Symbolic Cost Estimation of Parallel Applications

Arjan J.C. van Gemund

Faculty of Information Technology and Systems
Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft, The Netherlands
a.j.c.vangemund@its.tudelft.nl

Abstract. Symbolic cost models are an important performance engineering tool because of their diagnostic value and their very low solution cost when the computation features regularity. However, especially for parallel applications their derivation, including the symbolic simplifications essential for low solution cost, is an effort-intensive and error-prone process. We present a tool that automatically compiles process-oriented performance simulation models into symbolic cost models that are symbolically simplified to achieve extremely low solution cost. As the simulation models are intuitively close to the parallel program and machine under study, derivation effort is significantly reduced. Apart from its use as a stand-alone tool, the compiler is also used within a symbolic cost estimator for data-parallel programs. With minimal program annotation by the user, symbolic cost models are automatically generated in a matter of seconds, while the evaluation time of the models ranges in the milliseconds. Experimental results on four data-parallel programs show that the average prediction error is less than 15 %. Apart from providing program scalability assessment, the models correctly predict the best design alternative in all cases.

1 Introduction

Symbolic cost modeling is a performance modeling technique where parallel programs are mapped into explicit, algebraic performance expressions that are parameterized in terms of, e.g., the number of processors, problem size, and machine computation and communication parameters. Symbolic models are analytical, providing diagnostic insight in the complex interplay between program and machine parameters. An important benefit of symbolic cost models is their potentially low solution cost, compared to, e.g., simulation [13], Petri nets [2], queuing networks [1,11], and process algebras [10]. As most parallel programs and machines employ some form of (sequential or parallel) replication, cost models typically have a regular structure. This regularity enables symbolic simplification that dramatically reduces solution complexity by many orders of magnitude. As interactive parameter experimentation during parallel program development requires cost models to evaluate in (milli)seconds rather than minutes or hours, this feature is extremely valuable.

Although attractive in terms of solution complexity, the derivation of low-complexity cost models from parallel programs is an effort-intensive and error-prone process, that is akin to manual complexity analysis of parallel algorithms [4,14]. Aimed to provide tool support in this derivation process, a performance simulation formalism called PAMELA

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 147–156.
© Springer-Verlag Berlin Heidelberg 2002

(PerformAnce ModEling LAnguage) has been presented [7]. Both the parallel program and machine are modeled in terms of PAMELA. Instead of simulation, the PAMELA model is mechanically mapped into a symbolic cost model that has the lowest possible solution complexity, while offering a prediction accuracy that is sufficient to discriminate between various program design alternatives. As the simulation language is intuitively close to the parallel program (and machine), cost model derivation efficiency is significantly improved. While previous publications presented the methodology and calculus, in this paper we present a language implementation, featuring a compiler that automatically compiles PAMELA models into symbolic cost models.

An interesting feature of the PAMELA methodology is that for data-parallel programs the mapping of programs to PAMELA models can also be mechanized. As part of a data-parallel compiler project, a so-called Modeling Engine has been built that automatically generates PAMELA models from data-parallel programs that have been annotated with missing information on, e.g., loop bounds and branch probabilities [9]. Combined with the PAMELA compiler, a data-parallel program can be subsequently *compiled* into a symbolic cost model within seconds, while the cost model predicts performance within milliseconds. In this paper we also describe the results of this automatic cost estimator. The accuracy of the generated cost models is more than sufficient to allow a correct ranking of various coding and/or data partitioning strategies.

The paper is organized as follows. In Section 2 we present our tool implementation based on the PAMELA methodology. In Section 3 we briefly describe how PAMELA models are generated from data-parallel programs. In Section 4 we demonstrate the utility of the symbolic cost estimation process in the design of four well-known numeric applications. In Section 5 we summarize our contributions.

2 Symbolic Cost Estimation

In this section we briefly summarize the PAMELA modeling language and the compiler. As the language is intuitively close to the description of algorithms, the use of PAMELA is more cost effective than manual oriented approaches which do not offer tool support. Apart from its use as a stand-alone tool, the PAMELA compiler is also used in conjunction with the Modeling Engine that generates PAMELA models from data-parallel programs.

