JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **44,** 35–52 (1997)
ARTICLE NO. PC971349

# Determining the Execution Time Distribution for a Data Parallel Program in a Heterogeneous Computing Environment[1]

Yan Alexander Li,* John K. Antonio,† Howard Jay Siegel,‡ Min Tan,‡ and Daniel W. Watson§

**Intel Corporation, SC9-15, 2200 Mission College Boulevard, Santa Clara, California 95052-8119; †Department of Computer Science, Texas Tech University, Lubbock, Texas 79409-3104; ‡Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-1285; and §Department of Computer Science, Utah State University, Logan, Utah 84322-4205*

**An important problem in heterogeneous computing (HC) is predicting task execution time. A methodology is introduced for determining the execution time distribution for a given data parallel program that is to be executed in an SIMD, MIMD (SPMD), and/or mixed-mode SIMD/MIMD (SPMD) HC environment. The program is assumed to contain operations and constructs whose execution times depend on input-data values. The methodology uses a block-based approach to transform the program into a flow analysis tree and computes the execution time distribution for the program, given the execution modes for each node in the flow analysis tree, an estimated execution time distribution for each operation in both modes, and appropriate probabilistic models for control and data conditional constructs. The results are directly applicable to both mixed-machine and mixed-mode HC systems.** 

## 1. INTRODUCTION

A heterogeneous computing (*HC*) system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [SiA96, SiD97]. Two types of HC systems are mixed-mode machines and mixed-machine systems. A *mixed-mode machine* is defined here as a single parallel processing machine that is capable of operating in either the synchronous SIMD [Fly66] or asynchronous MIMD [Fly66] mode of parallelism and can dynamically switch between modes at instruction-level granularity [SiM96]. A *mixed-machine* system is a suite of independent machines of different types interconnected by a high-speed network.

For HC systems, *matching* involves deciding on which machine/mode each code block should be executed [SiA96]. Mapping for parallel and distributed computing systems, which is closely related to matching for HC systems, has been studied extensively in the past. Much of the work in mapping for parallel and distributed systems has focused on how to effectively execute multiple subtasks across a network of sequential processors/machines (e.g., see [CaK88, NiH81, NoT93]). In such an environment, load balancing can be an effective way to improve response time and throughput. Although some of the existing mapping concepts and techniques for parallel and distributed computing systems can be (and have been) applied to matching for HC systems, there is a fundamental distinction between *mapping* subtasks for a network of sequential processors/machines (e.g., a network of workstations) and *matching* subtasks for an HC system. In the latter case, the subtasks can be characterized based on "type of computation" present in each subtask to account for the fact that certain types of subtasks may execute most effectively using a particular machine/mode. In an HC environment, matching subtasks to machines/modes of the appropriate types is generally a more important factor than merely balancing the load.

To effectively utilize the computational resources in an HC system for executing the subtasks that compose a task, it is very important to be able to predict the total task execution time that results from any particular assignment of subtasks to modes/machines. This total task performance prediction is the basis of matching for HC systems.

Many matching and scheduling algorithms make the simplifying assumption that the execution time for each subtask is a known constant for each mode/machine in the system (e.g., [ChE93, Fre89, WaA94, WaS94]). However, there are elements of uncertainty, such as the uncertainty in input data values or in intermachine communication time, which can impact the execution time. Mode/machine choices for executing subtasks can also affect the execution time and its degree of uncertainty. For example, in a mixed-mode machine, during an MIMD to SIMD mode switch, all processors must wait for the last one to finish its MIMD execution before entering the synchronous SIMD mode. Thus, the amount of time a particular processor waits depends not only on when it finishes MIMD execution, but also on when the last processor finishes MIMD execution.

The *SPMD* (single program–multiple data) mode of parallelism is a special case of MIMD in which the processors

[1]This work was supported by Rome Laboratory under Contract F30602-94-C-0022, and by NRaD under Subcontract 20-950001-70.



0743-7315/97 $25.00

execute the same program asynchronously on their own data [DaG88]. Applications for this study are assumed to be data parallel programs written in a mode-independent language for execution on an SIMD/SPMD mixed-mode machine. Examples of mixed-mode machines include EXECUBE [Kog94], MeshSP [ICE95], OPSILA [DuB88], PASM [SiS96], TRAC [LiM87], and Triton [PhW93]. The model of a mixed-mode machine assumed here is a distributed memory machine, in which each processor is paired with a memory module to form a processing element (*PE*). When a PE switches mode, all that changes is the source of its instructions. For SIMD mode, PEs receive their instructions from a common control unit (*CU*), while in SPMD mode, each PE fetches its instructions from its own memory module.

Studies on how to make effective use of the heterogeneity present within mixed-mode machines can provide useful insights for how to make effective use of mixed-machine systems. For example, in [WaS94] an optimal mode selection technique was developed for mixed-mode machines that was later generalized for use with mixed-machine systems in [WaA94]. Similarly, much of the work presented here for predicting execution times for mixed-mode machines is also applicable and/or adaptable to mixed-machine systems. In particular, for a mixed-machine system consisting of SIMD, MIMD, and/or mixed-mode machines, much of the proposed methodology is directly applicable.

In a mixed-machine HC system, one component of the decision of whether to execute a given sequence of data parallel subtasks on a particular SIMD versus MIMD machine is predicting the execution time for that sequence on each machine. The proposed methodology will support these predictions when nondeterministic data-dependent values impact execution time. These predicted values could then be used in a subtask/machine matching scheme that also incorporates other factors, such as intermachine communication time (e.g., [WaA94]). Thus, while this work is being presented in mixed-*mode* machine context, the results from the single mode analysis presented are a necessary part of any automatic mixed-*machine* matching scheme.

Consider the execution of the loop in Fig. 1 on a mixed-mode machine. The loop body contains subtasks A and B. Assume the execution times of each subtask vary among the PEs (because the execution time of each subtask depends on input data values that vary across all PEs) and the loop control overhead time is ignored. Assume that, when executed independently, the average execution time of subtask A is minimal in SIMD mode and the average execution time of subtask B is minimal in MIMD mode. In Fig. 1, the wide

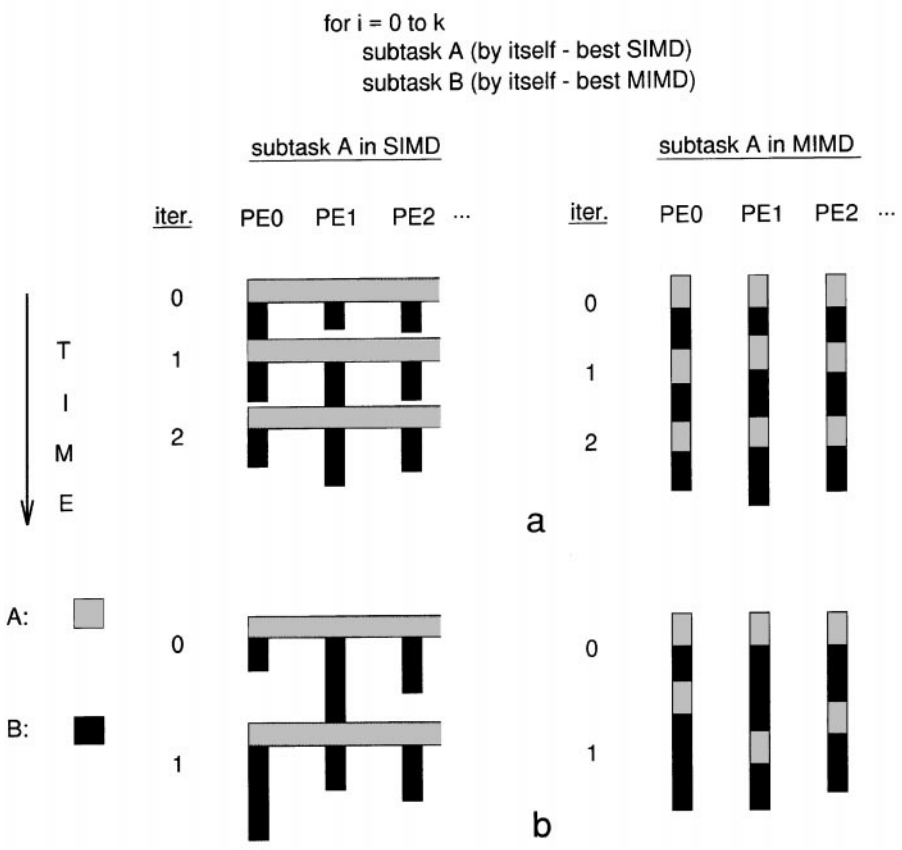


**FIG. 1.** Execution of a loop with subtasks of variable execution time: (a) low variance for execution time of subtask B; (b) high variance for execution time of subtask B.

rectangles spanning horizontally across all PE labels represent the execution time of a subtask in SIMD mode. The thin rectangles under each PE label represent the execution time in MIMD mode.The rectangles are shaded differently to represent the execution times of subtasks A and B.

Intuitively, executing subtask A in SIMD mode and subtask B in MIMD mode should be faster than any other combination (because the average execution time of subtask A is faster in SIMD and the average execution time of subtask B is faster in MIMD). This intuition is correct provided that the variation in execution time across the PEs is sufficiently small, as shown in Fig. 1a. However, as shown in Fig. 1b, if the variation is large across the PEs, then it is possible that executing both subtasks in MIMD mode is faster due to the effect of block juxtaposition, even if the average execution time of subtask A in MIMD mode is larger [BeK91]. Therefore, incorporating only information for average execution times may lead to incorrect machine/mode selections, because the effect of block juxtaposition is not captured.

The methodology proposed here, which does account for complicated effects like block juxtaposition, statically estimates the execution time distribution for a given data parallel program in an SIMD/SPMD mixed-mode computing environment. The program is assumed to contain operations whose execution time behaviors depend on input data values that cannot be perfectly predicted at compile time. For instance, in the example shown in Fig. 1, the number of iterations executed by the looping construct may be data dependent. Also, each subtask within the loop body may itself contain looping constructs where the number of iterations to be executed is uncertain. Probabilistic models are constructed to model these types of uncertainties, e.g., a probability density function is used to represent the number of iterations executed by each looping construct. The aggregate effect of these elements of uncertainty in the program is captured by computing the probability distribution for the total execution time.

In the proposed methodology, a block-based approach is used to transform the application program into a flow analysis tree in which the internal nodes represent control or data conditional constructs and the leaf nodes represent basic code blocks [AhS86]. The methodology takes as input the structure of the flow analysis tree, the mode in which each node in the flow analysis tree is to be executed (SIMD or SPMD), execution time distributions for all basic operations for both SIMD and SPMD modes, and appropriate probabilistic models for control and data conditional constructs. Based on this information, the execution time distribution for the entire program is computed. Deriving this proposed methodology for combining statistical information about an SIMD/SPMD mixed-mode program is the focus of this paper.

Section 2 presents the basic assumptions and a brief overview of the proposed approach. Methods for computing the execution time distribution of a single code block in either SIMD or SPMD mode are discussed in Section 3. The methods for computing the execution time distribution for the entire program executed in SPMD, SIMD, and mixed-mode are introduced in Sections 4, 5, and 6, respectively. Section 7 presents a hypothetical numerical example and an application study to demonstrate the effect of mode selections on the distribution of total execution time. The Appendix reviews the basic probability theory and notation used here.

