexity $O(n_A + n_B)$ where n_A, n_B are the number of values in each realization. In what follows we shall denote this operation $C = p_A A \uplus p_B B$ and by abuse of language we shall call it weighted sum.

III. Computation of probabilistic heads and tails

In this section we introduce the notion of release times (heads) and delivery times (tails) for CTGs, and in sequel we introduce an algorithm to calculate the heads and tails for series-parallel CTGs. In what follows we make the hypothesis that the number of available processors is unlimited, thus we should not worry about the placement of tasks.

Following the usual definition, the release time r_v for a task v, $v \in V$, is defined as the minimal time the task v must wait before it can start, and respectively, the delivery time q_v is defined as the minimal time the task v must remain in the system after it has finished its execution. In case of usual task graphs the heads can be calculated by examining the graph nodes in topological order, the head of a task v is $r_v = \max_{u \in \mathrm{pred}(v)} (r_u + d_u)$ where $\mathrm{pred}(v)$ are the predecessors of v and d_u is the duration of task u. The tails are calculated either by examining the tasks in inverse topological order using a similar relation for the tails of tasks or by applying the method described above on the inversed graph[3].

For a CTG single valued heads and tails cannot be defined because the execution paths of a CTG are different in function of the chosen paths in branch nodes. The following definition introduces the probabilistic heads and tails for CTGs.

Definition 1 (Probabilistic heads and tails)**.** Given a CTG, a *probabilistic head* of a task is a DRV which denotes the minimal time that has to elapse before this task can start its execution. Equivalently, a *probabilistic tail* of a task is a DRV which denotes the minimal time this task has to remain in the system after is has finished its execution.

By considering the heads and the tails as DRVs we, firstly, introduce the possibility to express the uncertainty of the CTG's branch nodes, and secondly, generalize the usual definition of heads and tails for task graphs. Obviously the heads and tails for a task graph can be represented as single valued (probability equals to one) heads and tails.

It is not straightforward to compute the probabilistic heads and tails for CTGs. Let $G = (V, E, d, p)$ be a series-parallel CTG (the graph in Figure 1 is series-parallel). A graph is series-parallel if it can be recursively constructed either by parallel composition or by serial composition of two series-parallel graphs, for more information on recognizing series-parallel graphs refer to [9]. Without loss of generality we suppose that G contains a single source vertex and a single sink vertex, denoted respectively s and t. In this case G is called 2-terminal series-parallel graph. The head of a task v, $v \in V$, is a DRV denoted R_v, and respectively, the tail is a DRV denoted Q_v. Initially we focus on the calculation of heads and afterwards we briefly describe the computation of tails based on the symmetry between the heads and the tails.

To calculate the heads of tasks, graph vertices are examined in topological order and for each vertex v, $v \in V$, relation $R_v = \max_{u \in \mathrm{pred}(v)} (R_u + d_u)$ is used to find the head of v. For any predecessor u of vertex v, $u \in \mathrm{pred}(v)$, the computation of the inner sum $R_u + d_u$ does not pose any problem, it corresponds to a simple shift of the pmf of R_u by d_u units to the right. In contrast to the previous sum operation, computing the "max" of several DRVs is not straightforward because we do not know if the DRVs are correlated or not. Four different computation rules can be distinguished:

1) Vertex v has only one predecessor. In this case the

[3]An inversed graph is a graph in which the directions of the edges have been inversed

calculation of the head is straightforward as no "max" operation is employed, $R_v = R_u + d_u$ where $u = \text{pred}(v)$.

2) Vertex v is a join task and its predecessors' heads are independent DRVs. The head of vertex v is computed using the maximum of independent DRVs method described in the previous section, $R_v = \max_{u \in \text{pred}(v)} (R_u + d_u)$.
3) Vertex v is a branch-in task. A probability p_{uv} is defined for any input edge $(u, v) \in E$ of vertex v. The heads of the predecessors of v are composite realizations of a single DRV which is the head of task v. Each realization R_u has a probability to be executed p_{uv}. Using the weighted sum operation described in the previous section the head of task v is $R_v = \biguplus_{u \in \text{pred}(v)} p_{uv} \cdot (R_u + d_u)$.
4) Vertex v is a join task and its predecessors' heads are dependent DRVs. The dependence between these heads is created in an ancestor split vertex v', refer to Proposition 2 for a proof and an illustration. In this case, a head R_v^{rel} of vertex v "relative" to vertex v' is computed and afterwards the "real" head is equal to the sum $R_v = R_{v'} + R_v^{rel}$ between the DRVs R_v^{rel} and $R_{v'}$ (which corresponds to the convolution of the DRVs' pmfs).

Proposition 2. *Let $G = (V, A, d, p)$ be a two-terminal series-parallel CTG and let v be a join vertex for which its predecessors' heads are dependent DRVs. Then it exists a split vertex v', $v' \in \text{anc}(v)$, where this dependence is created and v' is the nearest common ancestor of v's predecessors.*

Proof: The only place in CTGs where a dependence is created are the split vertices. Inevitably, in a descendant join vertex the maximum of dependent DRVs has to be computed.