2.1 Language

PAMELA is a process-algebraic language that allows a parallel program to be modeled in terms of a sequential, conditional, and parallel composition of processes that model workload. Work is described by the `use` *process*. The construct `use(r,t)` exclusively acquires service from *resource* `r` for `t` units time (excluding possible queuing delay). The scheduling policy that is currently supported is First-Come-First-Served (FCFS) with non-deterministic conflict arbitration. A resource `r` has a multiplicity that may be larger than unity. As in queuing networks, it is convenient to define an infinite-server resource called `rho` that has infinite multiplicity. Instead of writing `use(rho,t)` we will simply write `delay(t)`.

PAMELA features the following process composition operators: `;` for binary sequential composition, `seq (<index> = <lb>, <ub>)` for n-ary sequential composition, `||` for binary parallel composition, `par (<index> = <lb>, <ub>)` for n-ary parallel composition, `if (<cond>) [else]` for conditional composition.

PAMELA is a strongly typed language. Variables can be of three types: `process`, `resource`, or `numeric`. The latter type is used for time expressions, parameters, indices, and loop bounds. Each lhs variable can have a formal parameter list. The scope of these parameters is limited to the rhs expression.

The following PAMELA equations model the well-known Machine Repair Model (MRM) in which `P` clients either spend a mean time `t_l` on local processing, or request service from a single server `s` with service time `t_s` with a total cycle count of `N` iterations (unlike steady-state analysis, in our approach we require models to terminate; yet `N` may be symbolic).

```
numeric parameter P            % # clients
numeric parameter N            % # iterations
numeric t_l = 10               % local think time
numeric t_s = 0.1              % service time

resource parameter fcfs(i)     % predefined FCFS array
resource s = fcfs(0,1)         % s is FCFS type resource
                               % args: index, multiplicity

process main = par (p = 1, P)
                   seq (i = 1, N) {
                       delay(t_l) ;
                       use(s,t_s)
                   }
```

The example illustrates the high, "problem" level at which an application is modeled. In the PAMELA top-down modeling approach, problem parallelism is modeled explicitly in terms of `par` (or `;`) operators, to be constrained by mutual exclusion (`use`) when the processes are mapped onto a limited number of resources, such as software locks, file servers (cf. above example), (co)processors, communication links, memories, I/O disk handlers, etc. This high-level modeling approach allows virtually all parallel algorithms to be modeled in terms of *series-parallel* (SP) process expressions. This SP synchronization structure is essential to allow mechanization of the symbolic timing analysis. A lower, "implementation" level approach would require a message-passing paradigm which cannot be statically analyzed [8]. Also note, that due to the ultra-low solution complexity of the performance model high parameter values can be specified as the model typically evaluates in milliseconds.

As in ordinary mathematics, the semantics of a PAMELA model is based on expression substitution. Although for readability a model may be coded in terms of many equations, internally each expression is evaluated by recursively substituting every global variable by its corresponding rhs expression. Consequently, the above MRM model is internally rewritten to the equivalent normal-form model below (relevant equations shown only).

```
numeric parameter P
numeric parameter N
process main = par (p = 1, P)
                   seq (i = 1, N) {
                       delay(10) ;
                       use(fcfs(0,1),0.1)
                   }
```

where `s`, `t_l`, and `t_s` have been substituted. Note that the optional `parameter` modifier blocks this substitution process. In the above example this will cause `P` and `N` to appear as parameters in the eventual cost model.