## 2. OVERVIEW OF THE APPROACH

Applications for this study are assumed to be data parallel programs written in a mode-independent language for execution on an SIMD/SPMD mixed-mode machine. A *mode-independent language* (e.g., see [NiS93, WeW94]) is a language in which syntactic elements have interpretations under more than one mode of parallelism, and operations represent the most explicit level at which the program representation is identical for each mode of parallelism. Such languages make it possible to utilize the most appropriate parallel execution mode for each block of a given program.

As in the *BBMS* (block-based mode selection) framework introduced in [WaS94], a flow-analysis tree is used to represent the application program. The application program is divided into *code blocks*, identified by their leading statements called *leaders* [AhS86]. The first statement in a program is a leader, any statement that is a target of a branch at the machine-code level is a leader, any statement following a conditional branch at the machine-code level is a leader, and any statement requiring or following a synchronization or an inter-PE communication is a leader. After the code blocks are defined, the program is transformed into a flow analysis tree, whose structure represents the scope levels within the program. The root of the tree represents the scope of the entire program. The nonleaf nodes (excluding the root) represent control and data-conditional constructs. Code blocks are represented by the leaf nodes. An example program and its associated flow analysis tree are shown in Fig. 2. A simple model for the language is assumed here, as in [WaS94]; i.e., the only control constructs are loops and data conditionals.

It is assumed that leaf nodes (i.e., code blocks) are executed completely in either SIMD or SPMD mode, and mode changes are allowed only at interblock boundaries. It is also assumed that the sibling nodes are executed in an ordered sequence (from left to right) as they appear in the flow-analysis tree. Thus, the schedule for executing the code blocks is static and is defined by the program itself. Because decisions are made statically, the choice of mode for a block within a loop is the same for all loop iterations. Each iteration of a loop must begin and end execution in the same mode of parallelism (otherwise, a mode switch would need to be added to make this true). All blocks that are part of (i.e., descendants of) a data-conditional construct are executed in the same mode of parallelism (e.g., this is a requirement in the operation of PASM to avoid complex and costly bookkeeping overhead).

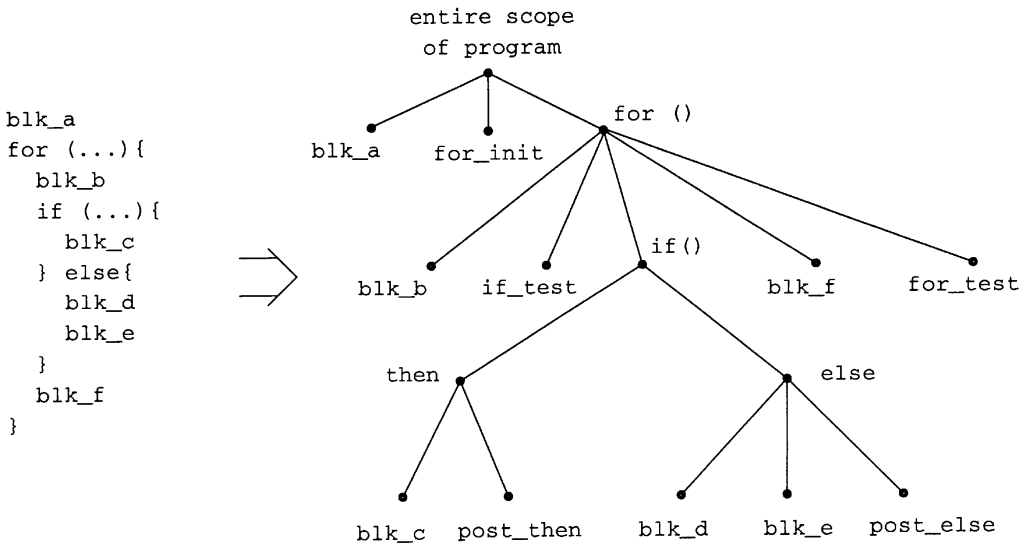


**FIG. 2.** Example program and its associated flow-analysis tree [WaS94].

The execution time of each basic operation within a code block in each mode on a single PE can either be deterministic (i.e., have a constant value) or an assume different values, each with a specified probability. In both cases, a discrete random variable is used to model the execution time, where the former is a special case in which the random variable assumes a given constant value with probability 1 (i.e., it is deterministic). An estimate for the execution time distribution of each operation in each mode is assumed to be known.

Because code blocks do not contain conditional or looping constructs, the execution times of most basic operations within a code block can be modeled as known constants (i.e., deterministic values) in practice. A possible exception is in modeling the time required for inter-PE communication operations. For these types of operations, general probability distributions are employed to model the uncertainty due to network contention for the particular parallel machine being used. For a given application domain, the probability distributions for all basic operations are assumed to be independent of: (a) the particular program within the domain being considered and (b) the values of the input data for the program. Thus, these distributions can be measured for a given machine (i.e., empirically estimated) and stored in a database to be referenced by the proposed approach.

It is assumed that the branching probability of each data conditional construct and the distribution for the number of iterations each loop will execute are application/data dependent and are available from the application programmer (e.g., in the form of compiler directives). In general, the more accurate the information that the application programmer provides, the better the prediction that can be made on the execution time distribution for the entire program. For example, a program may contain an exception-handling branch that takes a long time to execute. If the application programmer can indicate that this exception occurs only rarely (i.e., with low probability), then it can be predicted that the total execution time distribution is affected only slightly. Otherwise, an arbitrary assumption, such as using a branching probability of approximately one-half, gives a more pessimistic estimate for the total execution time distribution. In addition to getting information directly from the application programmer, empirical information can be derived based on a number of measured execution times with a representative sampling of data sets.

Given the above information and the mode of parallelism (i.e., SIMD or SPMD) in which each code block is to be executed, the execution time distribution of each block—which corresponds to a leaf node in the flow-analysis tree—is computed. After that, traversing the flow-analysis tree in depth-first order, each lowest level subtree is pruned. Repeating this step, the entire flow-analysis tree is pruned, and the execution time distribution of the whole program is computed. Related probability theory and notation can be found in textbooks on the subject (e.g., [MoG74]) and are reviewed in the Appendix. In the next section, the approach to compute the execution time distribution of a code block is introduced.

## 3. EXECUTION TIME DISTRIBUTION OF A CODE BLOCK

Throughout the rest of the paper, it is assumed that time is measured based on a given discrete unit and thus assumes nonnegative integer values. All execution times are modeled as discrete random variables. The case of a constant execution time is regarded as a special case where the random variable is equal to the specified constant with probability 1.

It is assumed that there are $N$ PEs in the mixed-mode machine. The code block associated with the $c$th leaf node in the flow-analysis tree is called *code block c*. Let $k_c$ denote the number of operations in code block $c$, and label the operations in code block $c$ as 0, 1, $\cdots$, $k_c - 1$. For each code block $c$, define two arrays of discrete random variables $I_c^{i,j}$ and $P_c^{i,j}$, $0 \le i \le k_c - 1$, $0 \le j \le N - 1$. The values of the random variables $I_c^{i,j}$ and $P_c^{i,j}$ correspond to the execution time of operation $i$ of block $c$ executed on PE $j$ in S*I*MD and S*P*MD modes, respectively. It is assumed that for each operation $i$, $I_c^{i,j}$ are independent and identically distributed (*i.i.d.*) for all PEs $j$. Likewise, for each PE $j$, $I_c^{i,j}$ are independent for all operations $i$. Both of these assumptions are also made for $P_c^{i,j}$.

In general, for SIMD execution a subset of the PEs may be disabled during execution of a block within a data conditional or loop. This is because in SIMD mode only the enabled PEs execute the instructions broadcast by the CU. Also, the number of enabled PEs for an SPMD block within a mixed-mode loop can vary from one iteration to the next. Thus, the number of enabled PEs is relevant for SPMD blocks as well. Because a code block itself contains neither data conditionals nor loops, any given PE must be either enabled to execute the whole block or disabled for the whole block. Thus, the number of enabled PEs does not change during the execution of a code block. In this section, the execution time distribution of a code block is derived for $e$ enabled PEs, where $e$ is an integer and $0 \le e \le N$. Based on the i.i.d. assumption for each operation across all PEs, for $e \ge 1$, the set of enabled PEs can be numbered 0, 1, $\cdots$, $e - 1$ without loss of generality.

For each code block $c$ executed in SPMD mode, define two random variables $\tilde{P}_c^j$ and $P_{c,e}$. The value of $\tilde{P}_c^j$ corresponds to the execution time of code block $c$ in SPMD mode on PE $j$, and the value of $P_{c,e}$ corresponds to the execution time of code block $c$ in SPMD mode with $e$ enabled PEs synchronized at the beginning and end of the block. There are two cases to consider: $e \ge 1$ and $e = 0$. For $e \ge 1$, the random variables $\tilde{P}_c^j$ and $P_{c,e}$ are determined from $P_c^{i,j}$ as

$$\tilde{P}_c^j = \sum_{i=0}^{k_c-1} P_c^{i,j}$$

$$P_{c,e} = \max_{0 \le j \le e-1} \{\tilde{P}_c^j\}.$$

This "max of sums" effect was first described in [FiC88]; however, each $\tilde{P}_c^j$ was a scalar value and not a random variable. The density function of $\tilde{P}_c^j$ and the distribution function of $P_{c,e}$ can be computed as [2]

$$f_{\tilde{P}_c^j}(\cdot) = f_{P_c^{0,j}}(\cdot) * f_{P_c^{1,j}}(\cdot) * \cdots * f_{P_c^{k_c-1,j}}(\cdot),$$

$$F_{P_{c,e}}(\cdot) = \prod_{j=0}^{e-1} F_{\tilde{P}_c^j}(\cdot) = (F_{\tilde{P}_c^0}(\cdot))^e.$$

For $e = 0$ (i.e., no enabled PEs), the block is assumed to take 0 time units with probability 1. Let

$$\delta(i) = \begin{cases} 1, & i = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Then, the associated distribution and density functions are given by

$$F_{P_{c,0}}(\cdot) = 1,$$
$$f_{P_{c,0}}(\cdot) = \delta(\cdot).$$

For each code block $c$ executed in SIMD mode, define a random variable $I_{c,e}$, and for each operation $i$ in this block, define a random variable $\tilde{I}_{c,e}^i$; both assume there are $e$ enabled PEs. The value of $\tilde{I}_{c,e}^i$ corresponds to the execution time of operation $i$ of code block $c$ in SIMD mode, and the value of $I_{c,e}$ corresponds to the execution time of code block $c$ in SIMD mode with $e$ enabled PEs. In SIMD mode, because every operation in the block is synchronized among all enabled PEs, the execution time of an operation is the maximum of the execution times of this operation among all enabled PEs. There are two cases to consider: $e \ge 1$ and $e = 0$. For $e \ge 1$, the random variables $\tilde{I}_{c,e}^i$ and $I_{c,e}$ are determined from $I_c^{i,j}$ as

$$\tilde{I}_{c,e}^i = \max_{0 \le j \le e-1} \{I_c^{i,j}\}$$

$$I_{c,e} = \sum_{i=0}^{k_c-1} \tilde{I}_{c,e}^i.$$

Then, the distribution function of $\tilde{I}_{c,e}^i$ and the density function of $I_{c,e}$ are given by [3]

$$F_{\tilde{I}_{c,e}^i}(\cdot) = \prod_{j=0}^{e-1} F_{I_c^{i,j}}(\cdot) = (F_{I_c^{i,0}}(\cdot))^e$$

$$f_{I_{c,e}}(\cdot) = f_{\tilde{I}_{c,e}^0}(\cdot) * f_{\tilde{I}_{c,e}^1}(\cdot) * \cdots * f_{\tilde{I}_{c,e}^{k_c-1}}(\cdot).$$

This "sum of maxs" effect was first described in [FiC88]; however, each $\tilde{I}_c^j$ was a scalar value and not a random variable.