Let $v \in V$ be a join vertex. Without loss of generality we suppose that v has two predecessors u and w. Let us denote by L the common ancestors of u and w, $L = \text{anc}(u) \cap \text{anc}(w)$. For any CTG the set L is not empty and it contains at least the source vertex s, because s is a common ancestor to all graph vertices. Let $v' \in L$ be the nearest common ancestor of u and w, thus $|L \setminus \text{anc}(v')| = 1$. Consider the sub-graph G' of G composed of vertices $\text{anc}(v) \setminus L$. Because G is series-parallel the graph G' has two connected components, the sub-graphs SG_1 and SG_2, refer to Figure 2 for an illustration. In graph G no edge exists between the components SG_1 and SG_2, no in (out) edges exists for SG_1, SG_2 except (v', p) for any $p \in \text{succ}(v')$ ((p, v) for any $p \in \text{pred}(v)$). We conclude that if the heads R_u and R_w are dependent DRVs then the only place where this dependence can be created is vertex v'. ■

Algorithm 1 finds the heads of vertices in a CTG graph using the four computation rules described earlier. In the algorithm, an array R_v^{mem}, $v \in V$, is used to track the vertices where a dependence was created. Particularly, if R_v^{mem} is different from zero then at vertex v a dependence was created and the rule 4 must be used for heads calculation. The graph vertices are examined in topological order. The head of each vertex is computed in function of its type, e.g. for branch-in nodes the weighted sum of predecessors' heads is used and for join nodes the maximum of predecessors' heads (lines 4 and 6). If at the nearest common ancestor, v', of the current vertex v predecessors a dependency was created (i.e. $R_{v'}^{mem} \neq 0$) then all the heads from v' to v are "relative" and must be updated (lines 9-11). If the current vertex is a split and the domain of its head DRV has more than one possible value then a new "relative" calculation is started from the current vertex (lines 14-17).

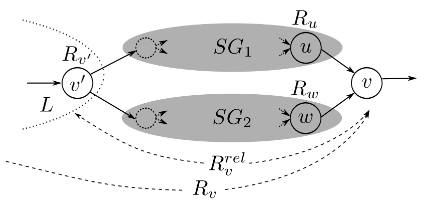

Figure 2: Illustration of the nearest common ancestor of a vertex (used in the proof of Proposition 2).

In order to compute the tails of vertices in a CTG the same algorithm applied to the inversed CTG is used. We shall note that the branch-out and branch-in vertices change place in the inversed CTG, that is branch-out nodes of a CTG correspond to branch-in nodes in the inversed CTG.

The computation of heads and tails implies the use of operations described in the previous section. The number of possible values of the resulting DRV potentially increases each time an operation is applied. In the case of the maximum operation this number increases linearly, whereas the sum operation increases it exponentially. For example, suppose we have two DRVs such that the first one has 3 possible values and the second one 5 possible values. Then, the maximum between the DRVs creates a DRV with at most 7 (3+5-1) possible values and the sum operation a DRV with 15 values (in the worst case). The complexity of Algorithm 1 is pseudo-polynomial because of the computations implying DRVs which tend to augment the domains of definition. We notice that the domain of values of DRV is included in $[0, CP]$, where CP gives the maximum completion time. In practical situations, a remedy will be to approximate the DRVs distributions. In this way, the computational complexity can be reduced at the price of less precise results.

IV. Example of algorithm execution

The algorithm for finding probabilistic heads and tails, described in the previous section, is applied on the CTG illustrated in Figure 1. Initially, R_v^{mem} and R_v are zero for

Algorithm 1 Algorithm for computing the probabilistic heads/tails of tasks in a CTG.

```
Input: A CTG $G = (V, E, d, p)$
Output: Heads $R_v$ for any vertex $v \in V$ as DRVs
1:  $R_v = R_v^{mem} = 0$ for any $v \in V$
2:  for $v \in V$ in topological order do
3:    if $v$ is a branch-in then {rule 3}
4:      $R_v = \biguplus_{u \in \mathrm{pred}(v)} p_{u,v} \cdot (R_u + d_u)$
5:    else {$v$ is a join, rules 1,2,4}
6:      $R_v = \max_{u \in \mathrm{pred}(v)} (R_u + d_u)$
7:      if $|\mathrm{pred}(v)| > 1$ then
8:        Find the nearest common ancestor, $v'$, of $v$'s predecessors
9:        if $R_{v'}^{mem} \neq 0$ then {rule 4}
10:         $R_u = R_u + R_{v'}^{mem}$ for any
                $u \in (\mathrm{anc}(v) \cup \{v\}) \setminus \mathrm{anc}(v')$
11:       end if
12:     end if
13:   end if
14:   if $v$ is a split and $|\mathrm{succ}(v)| > 1$ and $|D_{R_v}| > 1$ then
15:     $R_v^{mem} = R_v$
16:     $R_v = 0$
17:   end if
18: end for
```