Apart from the process operators mentioned above, PAMELA includes the usual numeric operators such as +, *, `mod`, `div`, ==, <, `max`, etc., as well as the reductions `sum (<index> = <lb>, <ub>)` and `max (<index> = <lb>, <ub>)`. Conditional numeric expressions are described using `if-then` just like conditional process expressions. Furthermore, as parts of the analysis result is expressed in terms of vectors, the `numeric` abstract data-type includes vectors as well as scalars, thus overloading all operators. A vector is denoted `[<scalar>, ..., <scalar>]` Hence, the expression `[1,2,3] * 4` is legal and, incidentally, will be compiled to `[4,8,12]` as a result of the compiler's internal numeric optimization engine. In order to generate unbounded, symbolic vectors PAMELA features the `unitvec` operator which returns a unit vector (base 0) in the dimension given by its argument. For instance, the expression `10 * unitvec(3)` will be compiled to `[0,0,0,10]`.

2.2 Compilation

A PAMELA model is translated to a time-domain performance model by substituting every `process` equation by a `numeric` equation that models the execution time associated with the original process. The lhs is derived from the original lhs by prefixing `T_`. Thus the cost model of a process expression `main` is denoted `T_main`. The result is a PAMELA model that only comprises `numeric` equations as the original `process` and `resource` equations are no longer relevant. The fact that the cost model is again a PAMELA model is for reasons of convenience as explained later on. In the following we briefly describe the translation process. A more detailed background can be found in [8].

The analytic approach underlying the translation process is based on critical path analysis of the delays due to condition synchronization [5,12,16] ("task synchronization"), combined with a lower bound approximation of the delays due to mutual exclusion synchronization ("queuing delay") as a result of resource contention [8]. In the following we assume a PAMELA model in which all expressions have already been substituted as the result of the normalization pass described earlier.

Per `process` equation four `numeric` equations are generated, whose lhs identifiers are derived from the original process variable by prefixing specific strings. Let `L` denote the lhs of a process equation. The first equation generated is `phi_L` which computes the condition synchronization delay by recursively applying the following transformation rules:

```
L = a ; b              -> phi_L = phi_a + phi_b
```

```
L = a || b            -> phi_L = max(phi_a,phi_b)
L = use(fcfs(a,b),t) -> phi_L = t / b
```

The second equation generated is `delta_L` which computes the mutual exclusion synchronization delay by

```
L = a ; b             -> delta_L = delta_a + delta_b
L = a || b            -> delta_L = max(delta_a,delta_b)
L = use(fcfs(a,b),t) -> delta_L = unitvec(a) * (t / b)
```

The delta vectors represent the aggregate workload per resource (index). The effective mutual exclusion delay is computed by the third equation, which is generated by the following transformation rule:

```
L = ...               -> omega_L = max(delta_L)
```

Finally, the execution time `T_L` is generated by the following transformation rules:

```
L = a ; b             -> T_L = T_a + T_b
L = a || b            -> T_L = max(max(T_a,T_b),omega_L)
L = use(fcfs(a,b),t) -> T_L = phi_L
```

The above `max(max(T_a,T_b),omega_L)` computation shows how each of the delays due to condition synchronization and mutual exclusion are combined in one execution time estimate that effectively constitutes a lower bound on the actual execution time. The recursive manner in which both delays are combined guarantees a bound that is the sharpest possible for an automatic symbolic estimation technique (discussed later on). Conditional composition is simply transferred from the process domain to the time domain, according to the transformation

```
L = if (c) a else b  -> X_L = if (c) X_a else X_b
```

where `X` stands for the `phi`, `delta`, `omega`, and `T` prefixes. The numeric condition, representing an average truth probability when embedded within a sequential loop, is subsequently reduced, based on the numeric (average truth) value of `c` according to

```
if (c) X_a else X_b  -> c * X_a + (1 - c) * X_b
```

An underlying probabilistic calculus is described in [6].