For $e = 0$, the block is assumed to take 0 time units with probability 1. Thus, the associated distribution and density functions are given by

$$F_{I_{c,0}}(\cdot) = 1,$$
$$f_{I_{c,0}}(\cdot) = \delta(\cdot).$$

Therefore, associated with each code block $c$ are $2(N + 1)$ random variables, $P_{c,e}$ and $I_{c,e}$, $0 \le e \le N$. The values of $P_{c,e}$ and $I_{c,e}$ represent the execution times of code block $c$ with $e$ enabled PEs in SPMD and SIMD modes, respectively.

## 4. SPMD EXECUTION OF AN ENTIRE PROGRAM

### 4.1. Overview of SPMD Execution

This section presents a methodology for computing the execution time distribution of an entire program executed in

[2] $F_{\tilde{P}_c^j}(\cdot) = F_{\tilde{P}_c^0}(\cdot)$ for all $j$ because of the i.i.d. assumption.

[3] $F_{I_c^{i,j}}(\cdot) = F_{I_c^{i,0}}(\cdot)$ for all $j$ because of the i.i.d. assumption.

SPMD mode. It is assumed that $e$ PEs ($1 \le e \le N$) are initially enabled, and all these $e$ PEs remain enabled throughout the execution of the entire program. A method for modeling a series of blocks by a single block having equivalent execution time characteristics is introduced first. Then, methods for modeling two basic program structures, pure data conditional construct and pure loop, are presented. A *pure loop* is a loop whose body is currently represented as a series of one or more leaf blocks. Similarly, a *pure data conditional construct* is a data conditional construct for which the "then" clause and the "else" clause are both currently represented as a series of one or more leaf blocks. It is shown that pure data conditional construct and pure loop can each be modeled as a single code block. Finally, it is shown how these methods can be combined in a divide-and-conquer way to handle arbitrary program structures.

### 4.2. A Series of Blocks

For a series of $C$ ($C \ge 2$) blocks (labeled 0, 1, $\cdots$, $C-1$) executed in SPMD mode, because no synchronization among the PEs at the block boundaries is explicitly specified in the source code, each PE will execute the operations of all blocks as one contiguous block. Therefore, this series can be modeled as an SPMD block $C$ with

$$\tilde{P}_C^j = \sum_{c=0}^{C-1} \sum_{i=0}^{k_c-1} P_c^{i,j} = \sum_{c=0}^{C-1} \tilde{P}_c^j$$

$$f_{\tilde{P}_C^j}(\cdot) = f_{\tilde{P}_0^j}(\cdot) * f_{\tilde{P}_1^j}(\cdot) * \cdots * f_{\tilde{P}_{C-1}^j}(\cdot).$$

If all $e$ PEs are synchronized before block 0 and after block $C-1$, then

$$P_{C,e} = \max_{0 \le j \le e-1} \{\tilde{P}_C^j\}$$

$$F_{P_{C,e}}(\cdot) = \prod_{j=0}^{e-1} F_{\tilde{P}_C^j}(\cdot) = (F_{\tilde{P}_C^0}(\cdot))^e.$$

### 4.3. Pure Data Conditional Construct

For a pure data conditional construct, if either the "then" clause or the "else" clause consists of a series of two or more blocks, then the series of blocks can be modeled as a single code block using the techniques discussed in the previous subsection. Hence, the focus of this subsection is on data conditional constructs in which each of the "then" and "else" clauses is represented by a single block.

Assume node $d$ of the flow-analysis tree corresponds to a pure data conditional construct, and block $t$ and block $s$ are its "then" and "else" clauses, respectively. It is assumed that PE $j$ executes the "then" clause with probability $p_t^j$, and this branching probability is independent and identical across the PEs. Thus, each PE $j$ executes the "else" clause with probability $1 - p_t^j$. Because each PE fetches instructions from its own memory in SPMD mode, some PEs may execute the "then" clause while the others are executing the "else" clause during the execution of a data conditional construct. By applying the Total Probability Theorem [MoG74], the whole data conditional construct can be represented by a single SPMD block having an equivalent execution time distribution. For each PE $j$,

$$f_{\tilde{P}_d^j}(\cdot) = p_t^j f_{\tilde{P}_t^j}(\cdot) + (1 - p_t^j) f_{\tilde{P}_s^j}(\cdot).$$

If all $e$ PEs are synchronized prior to and after completion of node $d$, the random variable and its associated distribution function for the execution time of node $d$ are given by

$$P_{d,e} = \max_{0 \le j \le e-1} \{\tilde{P}_d^j\}$$

$$F_{P_{d,e}}(\cdot) = \prod_{j=0}^{e-1} F_{\tilde{P}_d^j}(\cdot) = (F_{\tilde{P}_d^0}(\cdot))^e.$$

### 4.4. Pure Loop

For each pure looping construct, it is assumed that the given distribution for the number of iterations to be executed by each PE is i.i.d. across all PEs. This simplifying assumption makes the analysis tractable. If the pure loop body consists of a series of two or more blocks, then the series can be modeled as a single code block using the techniques discussed in Subsection 4.2. Therefore, the focus here is on determining the execution time distribution of a loop whose body is a single block.

Let node $\ell$ of the flow-analysis tree correspond to a pure loop whose body is modeled by block $b$, and the random variable for the number of iterations to be executed by PE $j$ is $\tilde{R}_\ell^j$.The value of $f_{\tilde{R}_\ell^j}(r)$ corresponds to the probability that PE $j$ will execute exactly $r$ iterations. For all $r \le 0$, $f_{\tilde{R}_\ell^j}(r) = 0$ (i.e., it is assumed that each PE $j$ will execute at least one iteration). Let $r_{\max}$ denote the maximum possible (i.e., with nonzero probability) number of iterations to be executed. Thus, for all $r > r_{\max}$, $f_{\tilde{R}_\ell^j}(r) = 0$. For each $r$, $1 \le r \le r_{\max}$, conceptually unroll the loop into a series of $r$ blocks (i.e., repeat loop body block $b$, $r$ times). The density function of the execution time of this series on PE $j$ is the convolution of $r$ density functions of $\tilde{P}_b^j$. By the Total Probability Theorem, the density function for the execution time of the looping construct on PE $j$ is the weighted sum of the density functions of the execution time of the unrolled loop for all possible $r$, i.e.,

$$f_{\tilde{P}_\ell^j}(\cdot) = \sum_{r=1}^{r_{\max}} f_{\tilde{R}_\ell^j}(r) \underbrace{(f_{\tilde{P}_b^j}(\cdot) * \cdots * f_{\tilde{P}_b^j}(\cdot))}_{r \text{ times}}.$$

If there are synchronizations prior to and after completion of the loop, then the random variable and its associated distribution function for the execution time across all $e$ PEs are given by

$$P_{\ell,e} = \max_{0 \le j \le e-1} \{\tilde{P}_\ell^j\}$$

$$F_{P_{\ell,e}}(\cdot) = \prod_{j=0}^{e-1} F_{\tilde{P}_\ell^j}(\cdot) = (F_{\tilde{P}_\ell^0}(\cdot))^e.$$

### *4.5. Arbitrary Program Construct*

It was shown in the previous two subsections that any subtree corresponding to a pure loop or a pure data conditional construct can be modeled as a single leaf node. Thus a divide-and-conquer approach can be used to calculate the execution time distribution of an arbitrary program.

Traversing the flow-analysis tree in depth-first order, each nonleaf node traversed can only have leaf nodes as its children. (A node is said to be traversed if and only if all its children have been traversed.) Therefore, each such nonleaf node (excluding the root node) corresponds to one of the following program structures: (1) a pure loop; (2) a pure data conditional construct; or (3) a clause of a data conditional construct that contains a series of blocks. The applicable technique is then used to model the subtree under this node by a single leaf node. This procedure is repeated until the children of the root node are represented as a series of (composite) leaf blocks. The SPMD execution time distribution of the entire program is thus the execution time distribution of the SPMD block $B_R$ modeling this series, denoted as $P_{B_R N}$ (assuming $N$ PEs are initially enabled to execute the program).

## 5. SIMD EXECUTION OF AN ENTIRE PROGRAM

### *5.1. Overview of SIMD Execution*

A methodology for computing the execution time distribution of an entire program executed in SIMD mode is presented. As discussed in Section 3, the number of enabled PEs associated with various scope levels of the program may be distinct for SIMD execution. Therefore, more bookkeeping (as compared with SPMD execution of an entire program) is needed to account for the number of enabled PEs for each portion of the program. Assuming $N$ PEs are used to execute the program in SIMD mode, each code block has $N + 1$ associated execution time distributions, one for each possible number $e$ of enabled PEs, $0 \le e \le N$. It is shown that each program construct can be modeled as a single block that has an equivalent execution time distribution for each possible value of $e$.

The organization of this section is similar to that of Section 4. A method for modeling a series of blocks by a single block having equivalent execution time characteristics is introduced first. A series of blocks can appear as children of the root node, within a clause of a data conditional, or the body of a loop. Therefore, the number of enabled PEs remains the same during the execution of whole series. (However, between iterations of a loop, the number of enabled PEs may decrease in SIMD and mixed-mode; this issue is addressed in Subsections 5.4 and 6.3.) It is shown that a pure data conditional construct and a pure loop can each be modeled as a single code block in SIMD mode. Finally, it is shown how these methods can be combined in a divide-and-conquer manner to model an arbitrary program structure as a single code block.

### *5.2. A Series of Blocks*

For a series of $C$ ($C \ge 2$) blocks executed in SIMD mode, because each operation is synchronized across all PEs, the total execution time is the sum of execution times of all blocks. Thus, this series is modeled as one SIMD block $C$ such that for each $e$, $0 \le e \le N$,

$$I_{C,e} = \sum_{c=0}^{C-1} I_{c,e}$$

$$f_{I_{C,e}}(\cdot) = f_{I_{0,e}}(\cdot) * f_{I_{1,e}}(\cdot) * \cdots * f_{I_{C-1,e}}(\cdot).$$

### *5.3. Pure Data Conditional Construct*

Consider the execution of a pure data conditional construct in SIMD mode with $e$ enabled PEs. Assume that the condition is evaluated in each PE with its own data. (The case of evaluating the condition in the CU is a simpler case, which is discussed later.) When instructions of the "then" clause are broadcast by the CU, PEs for which the condition is false are disabled; when instructions of the "else" clause are broadcast, PEs for which the condition is true are disabled. If all $e$ PEs are to execute the same clause, then the other clause is skipped; i.e., the instructions for the other clause are not broadcast. (In some systems it is less costly to simply broadcast the clause no PEs will execute, rather than test for this situation).