any vertex. Suppose that the used topological order coincides with the vertex indexing (several topological orders are defined for this graph). At the first iteration, $v = 1$, no instruction from the loop body is executed. In the next two iterations the heads of vertices 2 and 3 are computed, $R_2 = R_3 = 0$. No other action is performed. At the 4-th iteration the head of the current vertex 4 is computed using the weighted sum operation (vertex 4 is a branch-in), $R_4 = \{(2, 0.8), (3, 0.2)\}$. The conditional instruction (line 14) is not executed (vertex 4 is a branch-out, not a split). In the next iteration, vertex 5 is examined. The head of vertex 5 is computed, $R_5 = \{(4, 0.8), (5, 0.2)\}$. As vertex 5 is a split with two successors and its head contains more than one possible value the second condition (line 14) is executed. The current head value R_5 is copied to R_5^{mem} and in the sequel R_5 is nullified (a new "relative" calculation starts). Afterwards, in the next three iterations the heads of vertices $6, 7, 8$ are calculated and no other action is performed: $R_6 = \{(4, 0.8), (5, 0.2)\}$ and $R_7 = R_8 = 4$ (we shall note that these heads are "relative" to vertex 5). In the following iteration, the head of vertex 8, which is a join, is computed $R_8 = 9$. In sequel, the nearest common ancestor, $v' = 5$, is found and because R_5^{mem} is not zero to the heads of vertices $5, 7, 8, 9$ is added R_5^{mem}: $R_5 = \{(4, 0.8), (5, 0.2)\}$, $R_7 = R_8 = \{(8, 0.8), (9, 0.2)\}$ and $R_9 = \{(13, 0.8), (14, 0.2)\}$. In the next iteration $R_{10} = \{(11, 0.8), (12, 0.2)\}$ is found and in the last iteration the head of 11 is computed using the weighted sum operation $R_{11} = \{(13, 0.32), (14, 0.08), (17, 0.48), (18, 0.12)\}$.

Task, v	R_v
1	0
2	0
3	0
4	(2,0.80) (3,0.20)
5	(4,0.80) (5,0.20)
6	(4,0.80) (5,0.20)
7	(8,0.80) (9,0.20)
8	(8,0.80) (9,0.20)
9	(13,0.80) (14,0.20)
10	(11,0.80) (12,0.20)
11	(13,0.32) (14,0.08) (17,0.48) (18,0.12)

(a) Heads

Task, v	Q_v
1	(13,0.32) (14,0.08) (17,0.48) (18,0.12)
2	(11,0.40) (15,0.60)
3	(11,0.40) (15,0.60)
4	(9,0.40) (13,0.60)
5	9
6	2
7	4
8	4
9	0
10	0
11	0

(b) Tails

Table I: The computed heads and tails for each task from the CTG in Figure 1. The distributions are presented as pairs of possible values and theirs probabilities. When a distribution has only one value the probability is not reported.

Table I presents the heads and the tails obtained by the algorithm for each task of CTG. The heads of tasks $1, \ldots, 3$ are distributions containing only one possible value (i.e. the heads are constant values). The same can be said for the tails of tasks $5, \ldots, 11$. In contrast to this, the head of node 4 has two possible values: 3 with probability 20% and 2 with probability 80%. The head of the sink vertex, task 11, has 4 possible values and corresponds to the probabilistic completion time of the CTG. The most probable execution time of the CTG is then 17 (having the highest probability - 48%), the mean execution time is 15.6, the minimal and the maximal execution times are respectively 13 and 18.

V. CONCLUSIONS

In this paper we have examined the determination of probabilistic parameters of conditional task graphs. A CTG is a generalization of the task graphs (DAGs) in which conditional branches are modeled. A conditional branch is a special task that executes one of its successors in function of a condition.

We have proposed an algorithm for finding probabilistic heads and tails for each vertex of a series-parallel CTG. The algorithm has a pseudo-polynomial complexity. The execution time of the algorithm depends on the sizes of the domains of definition of the DRVs used to represent the heads and the tails. At the price of less precise results the execution time of the algorithm can be reduced in order to be able to use it for graphs encountered in practice.

In a subsequent work we plan to use the probabilistic heads and tails for defining priority rules for list scheduling heuristics. Then, we envisage to use such a heuristic for scheduling CTGs on multiprocessor systems and to evaluate the benefits brought by this method.

References

[1] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, pp. 406–471, December 1999.

[2] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, vol. 33, no. 11, pp. 1023–1029, 1984.

[3] A. Nádas, "Probabilistic PERT," *IBM J. Res. Dev.*, vol. 23, pp. 339–347, 1979.

[4] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," in *Proceedings of the 6th Heterogeneous Computing Workshop (HCW '97)*, 1997, pp. 172–184.

[5] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for mpsoc," in *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 598–603.

[6] A. Kamthe and S.-Y. Lee, "A stochastic approach to estimating earliest start times of nodes for scheduling dags on heterogeneous distributed computing systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 1 - Volume 02*, 2005, p. 121b.

[7] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '01, 2001, pp. 620–625.

[8] M. Lombardi and M. Milano, "Allocation and scheduling of conditional task graphs," *Artif. Intell.*, vol. 174, pp. 500–529, 2010.

[9] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The recognition of series parallel digraphs," in *Proceedings of the eleventh annual ACM symposium on Theory of computing*, ser. STOC '79, 1979, pp. 1–12.