Returning to the MRM example, based on the above translation process the PAMELA model of the MRM is internally compiled to the following time domain model (`T_main` shown only):

```
numeric parameter P
numeric parameter N
numeric T_main = max(max (p = 1, P) {
                         sum (i = 1, N) {
                             10.1
                         }
                     },
                     max(sum (p = 1, P) {
                               sum (i = 1, N) {
                                   [ 0.1 ]
                               }
                         }))
```

Although this result is a symbolic cost model, evaluation of this model would be similar to simulation. Due to the regularity of the original (MRM) computation, however, this model is amenable to simplification, a crucial feature of our symbolic cost estimation approach. The simplification engine within the PAMELA compiler automatically yields the following cost model:

```
numeric parameter P
numeric parameter N
numeric T_main = max((N * 10.1),(P * (N * 0.1)))
```

which agrees with the result of bounding analysis in queueing theory (the steady-state solution is obtained by symbolically dividing by `N`). This result can be subsequently evaluated for different values of `P` and `N`, possibly using mathematical tools other than the PAMELA compiler. In PAMELA further evaluation is conveniently achieved by simply recompiling the above model after removing `parameter` modifiers while providing a numeric rhs expression. For example, the following instance

```
numeric P = 1000
numeric N = 1000000
numeric T_main = max((N * 10.1),(P * (N * 0.1)))
```

is compiled (i.e., evaluated) to

```
numeric T_main = 100000000
```

While the prediction error of the symbolic model compared to, e.g., simulation is zero for $P = 0$ and $P \to \infty$, near to the saturation point ($P = 100$) the error is around 8%. It is shown that for very large PAMELA models (involving $\mathcal{O}(1000+)$ resources) the worst case average error is limited to 50% [8]. However, these situations seldom occur as typically systems are either dominated by condition synchronization or mutual exclusion, in which case the approximation error is in the percent range [8].

Given the ultra-low solution complexity, the accuracy provided by the compiler is quite acceptable in scenarios where a user conducts, e.g., application scalability studies as a function of various machine parameters, to obtain an initial assessment of the parameter sensitivities of the application. This is shown by the results of Section 4. In particular, note that on a Pentium II 350 MHz the symbolic performance model of the MRM only requires 120 μs per evaluation (*irrespective* of N and P), while the evaluation of the original model, in constrast, would take approximately 112 Ks. The $\mathcal{O}(10^9)$ time reduction provides a compelling case for symbolic cost estimation.

3 Automatic Cost Estimation

In this section we describe an application of the PAMELA compiler within an automatic symbolic cost estimator for data-parallel programs. The tool has been developed as part of the JOSES project, a European Commission funded research project aimed at developing high-performance Java compilation technology for embedded (multi)processor systems [9]. The cost estimator is integrated as part of the Timber compiler [15], which compiles parallel programs written in Spar/Java (a Java dialect with data-parallel features similar to HPF) to distributed-memory systems. The cost estimator is based on a

combination of a so-called Modeling Engine and the PAMELA compiler. The Modeling Engine is a Timber compiler engine that generates a PAMELA model from a Spar/Java program. The PAMELA compiler subsequently compiles the PAMELA model to a symbolic cost model. While symbolic model compilation is automatic, PAMELA model generation by the Timber compiler cannot always be fully automatic, due to the undecidability problems inherent to static program analysis. This problem is solved by using simple compiler pragmas which enables the programmer to portably annotate the source program, supplying the compiler with the information required (e.g., branch probabilities, loop bounds). Experiments with a number of data-parallel programs show that only minimal user annotation is required in practice.

For all basic (virtual) machine operations such as +, ..., *, (computation), and = (local and global communication) specific PAMELA process calls are generated. During PAMELA compilation, each call is substituted by a corresponding PAMELA machine model that is part of a separate PAMELA source file that models the target machine. All parallel, sequential, and conditional control flow constructs are modeled in terms of similar PAMELA constructs, except unstructured statements such as `goto`, `break`, which cannot be modeled in PAMELA. In order to enable automatic PAMELA model generation, the following program annotations are supported: the `lower` and `upper` bound pragmas (when loop bounds cannot be symbolically determined at compile-time), the `cond` pragma (for data-dependent branch conditions), and the `cost` pragma (for assigning an entire, symbolic cost model for, e.g., some complicated sequential subsection).