Recall that node $d$ corresponds to a pure data conditional construct to be executed in SIMD mode, and its "then" and "else" clauses can be represented as the single code blocks $t$ and $s$, respectively (using the techniques described in Subsection 5.2). PE $j$ executes the "then" clause with probability $p_t^j$, and this branching probability is independent and identical across the PEs. Let $e_t$, $0 \le e_t \le e$, denote the number of enabled PEs during the execution of block $t$. Thus, the number of enabled PEs during the execution of block $s$ is $e - e_t$. Let $p_t^0 = p_t^j$ for all enabled PEs $j$. The probability for each possible value of $e_t$ is defined by a binomial distribution

$$\Pr[e_t = k] = \begin{cases} \binom{e}{k} (p_t^0)^k (1 - p_t^0)^{e-k} & 0 \le k \le e \\ 0 & \text{otherwise.} \end{cases}$$

By the Total Probability Theorem, the entire data conditional construct (i.e., node $d$) can be represented by a single block having the execution time density function

$$f_{I_{d,e}}(\cdot) = \sum_{k=0}^{e} \Pr[e_t = k](f_{I_{t,k}}(\cdot) * f_{I_{s,e-k}}(\cdot)), \quad 0 \le e \le N.$$

When the condition is evaluated in the CU using CU data, all PEs execute the same clause. Assume $p_t$ is the probability that the condition is satisfied, then all PEs execute block $t$ with probability $p_t$ and block $s$ with probability $1 - p_t$. Node $d$ can be represented by a single SIMD block with the execution time density function

$$f_{I_{d,e}}(\cdot) = p_t\, f_{I_{t,e}}(\cdot) + (1 - p_t)\, f_{I_{s,e}}(\cdot), \qquad 0 \le e \le N.$$

### 5.4. Pure Loop

Consider the execution of a pure loop in SIMD mode. It is assumed that bounds for looping iterations are based on local data from each PE. (The case of a common loop bound in the CU is simpler and will be discussed later.) Then the number of iterations to be executed by each PE can be distinct. The CU must broadcast instructions for the PE(s) that executes the largest number of iterations. Those PEs that finish earlier are disabled until the loop is finished. Therefore, as the number of iterations increases, the number of enabled PEs is nonincreasing. For each iteration, the number of enabled PEs is the same across the loop body because the loop body is represented as a series of leaf blocks.

Recall that node $\ell$ of the flow-analysis tree corresponds to a pure loop whose body is denoted by block $b$. (If a pure loop body consists of a series of blocks, the series can be modeled as a single code block using the techniques discussed previously.) Recall that the random variable associated with the number of iterations to be executed by PE $j$ is denoted by $\tilde{R}_\ell^j$, and $r_{\max} \ge 1$ denotes the maximum possible (i.e., with nonzero probability) number of iterations to be executed. Thus, if $r < 1$ or $r > r_{\max}$, then $f_{\tilde{R}_\ell^j}(r) = 0$. Because $f_{\tilde{R}_\ell^j}(\cdot) = f_{\tilde{R}_\ell^0}(\cdot)$ for all $j$, for $1 \le r \le r_{\max}$, the probability that any PE $j$ executes $r$ or more iterations is given by

$$\rho_r = \sum_{i=r}^{r_{\max}} f_{\tilde{R}_\ell^0}(i). \tag{1}$$

Therefore, if $r = 1$, then $\rho_r = 1$, and if $r > r_{\max}$, then $\rho_r = 0$.

Conceptually unroll the loop $r_{\max}$ times and label the iterations as $1, 2, \cdots, r_{\max}$. Let $X_{r,e}$ denote the random variable associated with the combined SIMD execution time of all iterations starting from (and including) iteration $r$, given that $e$ PEs are enabled for iteration $r$. Obviously, if $r = r_{\max}$, then $X_{r,e} = I_{b,e}$. The objective is to determine $X_{1,e} = I_{\ell,e}$, which corresponds to the $N + 1$ execution time distributions of the entire loop. By considering the iterations in reverse order (i.e., $r_{\max}$ down to 1), density functions of $X_{r,e}$ for each $e$ ($0 \le e \le N$) are computed based on previously computed density functions of $X_{r+1,h}$ ($0 \le h \le e$). If a PE is enabled for iteration $r$, then the ratio $\rho_{r+1}/\rho_r$ represents the probability that it will be enabled at iteration $r + 1$. To compute the density functions for $X_{r,e}$, $1 \le r < r_{\max}$, it is necessary to know the probability

for $e \leftarrow 0$ to $N$

  $f_{X_{r_{\max},e}}(\cdot) \leftarrow f_{I_{b,e}}(\cdot)$

for $r \leftarrow r_{\max} - 1$ down to 1

  for $e \leftarrow 0$ to $N$

    $f_{X_{r,e}}(\cdot) \leftarrow f_{I_{b,e}}(\cdot) * \left(\sum_{h=0}^{e} \beta_{r,e}(h) f_{X_{r+1,h}}(\cdot)\right)$

for $e \leftarrow 0$ to $N$

  $f_{I_{\ell,e}}(\cdot) \leftarrow f_{X_{1,e}}(\cdot)$

**FIG. 3.** Algorithm for computing SIMD loop execution time density function.

that $h$ PEs, $0 \le h \le e$, will execute iteration $r + 1$, given that $e$ PEs executed iteration $r$. This is given by binomial distribution

$$\beta_{r,e}(h) = \binom{e}{h} \left(\frac{\rho_{r+1}}{\rho_r}\right)^h \left(1 - \frac{\rho_{r+1}}{\rho_r}\right)^{e-h}.$$

By applying the algorithm of Fig. 3, loop $\ell$ is modeled as a single SIMD block, whose execution time density function, $f_{I_{\ell,e}}(\cdot)$, is determined for each $e$, $0 \le e \le N$.

If iterations are based on a common loop bound in the CU, then all PEs execute the same number of iterations. Let $R_\ell$ denote the random variable associated with the number of iterations to be executed (based on CU data). Recall $r_{\max} \ge 1$ denotes the maximum possible number of iterations to be executed. Then node $\ell$ can be represented by a single SIMD block with the execution time density function

$$f_{I_{\ell,e}}(\cdot) = \sum_{r=1}^{r_{\max}} f_{R_\ell}(r)\Big(\underbrace{f_{I_{b,e}}(\cdot) * \cdots * f_{I_{b,e}}(\cdot)}_{r \text{ times}}\Big), \qquad 0 \le e \le N.$$

### 5.5. Arbitrary Program Construct

As in Subsection 4.5, by traversing the flow-analysis tree in depth-first order, each nonleaf node traversed can only have leaf nodes as its children. Therefore, each such nonleaf node (excluding the root node) corresponds to one of the following program structures: (1) a pure loop; (2) a pure data conditional construct; or (3) a clause of a data conditional construct that contains a series of blocks. The applicable technique is then used to model the subtree under this node as a single leaf node. (Note that associated with each leaf node is $N + 1$ different distributions; one for each possible number of enabled PEs.) This procedure is repeated until children of the root node are represented as a series of (composite) leaf nodes. The SIMD execution time distribution of the entire program is thus the execution time distribution of the equivalent SIMD leaf node

$B_R$ for this series, denoted as $I_{B_R, N}$ (assuming $N$ PEs are initially enabled to execute the program).

## 6. MIXED-MODE EXECUTION OF A PROGRAM

### *6.1. Overview of Mixed-Mode Execution*

A methodology is presented to estimate the execution time distribution of a program executed in mixed-mode. Many of the concepts developed in Sections 4 and 5 are used for the mixed-mode analysis. In contrast to single-mode execution, during mixed-mode execution of a program, a significant factor that affects the overall execution time distribution is the possible difference in execution time distribution for each code block associated with SIMD and SPMD modes. In addition, mode switching overheads and the synchronization required at a mode switch from SPMD to SIMD also play important roles in shaping the distribution. The number of enabled PEs must be considered for each scope level of the program.

For a mixed-mode program, the calculation of the execution time distribution for an individual code block is the same as in single-mode execution. The appropriate single-mode modeling technique is first applied to represent any single-mode subtree as a single code block with equivalent execution characteristics. This includes all data conditional constructs, because all descendants of a data conditional construct must be executed in the same mode (as explained in Section 2). It will be shown that, in contrast to single-mode execution, where a series of blocks or a loop can be modeled as a single code block, in mixed-mode execution, a series of blocks or a pure loop can be modeled as a series of at most three blocks with equivalent execution characteristics. A method for combining these techniques to model arbitrary mixed-mode program execution is presented.

### *6.2. A Series of Mixed-Mode Blocks*

It was demonstrated earlier that the single-mode execution of a series of blocks can be modeled as a single code block. Thus, a series of blocks executed in mixed-mode can be modeled as a series of alternating SIMD and SPMD blocks. A series of SIMD and SPMD blocks that compose all the children of an arbitrary node can begin and end with either mode, so there are four cases to consider: (1) begin with SIMD and end with SIMD; (2) begin with SIMD and end with SPMD; (3) begin with SPMD and end with SIMD; and (4) begin with SPMD and end with SPMD.

Recall from Section 2 that cases (2) and (3) are possible with the root node only. For case (1), it is shown below that the series can be modeled as a single code block (in SIMD mode). This result enables cases (2) and (3) to be modeled as a series of two blocks and case (4) to be modeled as a series of three blocks. For case (1), it will be shown next how a three-block series can be modeled as a single code block, and the case of more than three blocks follows inductively.

Let $I_{\text{to–SPMD}, e}$ and $P_{\text{to–SIMD}, e}$ denote the random variable whose values represent the mode-switching times from SIMD to SPMD and from SPMD to SIMD, respectively, with $e$ enabled PEs. In some machines, e.g., PASM, these correspond to the times required to execute a branching instruction and are therefore constants. For $e = 0$, both mode-switching times are assumed to be zero, i.e., $f_{P_{\text{to–SIMD}, 0}}(\cdot) = f_{I_{\text{to–SPMD}, 0}}(\cdot) = \delta(\cdot)$. Consider a series of three blocks labeled 0, 1, and 2. Assume that block 0 and block 2 are executed in SIMD mode, and block 1 is executed in SPMD mode. Although the PEs can execute block 1 in an asynchronous fashion, they must begin execution at the same time because block 0 is in SIMD mode, and they must wait for the last PE to finish before they begin to execute block 2, which is executed in SIMD mode. Thus, the three blocks can be modeled as an SIMD block $B$ whose execution time can be calculated as follows for each $e$, $0 \le e \le N$:

$$I_{B, e} = I_{0, e} + I_{\text{to–SPMD}, e} + P_{1, e} + P_{\text{to–SIMD}, e} + I_{2, e}$$

$$f_{I_{B, e}}(\cdot) = f_{I_{0, e}}(\cdot) * f_{I_{\text{to–SPMD}, e}}(\cdot) * f_{P_{1, e}}(\cdot) * f_{P_{\text{to–SIMD}, e}}(\cdot) * f_{I_{2, e}}(\cdot).$$

Inductively, a series of blocks that begins and ends with SIMD mode can be modeled as a single SIMD block by repeatedly applying this method and the methods from Sections 4 and 5. This is modeled as an SIMD block because the PEs must be synchronized at the end of this composite block.

From the above discussion, the following conclusions are made.

1. If the series begins and ends with SIMD blocks, then it can be modeled as a single SIMD (composite) block.
2. If the series begins with an SIMD block and ends with an SPMD block, then it can be modeled as a series consisting of an SIMD (composite) block followed by an SPMD (composite) block.
3. If the series begins with an SPMD block and ends with an SIMD block, then it can be modeled as a series consisting of an SPMD (composite) block followed by an SIMD (composite) block.
4. If the series begins with an SPMD block and ends with an SPMD block, then it can be modeled as either a single SPMD block (if all blocks in the original series are in SPMD mode), or a series of three (composite) blocks executed in the order of SPMD, SIMD, and SPMD.

For cases (2), (3), and (4), the beginning and/or the ending SPMD (composite) blocks in the resulting series are the equivalent of the longest subseries of SPMD blocks from the beginning and/or the end of the original series. For each of these cases, the series of composite blocks must begin and end with the same mode as the original series in order to be able to correctly merge with other nodes (due to synchronous nature of SIMD and asynchronous nature of SPMD).