A particular feature of the automatic cost estimator is the approach taken to modeling program parallelism. Instead of modeling the generated SPMD message-passing code, the modeling is based on the source code which is still expressed in terms of the original data-parallel programming model. Despite the fact that a number of low-level compiler-generated code features are therefore beyond the modeling scope, this high-level approach to modeling is essential to modeling correctness [9]. As a simple modeling example, let the vector `V` be cyclically partitioned over P processors. A (pseudo code) statement

```
forall (i = 1 .. N) V[i] = .. * ..;
```

will generate (if the compiler would use a simple owner-computes rule)

```
par (i = 1, N) { ... ; ... ; mult(i mod P) ; ... }
```

The PAMELA machine model includes a model for `mult` according to

```
resource cpu(p) = fcfs(p,1)
...
mult(p) = use(cpu(p),t_mult)
...
```

which models multiplication workload being charged to processor (index) `p`.

4 Experimental Results

In the following we apply the automatic cost estimator to four test codes, i.e., MATMUL (Matrix Multiplication), ADI (Alternate Implicit Integration), GAUSS (Gaussian

Elimination), and PSRS (Parallel Sorting by Regular Sampling). The actual application performance is measured on a 64 nodes partition of the DAS distributed-memory machine [3], of which a PAMELA machine model has been derived, based on simple computation and communication microbenchmarks [9]. In these microbenchmarks we measure local and global vector load and store operations at the Spar/Java level, while varying the access stride to account for cache effects. The first three regular applications did not require any annotation effort, while PSRS required 6 annotations.

The MATMUL experiment demonstrates the consistency of the prediction model for various N and P. MATMUL computes the product of $N \times N$ matrices A and B, yielding C. A is block-partitioned on the i axis, while B and C are block-partitioned on the j-axis. In order to minimize communication, the row of A involved in the computation of the row of C is assigned to a replicated vector (i.e., broadcast). The results for N = 256, 512, and 1,024 are shown in Figure 1. The prediction error is 5 % on average with a maximum of 7 %.

The ADI (horizontal phase) speedup prediction for a $1,024 \times 1,024$ matrix, shown in Figure 2, clearly distinguishes between the block partitioning on the j-axis (vertical) and the i-axis (horizontal). The prediction error of the vertical version for large P is caused by the fact that the PAMELA model generated by the compiler does not account for the loop overhead caused by the SPMD level processor ownership tests. The maximum prediction error is therefore 77 % but must be attributed to the current Modeling Engine, rather than the PAMELA method. The average prediction error is 15 %.

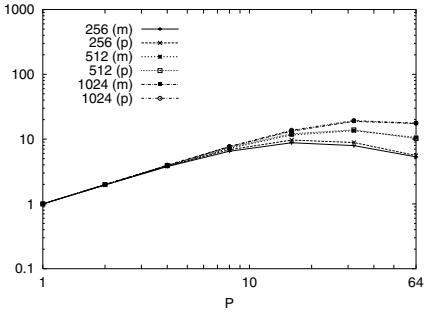

Fig. 1. MATMUL execution time [s] (N = 256, 512, and 1,024)

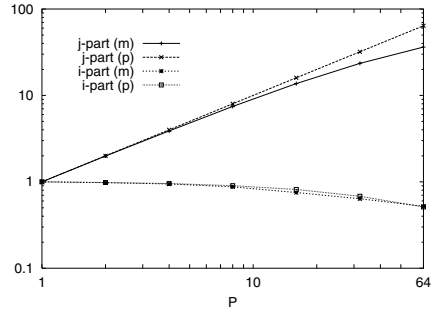

Fig. 2. ADI speedup (j and i-axis data partitioning)

The GAUSS application illustrates the use of the PAMELA model in predicting the difference between cyclic and block partitioning. The 512×512 matrix is partitioned on the j-axis. The submatrix update is coded in terms of a j loop, nested within an i loop, minimizing cache misses by keeping the matrix access stride as small as possible. The speedup predictions in Figure 3 clearly confirm the superior performance of block partitioning. For cyclic partitioning the access stride increases with P which causes delayed speedup due to increasing cache misses. The prediction error for large P is caused