*6.3. Pure Loop*

Recall from Section 2 that it is assumed that the different blocks of a loop body may use different modes of parallelism, but must be the same for all iterations of the loop, and each iteration of a loop must begin and end execution in the same mode. Therefore, the body of a mixed-mode loop contains a series of blocks that begins and ends with the same mode (the last block may contain only a mode switch instruction). From the discussion in the previous subsection, if it begins and ends with SIMD mode, then the loop body can be modeled as a single block and the single-mode technique of Subsection 5.4 can be applied to model the loop as a single SIMD block. Otherwise, the loop body begins and ends in SPMD mode and can be modeled as either a single SPMD block (if all blocks in the original series are in SPMD mode), or a series of three (composite) blocks executed in the order of SPMD, SIMD, and SPMD. This case will be considered in the remainder of this subsection.

Suppose the loop body is modeled as a series of three blocks labeled 0, 1, and 2, where block 0 and block 2 are (composite) SPMD blocks and block 1 is a (composite) SIMD block. As was defined earlier, random variables $\tilde{P}_0^j$ and $\tilde{P}_2^j$ represent the SPMD execution times of blocks 0 and 2 on PE $j$, respectively, and $I_{1,e}$ represents the SIMD execution time of block 1 with $e$ enabled PEs. Recall that $\tilde{R}_\ell^j$ is the random variable that corresponds to the number of iterations that PE $j$ is to execute the loop body, and $r_{\max} \geq 1$ denotes the maximum possible number of iterations to be executed by any PE.

Consider first the special case in which the number of iterations to be executed by each PE is the same (deterministic) value. Thus, for each PE $j$, $f_{\tilde{R}_\ell^j}(r_{\max}) = 1$, i.e., $f_{\tilde{R}_\ell^j}(r) = \delta(r - r_{\max})$, which indicates that each PE $j$ executes the loop body exactly $r_{\max}$ times. Conceptually unroll the loop body $r_{\max}$ times, which is a series of blocks in which the first and last blocks are executed in SPMD mode (see Fig. 4). Using the appropriate technique of the previous subsection, this unrolled series can be modeled as a series of three blocks. The first block is the SPMD block 0, the second block is a composite SIMD block for the subseries beginning with block 1 of iteration 1 and ending with block 1 of iteration $r_{\max}$, and the third block is the SPMD block 2. These are indicated by A, B, and C, respectively, in Fig. 4.

Consider next the general situation, i.e., $f_{\tilde{R}_\ell^j}(r_{\max}) \neq 1$. For this case, the number of iterations executed by each PE is generally distinct. If the next block after the loop is executed in SPMD mode, then it is conceptually possible for PEs to begin executing this block immediately after they complete the loop (because the last block of the loop is executed in SPMD mode). However, a large amount of bookkeeping overhead would be required to implement this feature on an actual mixed-mode machine (e.g., consider the situation in which a group of PEs that complete the loop are executing a SPMD block outside the loop and the other PEs are executing the SIMD block within the loop body). To avoid this, it is assumed that PEs are disabled as they complete the loop and that they remain disabled until all PEs finish the loop. This is straightforward to implement on a mixed-mode machine by having each PE disable itself when it has completed the loop, and using the CU to determine when all PEs that were initially enabled for the loop become disabled (which indicates that the loop has been completed by all initially enabled PEs).

Similar to the SIMD loop considered in Subsection 5.4, the CU must broadcast instructions for SIMD portions of block 1 for the PE(s) that executes the greatest number of iterations. As discussed above, the PEs that finish earlier are disabled until the entire loop is finished. An approach similar to that of Subsection 5.4 is used to track the number of enabled PEs at each iteration and to model the entire loop for each $e$, $0 \leq e \leq N$.

The loop is conceptually unrolled $r_{\max}$ times. This results in a series of mixed-mode blocks beginning with block 0 and ending with block 2, as shown in Fig. 4. For $0 \leq j < e$, the probability that any PE $j$ executes iteration $r$, denoted by $\rho_r$, is given by Eq. (1) in Subsection 5.4. Assume that there are $e$ enabled PEs at the beginning of iteration $r$. The combined mixed-mode execution time of iterations $r$ to $r_{\max}$ can be represented by a series of two blocks executed in SPMD and SIMD modes, respectively. The first block is block 0 and the second (composite SIMD) block models the series of blocks from block 1 of iteration $r$ through block 2 of iteration $r_{\max}$. The last SPMD block 2 is included in this composite SIMD block because of the assumed synchronization at the end of the loop.

Let $Y_{r,e}$ denote the random variable associated with the execution time of the SIMD block that models the series of blocks from block 1 of iteration $r$ through block 2 of $r_{\max}$ in the unrolled loop, given that $e$ PEs are enabled for iteration

| iteration no. | 1 | | | 2 | | | 3 | | | 4 | | | 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mode | P | I | P | P | I | P | P | I | P | P | I | P | P | I | P |
| block no. | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| same # iter. | A | B (SIMD) | | | | | | | | | | | | | C |
| diff. # iter. | D | E (SIMD) | | | | | | | | | | | | | |

**FIG. 4.** Conceptually unrolling a loop consisting of a series of three blocks with $r_{\max} = 5$ (P = SPMD, I = SIMD).

for $e \leftarrow 0$ to $N$

$$f_{Y_{r_{\max},e}}(\cdot) \leftarrow f_{I_{1,e}}(\cdot) * f_{I_{\text{to-SPMD},e}}(\cdot) * f_{P_{2,e}}(\cdot)$$

for $r \leftarrow r_{\max} - 1$ down to 1

for $e \leftarrow 0$ to $N$

$$f_{Y_{r,e}}(\cdot) \leftarrow f_{I_{1,e}}(\cdot) * f_{I_{\text{to-SPMD},e}}(\cdot) * \left[\sum_{h=0}^{e} \beta_{r,e}(h) \left(f_{V_{e,h}}(\cdot) * f_{P_{\text{to-SIMD},e}}(\cdot) * f_{Y_{r+1,h}}(\cdot)\right)\right]$$

**FIG. 5.** Algorithm for computing the density function of the composite SIMD block of a two-block equivalent series for a mixed-mode loop that starts and ends in SPMD.

$r$. Recall that the number of enabled PEs is nonincreasing as the number of iterations increases. For $r = r_{\max}$, $Y_{r_{\max}, e} = I_{1, e} + I_{\text{to–SPMD, e}} + P_{2, e}$. The algorithm described in Fig. 5 is used to compute the density functions of $Y_{r, e}$ based on the density functions of $Y_{r+1, h}$ $(0 \le h \le e)$ for all $e$, $0 \le e \le N$. The procedure is quite similar to that of Fig. 3. The main difference is in the combining of the SPMD block 2 of iteration $r$ with the SPMD block 0 of iteration $r + 1$. These two SPMD blocks cannot be combined in the usual way (as described in Subsection 4.2) because the number of enabled PEs for each block can be distinct. A temporary random variable $V_{e, h}$ is used to represent the execution time of these two blocks when $e$ PEs execute iteration $r$ and $h$ PEs execute iteration $r + 1$. Thus, if both blocks are executed by $e$ PEs (i.e., $h = e$), then $V_{e, h}$ is equal to the maximum of $\tilde{P}_2^j + \tilde{P}_0^j$ across $e$ PEs. However, if $h < e$ PEs are enabled for iteration $r + 1$, then $V_{e, h}$ is equal to the maximum of $\tilde{P}_2^j + \tilde{P}_0^j$ over $h$ PEs and the maximum of $\tilde{P}_2^j$ over $e - h$ PEs. Therefore, the formula for $V_{e, h}$ is

$$V_{e, h} = \begin{cases} \max\limits_{0 \le j < e} \{\tilde{P}_2^j + \tilde{P}_0^j\} & h = e,\ h > 0 \\ \max\{\max\limits_{0 \le j < h} \{\tilde{P}_2^j + \tilde{P}_0^j\}, & \\ \quad \max\limits_{0 \le j < e-h} \{\tilde{P}_2^j\}\} & 0 < h < e. \end{cases} \tag{2}$$

For $e = h = 0$, define $V_{0, 0} = 0$, i.e., $f_{V_{0, 0}} = \delta(\cdot)$.

To summarize, a two-block series is used to model the general case in which the loop body starts and ends in SPMD mode. The first block in this series, indicated by D in Fig. 4, is the SPMD block 0 of the loop body, $P_{0, e}$. The second block, indicated by E in Fig. 4, is the composite SIMD block $Y_{1, e}$, whose density function is computed according to Fig. 5.

### 6.4. Arbitrary Program Construct

As stated earlier, because all descendents of a data conditional construct must be executed in the same mode, each data conditional construct can be modeled as a single code block using the single-mode modeling techniques of Sections 4 and 5. It was shown in the previous subsections that a series of blocks or a pure loop can be modeled as a series of at most three blocks. As in the single-mode case, a divide-and-conquer algorithm can be used to calculate the execution time distribution of the whole program. Traversing the flow-analysis tree in depth-first order, each nonleaf node $n$ traversed can only have leaf nodes as its children. Therefore, each such nonleaf node (excluding the root node) corresponds to one of the following program structures: (1) a pure loop; (2) a pure data conditional construct; or (3) a series of blocks that is a clause of a data conditional construct.

Using the techniques introduced above, the subtree under node $n$ is modeled as a series of at most three leaf nodes (always a single leaf node for cases (2) and (3)). Node $n$ is then replaced with this series of leaf node(s). This procedure is repeated until the children of the root node are represented by a series of at most three composite leaf blocks. There are four possible cases.

1. The root is modeled as a single SIMD block, denoted by $R_0$, in which case the execution time is $I_{R_0, N}$.
2. The root is represented by a series of two blocks, $R_0$ and $R_1$, where $R_0$ is modeled as an SIMD block and $R_1$ is modeled as an SPMD block. The execution time of the program is $I_{R_0, N} + I_{\text{to–SPMD}, N} + P_{R_1, N}$.
3. The root is represented by a series of two blocks, $R_0$ and $R_1$, where $R_0$ is modeled as an SPMD block and $R_1$ is modeled as an SIMD block. The execution time of the program is $P_{R_0, N} + P_{\text{to–SIMD}, N} + I_{R_1, N}$.
4. The root is represented by a series of three blocks, $R_0$, $R_1$, and $R_2$, where $R_0$ is modeled as an SPMD block, $R_1$ is modeled as an SIMD block, and $R_2$ is modeled as an SPMD block. The execution time of the program is $P_{R_0, N} + P_{\text{to–SIMD}, N} + I_{R_1, N} + I_{\text{to–SPMD}, N} + P_{R_2, N}$.

## 7. NUMERICAL STUDIES

### 7.1. Hypothetical Mixed-Mode Example

To demonstrate how mode selections can affect the total execution time distribution, numerical parameters are associated with the nodes of the flow analysis tree of Fig. 2 to construct a very simple example. The hypothetical program is assumed to be executed on an 8-PE SIMD/SPMD mixed-mode machine. It is assumed that in each mode, the execution time for basic operations (i.e., operations that are not looping or conditional constructs) are constants (i.e., deterministic values). Therefore, the execution time of each original leaf block for each mode is deterministic. It is also assumed that the execution time of each basic operation is identical in SIMD and SPMD modes except for inter-PE communication operations, in which case SIMD mode is assumed to execute faster than SPMD mode. Only `blk_f` is assumed to contain inter-PE communications; thus it executes faster in SIMD mode than in SPMD mode. Execution times of each block are listed in Table I. Simplifying assumptions are made for the "overhead" operations listed; the framework developed here can readily support distinct values for each overhead operation in each mode. For each PE, the loop is assumed to execute 8, 9, 10, 11, or 12 iterations with equal probability (i.e., for all $r$, $8 \le r \le 12$, $f_{\tilde{R}_\ell^j}(r) = 0.2$),

**TABLE I**
**Assumed Execution Times of Each Block in SIMD and SPMD Modes for the Flow Analysis Tree of Fig. 2**

| Mode | `blk_a` | `blk_b` | `blk_c` | `blk_d` | `blk_e` | `blk_f` | Overhead[a] |
|---|---|---|---|---|---|---|---|
| SIMD | 12 | 15 | 10 | 29 | 23 | 10 | 1 |
| SPMD | 12 | 15 | 10 | 29 | 23 | 35 | 1 |

[a]Simplifying assumption for overhead for: mode switching, `for_init`, `for_test`, `if_test`, `post_then`, and `post_else`.

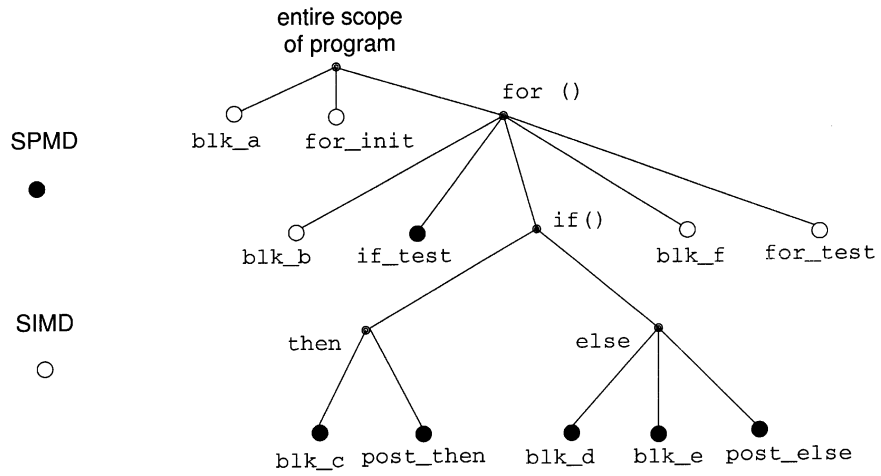


**FIG. 6.** Mixed-mode execution for the numerical example.

and the data conditional statement is assumed to execute the "then" clause with probability 0.8 for each PE (i.e., $p_t^j = 0.8$).

The execution time density function of the whole program is computed for three cases: executing the entire program in SIMD mode, executing the entire program in SPMD mode, and executing the program using a particular mixed-mode scheme, shown in Fig. 6. For the mixed-mode scheme considered, the `if` statement (and its descendents) and the `if_test` are executed in SPMD mode, and the rest of the program is executed in SIMD mode. The expected values of the resulting distributions for program execution time are listed in the first row of Table II. These values were computed after evaluating the density function for each case considered. Based on these computed expected values for this example's assumed numerical parameters, SPMD's advantages associated with effectively executing the data conditional construct and juxtaposing SPMD blocks without interblock synchronization exceed its disadvantage for inter-PE communication time (compared with SIMD and mixed-mode).

To illustrate the importance of having accurate execution time predictions for making proper mode selections, approximations for expected execution times for the above cases are also computed using a simple "average value approach." In the average value approach, the flow-analysis tree is traversed in a depth-first order, as in the proposed approach. However, instead of modeling each pruned portion of the three with the appropriate random variable(s), an approximate expected value is used. For looping constructs, the number of iterations executed (for both SIMD and SPMD modes) is approximated by the expected number of iterations to be executed by each PE. Thus, the looping construct of the numerical example is assumed to execute exactly 10 iterations. For SPMD execution of a subtree, the expected execution time of a single PE is used to approximate the expected execution time across all PEs. This approximation ignores the aspect of determining the maximum time across all PEs caused by a synchronization before switching to SIMD mode and upon program completion in SPMD mode (such as described in Fig. 1).

Applying the average value approach to the three cases considered for the numerical example requires the following calculations. For SPMD execution of the entire program, the execution time of the `if` statement (and its descendents) is $(10 + 1) \times 0.8 + (29 + 23 + 1) \times 0.2 = 19.4$, the execution time of the loop is $(15 + 1 + 19.4 + 35 + 1) \times 10 = 714$, and the total program execution time is $12 + 1 + 714 = 727.0$. For SIMD execution of the entire program, the execution time of the `if` statement (and its descendents) is $(10 + 1) \times 0.8^8 + (29 + 23 + 1) \times 0.2^8 + (10 + 1 + 29 + 23 + 1) \times (1 - 0.8^8 - 0.2^8) = 55.11$, the execution time of the loop is $(15 + 1 + 55.11 + 10 + 1) \times 10 = 821.1$, and the total execution time is $12 + 1 + 821.1 = 834.1$. For the mixed-mode case shown in Fig. 6, the execution time of the `if` statement (and its descendents) is $(10 + 1) \times 0.8 + (29 + 23 + 1) \times 0.2 = 19.4$, the execution time of the loop is $(15 + 1 + 1 + 19.4 + 1 + 10 + 1) \times 10 = 484$, and the total execution time is $12 + 1 + 484 = 497.0$. These results are tabulated in the second row of Table II.

From Table II, note that the *actual* expected values (computed using the proposed approach) are significantly different from the corresponding *approximate* expected values (computed using the average value approach). Based on the average value approach, the mixed-mode case has an expected execution time that is significantly better than both the SIMD and SPMD executions of the entire program. However, the

**TABLE II**
**Actual and Approximate Expected Values for Execution Time of the Whole Program in SIMD, SPMD, and According to the Mixed-Mode Scheme of Fig. 6**

| Expected value of execution time | SIMD | SPMD | Mixed-mode |
|---|---|---|---|
| Actual (proposed approach) | 1002.0 | 889.4 | 915.8 |
| Approximate (avg. value approach) | 834.1 | 727.0 | 497.0 |

actual expected values indicate that SPMD execution of the entire program is the best choice of the three cases considered for the given numerical values and given mixed-mode choices (other numerical values and mixed-mode choices may result in mixed-mode being the best method).

One source of inaccuracy associated with the average value approach for the mixed-mode and SPMD calculations is due to using the expected execution time of a single PE to approximate the expected execution time across all PEs. As mentioned earlier, this approximation ignores the aspect of determining the maximum time across all PEs caused by a synchronization before switching to SIMD mode and upon program completion in SPMD mode. For the mixed-mode case considered, the ignored synchronization time was especially significant because it occurred within a looping construct (i.e., the error associated with the approximation was compounded multiple times). Another source of inaccuracy associated with the average value approach for the SIMD and mixed-mode cases is due to the approximation used for the number of iterations for the looping construct. In particular, for SIMD execution of the looping construct, the CU must broadcast instructions for the PE(s) that executes the largest number of iterations. Thus, the expected number of iterations for the SIMD loop is actually higher than the approximate value used, which represents the expected number of iterations for which at least a single PE is enabled during SIMD execution of the loop.

### *7.2. An Application Case Study*

This subsection illustrates the result of applying the proposed methodology for predicting the performance of an SPMD program for solving satisfiability problems [Li96]. The satisfiability problem (*SAT*) is a textbook example of an NP-complete problem [CoL90, KuG94]. Given a boolean expression, the goal of SAT is to determine whether there exist values for the variables of the expression that cause the expression to evaluate to TRUE (i.e., logical 1). For algorithmic simplicity, and without loss of generality, only instances of SAT expressed in conjunctive normal form (*CNF*) were considered (CNF is the AND of clauses, where each clause is the OR of one or more literals). CNFs with exactly three literals per clause were used (called *3-SAT*) in this study [KuG94]. Also, only unsatisfiable instances were considered in order to isolate the source of uncertainty in the parallel execution time.

The Davis–Putnam algorithm [DaP60] was adopted for solving 3-SATs in this study. This algorithm uses a depth-first traversal of a binary search tree in which a node at level $i$ represents a permutation of possible values for the first $i$ variables. Thus, the complete search tree has a number of levels equal to the number of variables in the expression.

Each nonleaf node in the search tree represents a partial assignment of the values of the variables, and the leaf nodes represent a complete assignment of values to variables. The algorithm operates by determining whether a (partial) assignment associated with a given node being searched evaluates to 1. If it does, then the algorithm halts and declares the expression as satisfiable. If the variable assignments for the node cause the expression to evaluate to 0, then the algorithm backtracks. Finally, if the value of the expression cannot be determined based on the partial assignment of the node, then the algorithm searches deeper into the tree. Because the time required to search any single (arbitrary) node of the tree is approximately the same constant, the number of nodes searched in the search tree was used as the basic measure of execution time. The total number of nodes searched for a given instance of the problem is not known a priori; it depends on the particular CNF expression.

Search algorithms such as the Davis–Putnam algorithm are often implemented using a recursive function. However, because the execution time of programs with a recursive function was not analyzed in this paper, the algorithm was instead implemented with only data conditional and looping constructs, as shown in Fig. 7. The flow-analysis tree of Fig. 7 is for solving a 3-SAT problem with 12 variables. Because program performance is measured by the number of nodes searched in the binary tree, the control overhead for loops

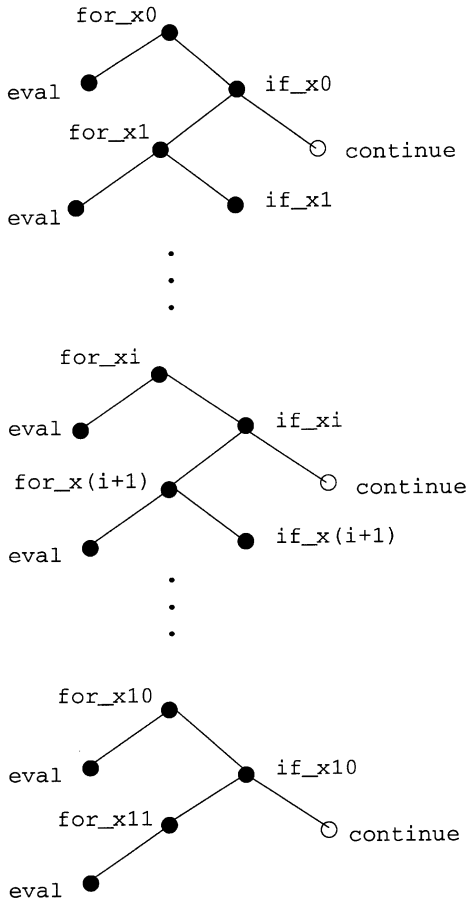


**FIG. 7.** The flow-analysis tree associated with the sequential program for solving 3-SAT with 12 variables.

and data conditionals are not shown in the figure. For each $i$, $0 \le i \le 11$, boolean variable $x_i$ is associated with a looping construct `for_xi` that executes two iterations; $x_i$ is assigned to 1 and 0 in the first and second iteration, respectively. The body of loop `for_xi` begins with an "eval" node, in which the formula of the instance is evaluated based on the current partial assignment of values to $x_0$ through $x_i$ ($x_{i+1}$ through $x_{11}$ are treated as unknowns). For $i < 11$, the loop body of `for_xi` also includes a data conditional construct `if_xi`, such that if the formula is undetermined (i.e., cannot be evaluated to 0), then loop `for_x(i+1)` is invoked to extend the current partial assignment to $x_{i+1}$ (this corresponds to the "then" clause of `if_xi`). Otherwise, the SAT returns FALSE (it cannot be TRUE because the instances considered are unsatisfiable), and the algorithm backtracks. Thus, the "else" clause of `if_xi` is a null node, labeled "`continue`" in the figure.

A parallel version of the Davis–Putnam algorithm was implemented on four PEs of an Intel Paragon as an SPMD program, in which the search tree was partitioned statically and distributed to four PEs, labeled 0 to 3. For each PE, a search was performed with fixed values assigned for the two variables $x_0$ and $x_1$ according to the two least significant bits of the PE label.

A set of 64,000 randomly generated unsatisfiable 3-SAT instances were used in this study. The elements of uncertainty in this program that affect the number of nodes searched are the branch probabilities associated with the data conditional constructs. The program was instrumented to record branching decisions. These empirically determined parameters were used to calculate the predicted distribution of execution time (using the approach of Section 4). The purpose of the instrumentation was to obtain a precise set of parameters (in this case, branching probabilities) in order to demonstrate the accuracy of the proposed analytical approach. Also, the application programmer could provide these parameters (or estimates of these parameters) based on knowledge about and experience with the program under consideration. Preliminary studies concerning the effect of using estimates for these branching probabilities (instead of precise values) are presented in [Li96]. In practice, in application areas such as image processing, the characteristics of the input, e.g., sequence satellite images, may be similar. Hence, information about program behavior for a given set of images can be used in predicting future behavior for other images with similar characteristics. In addition, by collecting conditional probabilities on a given machine, these parameters can then be used as input to the proposed analysis approach to predict performance for hypothetical machines with different properties (e.g., different basic operation times, different number of PEs, and/or different architecture).

The predicted distribution (based on using the proposed approach) and sample distribution (based on the actual number of nodes searched) are compared in Fig. 8. The predicted distribution was determined by applying the approach of Section 4 based on the empirically collected branching probabilities and the structure of the algorithm's flow-analysis tree (shown in Fig. 7).

It should be noted that the predicted and sample distributions would coincide (exactly) under ideal conditions, in which the number of nodes searched for different iterations of a loop were independent; the branching probabilities associated with different data conditional constructs on the same PE were independent; and the branching probabilities associated with the same data conditional construct across all PEs were independent. By comparing the predicted and sample distributions, it must be the case that these assumptions are not completely satisfied in this application, and thus some error exists.

Consider the particular assumption that the branching probabilities associated with the same conditional construct are independent across all PEs. To understand why this assumption is not completely satisfied for the parallel implementation of the Davis–Putnam algorithm, recall that the initial formulas assigned to PE 0 and PE 1 are nearly identical. In particular, in the formula for PE 0, variables $x_0$ and $x_1$ are both set to zero; in the formula for PE 1, variable $x_0$ is set to zero and $x_1$ is set to one. The other parts of these two formulas (i.e., the initially unassigned variables) are identical. So, it is likely that early assignments of variables (corresponding to upper portions in the flow-analysis tree) will often result in the same decision for these two PEs (because the initial formulas are nearly the same). This implies that there is actually a correlation among branching decisions across PEs, and thus they are not independent. For more details and descriptions of other studies, refer to [Li96].

Based on the conducted studies, it can be concluded that the proposed approach produced good predictions. To further illustrate the merit of the proposed approach, the straightforward "average value approach" (used for comparison with the proposed approach in Subsection 7.1) was also applied to the four-PE study. This average value approach, which can only estimate the average number of nodes searched, resulted in a predicted average number of nodes searched of 75.25. This is significantly different from the predicted average of the proposed approach, which was 103.67, and the actual (sample) average, which was 112.51. The fundamental reason for the poor performance of the average value approach is that the execution time of an SPMD program requires the maximum of the execution time across all PEs, and the average value approach cannot account for the effect of this "max" operation.

## 8. SUMMARY

A methodology was introduced for estimating the execution time distribution for a given data parallel program that is to be executed in an SIMD/SPMD mixed-mode heterogeneous environment. A block-based approach was used to transform the application program into a flow-analysis tree in which the internal nodes represent control and data conditional constructs and the leaf nodes represent basic code blocks.

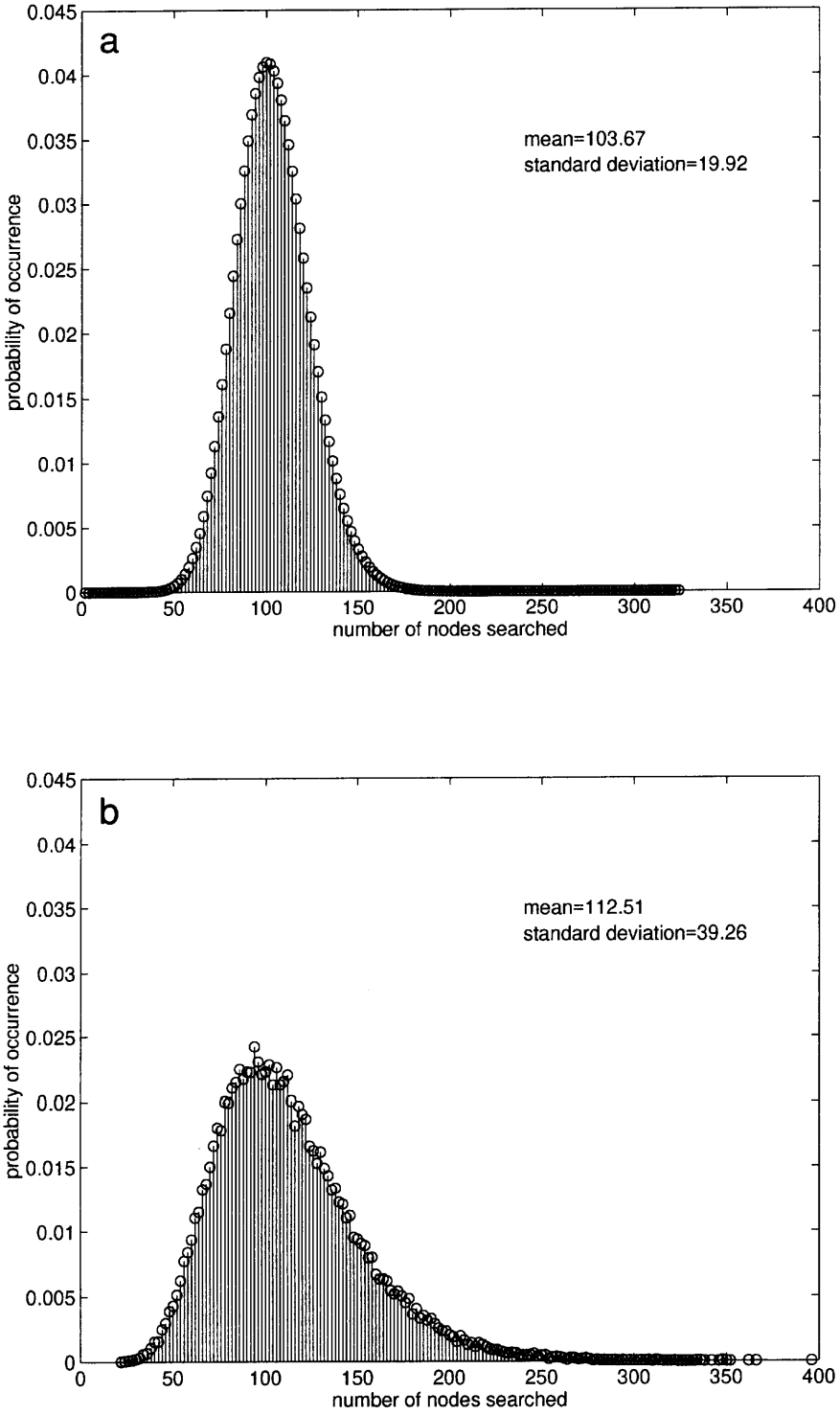


**FIG. 8.** Predicted versus sample distribution of number of nodes searched by the 4-PE SPMD program. (a) Predicted distribution from proposed approach. (b) Sample distribution based on 64,000 executions.

Given the mode in which each node in the flow-analysis tree is to be executed (SIMD or SPMD), the execution time distribution for each operation for both SIMD and SPMD modes, and appropriate probabilistic models for control and data conditional constructs, the methodology computes the execution time distribution for the program.

For SPMD mode, synchronization among PEs is currently limited to be among all PEs. Synchronization among a subset of PEs is important in many practical applications in which two or more PEs communicate. Thus, developing a precise model for arbitrary synchronization patterns is an area that needs further study.

Much of the developed methodology is directly applicable for mixed-machine systems consisting of SIMD, MIMD, and/or mixed-mode machines. Extensions of the proposed framework to compute execution time distributions for mixed-machine systems (i.e., suites of interconnected heterogeneous parallel machines) are presented in [LiA97]. One important issue that needs to be addressed in predicting the overall execution time in mixed-machine systems is modeling the time required for intermachine communication.

In summary, the precise quantification of how input data values affect program execution time in a heterogeneous computing environment, based on modeling the uncertainty with probabilistic distributions, is a primary contribution of the paper. The advantage—in terms of making effective mode selections—of using the proposed methodology to predict the distribution for the execution time of a program, over a simple "average value approach," was illustrated with the numerical examples.

## APPENDIX: PROBABILITY THEORY AND NOTATION

The purpose of this Appendix is to provide an overview of relevant concepts and notation from basic probability theory. Additional definitions and derivations can be found in textbooks on the subject (e.g., [MoG74]).

A *sample space* is the collection of all possible outcomes of a conceptual experiment. An *event* is a subset of the sample space. A *probability function*, denoted by $\Pr[\cdot]$, is a function that maps each event to a real number in [0, 1], which represents the likelihood that a given event occurs. All possible execution times for an SIMD/SPMD program represent events within a sample space.

A *random variable*, denoted by $X$ or $X(\cdot)$, is a function that maps each event to a real number. For instance, suppose the execution time of a program takes on one of several possible values. The random variable $X$ is used to map the event "program execution time = $x$ seconds" to the real number $x$. A random variable $X$ is *discrete* if the range of $X$ is countable. Throughout this paper, only discrete random variables are used, and the discrete values from the range of the random variable $X$ is $x_i$, $i \geq 0$. Let $X = x_i$ denote the event that is mapped to the value $x_i$. Let $X \leq x_i$ denote the union of all events that are mapped to values less than or equal to the value $x_i$.

The *density function* of $X$ is

$$f_X(x) = \begin{cases} \Pr[X = x_i], & \text{if } x = x_i,\ i = 0,\ 1,\ \cdots \\ 0, & \text{otherwise.} \end{cases}$$

The *distribution function* of $X$ is

$$F_X(x) = \Pr[X \leq x].$$

Three basic properties of a distribution function are: (1) $F_X(-\infty) = 0$, (2) $F_X(\infty) = 1$, and (3) $\forall x_j \geq x_i$, $F_X(x_j) \geq F_X(x_i)$. Also, as shown below, the distribution function can be derived from the density function and vice versa:

$$F_X(x) = \sum_{\{i:\, x_i \leq x\}} f_X(x_i)$$

$$f_X(x_i) = \begin{cases} F_X(x_i) - F_X(x_{i-1}) & i \geq 1 \\ F_X(x_0) & i = 0. \end{cases}$$

Consider two random variables $X_0$ and $X_1$, and let $X_0 = x_{0,j}$ and $X_1 = x_{1,k}$ denote arbitrary events associated with these random variables. The random variables $X_0$ and $X_1$ are defined to be *independent* if and only if for all $x_{0,j}$ and $x_{1,k}$

$$\Pr[X_0 = x_{0,j} \cap X_1 = x_{1,k}] = \Pr[X_0 = x_{0,j}] \Pr[X_1 = x_{1,k}].$$

Let $X_0, X_1, \cdots, X_{k-1}$ be a collection of $k$ independent random variables defined on the same probability space and assume that the range for each of these random variables is a subset of $\{\Delta \times i:\ i = 0,\ 1,\ \cdots\}$, for some real constant $\Delta$. Without loss of generality, $\Delta = 1$ is assumed in this paper.

Let the random variable $Y$ denote the sum of the independent random variables $X_0, X_1, \cdots, X_{k-1}$, i.e., $Y = \sum_{i=0}^{k-1} X_i$. For $k = 2$, the density function of $Y$ is the *convolution* of $f_{X_0}(\cdot)$ and $f_{X_1}(\cdot)$, denoted by $f_Y(\cdot) = f_{X_0}(\cdot) * f_{X_1}(\cdot)$, which is defined by

$$f_Y(j) = \sum_{i=0}^{\infty} f_{X_0}(j - i)\, f_{X_1}(i), \qquad j = 0,\ 1,\ \cdots$$

In general, the density function of $Y$ is given by

$$f_Y(\cdot) = f_{X_0}(\cdot) * f_{X_1}(\cdot) * \cdots * f_{X_{k-1}}(\cdot). \tag{3}$$

Let the random variable $Z$ denote the maximum over the set of independent random variables $X_0, X_1, \cdots, X_{k-1}$, i.e., $Z = \max\{X_0, X_1, \cdots, X_{k-1}\}$. Because of the independence assumption, the distribution of $Z$ is derived as

$$F_Z(i) = \Pr[Z \leq i] = \Pr[X_0 \leq i] \Pr[X_1 \leq i] \cdots \Pr[X_{k-1} \leq i].$$

Therefore,

$$F_Z(i) = F_{X_0}(i) F_{X_1}(i) \cdots F_{X_{k-1}}(i). \tag{4}$$

## ACKNOWLEDGMENT

A preliminary version of portions of this material was presented at the 4th Heterogeneous Computing Workshop, April 1995.

## REFERENCES


[AhS86] A. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

[BeK91] T. B. Berg, S.-D. Kim, and H. J. Siegel. "Limitations Imposed on Mixed-Mode Performance of Optimized Phases Due to Temporal Juxtaposition." *J. Parallel Distrib. Comput.* **13,** 2 (Oct. 1991), 154– 169.

[CaK88] T. L. Casavant and J. G. Kuhl. "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems." *IEEE Trans. Software Engrg.* **14,** 2 (Feb. 1988), 141–154.

[ChE93] S. Chen, M. M. Eshaghian, A. Khokhar, and M. E. Shaaban. "A Selection Theory and Methodology for Heterogeneous Supercomputing." *Proceedings of the Workshop on Heterogeneous Processing, Apr. 1993,* pp. 15–22.

[CoL90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[DaG88] F. Darema, D. A. George, V. A. Norton, and G. F. Pfister. "A Single-Program-Multiple-Data Computational Model for EPEX/ FORTRAN." *Parallel Comput.* **7** (Apr. 1988), 11– 24.

[DaP60] M. Davis and H. Putnam. "A Computing Procedure for Qualification Theory." *J. Assoc. Comput. Mach.***7** (1960), 201–215.

[DuB88] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon. "Image Processing on a SIMD/SPMD Architecture: OPSILA." *Proceedings of the 9th International Conference on Pattern Recognition, Nov. 1988,* pp. 14–17.

[FiC88] S. A. Fineberg, T. L. Casavant, T. Schwederski, and H. J. Siegel. "Non-Deterministic Instruction Time Experiments on the PASM System Prototype." *Proceedings of the 1988 International Conference on Parallel Processing, Aug. 1988,* pp. 444–451.

[Fly66] M. J. Flynn. "Very High-Speed Computing Systems." *Proc. IEEE* **54,** 12 (Dec. 1966), 1901– 1909.

[Fre89] R. F. Freund. "Optimal Selection Theory for Superconcurrency.'' *Proceedings of Supercomputing '89, Nov. 1989,* pp. 47–50.

[ICE95] Integrated Computing Engines, Inc. *The MeshSP.* Technical Report, ICE, Waltham, MA, July 1995.

[Kog94] P. M. Kogge, "EXECUBE —A New Architecture for Scalable MPPs." *Proceedings of the 1994 International Conference on Parallel Processing, Aug. 1994,* Vol. 1, pp. 77–84.

[KuG94] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing.* Benjamin/Cummings, Redwood City, CA, 1994.

[Li96] Y. A. Li. *A Probabilistic Framework for Estimation of Execution Time in Heterogeneous Computing Systems,* Ph.D. thesis, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, Aug. 1996.

[LiA97] Y. A. Li and J. K. Antonio. "Estimating the Execution Time Distribution for a Task Graph in a Heterogeneous Computing System." *Proceedings of the Heterogeneous Computing Workshop HCW 97, Apr. 1997,* pp. 172–184.

[LiM87] G. J. Lipovski and M. Malek. *Parallel Computing: Theory and Comparisons.* Wiley, New York, 1987.

[MoG74] A. M. Mood, F. A. Graybill, and D. C. Boes. *Introduction to the Theory of Statistics.* McGraw-Hill, New York, 1974.

[NiH81] L. M. Ni and K. Hwang. "Optimal Load Balancing Strategies for a Multiple Processor System." *Proceedings of the 1981 International Conference on Parallel Processing, Aug. 1981,* pp. 352–357.

[NiS93] M. A. Nichols, H. J. Siegel, and H. G. Dietz. "Data Management and Control-Flow Aspects of an SIMD/SPMD Parallel Language/Compiler." *IEEE Trans. Parallel Distrib. Systems* **4,** 2 (Feb. 1993), 222–234.

[NoT93] M. G. Norman and P. Thanisch. "Models of Machines and Computation for Mapping in Multicomputers." *ACM Comput. Surveys* **25,** 3 (Sept. 1993), 263– 302. [Also University of Edingburgh Technical Report TR-91-14]

[PhW93] M. Philippsen, T. Warschko, W. F. Tichy, and C. Herter. "Project Triton: Towards Improved Programmability of Parallel Machines." *Proceedings of the 26th Hawaii International Conference on System Sciences, Jan. 1993,* pp. 192–201.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li. "Heterogeneous Computing." In A. Y. Zomaya (Ed.). *Parallel and Distributed Computing Handbook.* McGraw-Hill, New York, 1996, pp. 725 –761.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio. "Software Support for Heterogeneous Computing." In J. Allen and B. Tucker (Eds.). *The Computer Science and Engineering Handbook.* CRC Press, Boca Raton, FL, 1997, pp. 1886– 1909.

[SiM96] H. J. Siegel, M. Maheswaran, D. W. Watson, J. K. Antonio, and M. J. Atallah. "Mixed-Mode System Heterogeneous Computing." In M. M. Eshaghian (Ed.). *Heterogeneous Computing.* Artech House, Norwood, MA, 1996, pp. 19 –65.

[SiS96] H. J. Siegel, T. Schwederski, W. G. Nation, J. B. Armstrong, L. Wang, J. T. Kuehn, R. Gupta, M. D. Allemang, D. G. Meyer, and D. W. Watson. "The Design and Prototyping of the PASM Reconfigurable Parallel Processing System." In A. Y. Zomaya (Ed.). *Parallel Computing: Paradigms and Applications.* International Thomson Computer Press, London, 1996, pp. 78–114.

[WaA94] D. W. Watson, J. K. Antonio, H. J. Siegel, and M. J. Atallah. "Static Program Decomposition Among Machines in an SIMD/ SPMD Heterogeneous Environment with Non-Constant Mode Switching Costs." *Proceedings of the Heterogeneous Computing Workshop HCW 94, Apr. 1994,* pp. 58 –65.

[WaS94] D. W. Watson, H. J. Siegel, J. K. Antonio, M. A. Nichols, and M. J. Atallah. "A Block-Based Mode Selection Model for SIMD/ SPMD Parallel Environments." *J. Parallel Distrib. Comput.* **21,** 3 (June 1994), 271–288.

[WeW94] C. C. Weems, G. E. Weaver, and S. G. Dropsho. "Linguistic Support for Heterogeneous Parallel Processing: A Survey and an Approach.'' *Proceedings of the Heterogeneous Computing Workshop HCW 94, Apr. 1994,* pp. 81 –88.


---


YAN A. LI received his B.E. degree from Tsinghua University, Beijing, China, in 1991, and his M.S.E.E. and Ph.D. degrees, both from Purdue University, West Lafayette, IN, in 1993 and 1996, respectively. He is currently a senior system architect at Intel Corporation. He is a member of IEEE and Eta Kappa Nu. His major research interest includes parallel processing, high-performance heterogeneous computing, computer architecture, and computer systems simulation.

JOHN K. ANTONIO received the B.S., M.S., and Ph.D. degrees in electrical engineering from Texas A&M University, College Station, TX. He currently holds the position of associate professor of Computer Science within the College of Engineering at Texas Tech University. Before joining Texas Tech, he was with the School of Electrical and Computer Engineering at Purdue University. His current research interests include heterogeneous systems, configuration techniques for embedded parallel systems, and computational aspects of control and optimization. He has co-authored over 50 publications in these and related areas. He is a member of the IEEE Computer Society and is also a member of the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honorary societies.

HOWARD JAY SIEGEL is a professor and coordinator of the Parallel Processing Laboratory in the School of Electrical and Computer Engineering at Purdue University. He received two B.S. degrees from MIT, and the M.A.,

M.S.E., and Ph.D. degrees from Princeton University. He has coauthored over 230 technical papers in the areas of interconnection networks, reconfigurable parallel systems (PASM), parallel algorithms, and heterogeneous computing. He is a Fellow of the IEEE, was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and served on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers.

MIN TAN is a Ph.D. candidate in the School of Electrical and Computer Engineering at Purdue University. He attended Shanghai Jiao Tong University, P.R. China, in 1988. He received a B.A. degree from Western Maryland College in 1993 and an M.S. degree in electrical engineering from Purdue University in 1994. His research interests include heterogeneous computing, data staging, and parallel applications. He has authored or coauthored 12 technical papers in these and related areas. Mr. Tan is a member of IEEE, the IEEE Computer Society, and the Eta Kappa Nu honorary society.

DANIEL W. WATSON is an assistant professor at the Department of Computer Science at Utah State University. He received the B.S. degree in electrical engineering at Tennessee Tech University in 1985 and received the M.S.E.E. and Ph.D. degrees from the School of Electrical Engineering at Purdue University in 1989 and 1993, respectively. He has coauthored numerous conference papers, journal articles, and book chapters on the topics of mixed-mode parallelism and heterogeneous computing. He has been the recipient of faculty fellowships at Phillips and NRaD research laboratories for his work in heterogeneous computing, and is a member of the IEEE Computer Society, Association for Computing Machinery, and the Gamma Beta Phi, Tau Beta Pi, and Eta Kappa Nu honorary societies.

Received April 17, 1995; revised April 14, 1997; accepted May 11, 1997