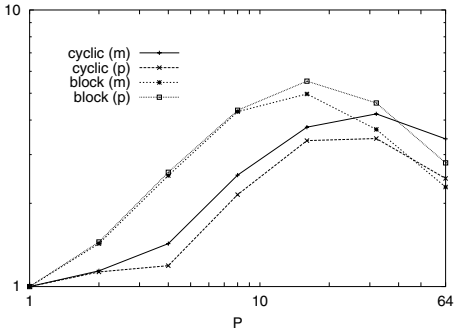

Fig. 3. GAUSS speedup (cyclic and block mapping)

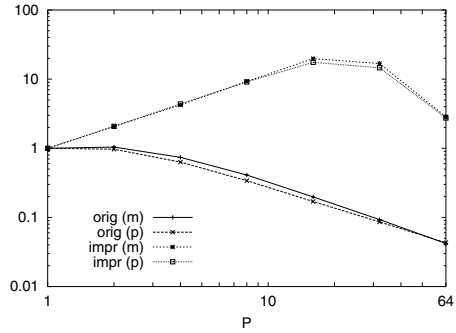

Fig. 4. PSRS speedup (original and improved data mapping)

by the fact that individual broadcasts partially overlap due to the use of asynchronous communication, which is not modeled by our PAMELA machine model. The prediction error is 13 % on average with a maximum of 35 %.

The PSRS application sorts a vector X of N elements into a result vector Y. The vectors X and Y are block-partitioned. Each X partition is sorted in parallel. Using a global set of pivots X is repartitioned into Y, after which each Y partition is sorted in parallel. Figure 4 shows the prediction results for $N = 819,200$ for two different data mapping strategies. Due to the dynamic, data-dependent nature of the PSRS algorithm, six simple loop and branching annotations were necessary. Most notably, the Quicksort procedure that is executed on each processor in parallel, required a few sequential profiling runs in order to enable modeling by the Modeling Engine. In the original program all arrays except X and Y are replicated (i.e., pivot vector and various index vectors). This causes a severe $\mathcal{O}(NP)$ communication bottleneck. In the improved program version this problem is solved by introducing a new index vector that is also partitioned. The prediction error is 12 % on average with a maximum of 26 %.

5 Conclusion

In this paper we present a tool that automatically compiles process-oriented performance simulation models (PAMELA models) into symbolic cost models that are symbolically simplified to achieve extremely low evaluation cost. As the simulation models are intuitively close to the parallel program and machine under study, the complex and error-prone effort of deriving symbolic cost models is significantly reduced. The PAMELA compiler is also used within a symbolic cost estimator for data-parallel programs. With minimal program annotation by the user, symbolic cost models are automatically generated in a matter of seconds, while the evaluation time of the models ranges in the milliseconds. For instance, the 300 s execution time of the initial PSRS code for 64 processors on the real parallel machine is predicted in less than 2 ms, whereas simulation would have taken over 32,000 s. Experimental results on four data-parallel programs show that the average error of the cost models is less than 15 %. Apart from providing a good scalability assessment, the best design choice is correctly predicted in all cases.

Acknowledgements

This research was supported in part by the European Commission under ESPRIT LTR grant 28198 (the JOSES project). The DAS I partition was kindly make available by the Dutch graduate school "Advance School for Computing and Imaging" (ASCI).

References

1. V.S. Adve, *Analyzing the Behavior and Performance of Parallel Programs*. PhD thesis, University of Wisconsin, Madison, WI, Dec. 1993. Tech. Rep. #1201.
2. M. Ajmone Marsan, G. Balbo and G. Conte, "A class of Generalized Stochastic Petri Nets for the performance analysis of multiprocessor systems," *ACM TrCS*, vol. 2, 1984, pp. 93–122.
3. H. Bal *et al.*, "The distributed ASCI supercomputer project," *Operating Systems Review*, vol. 34, Oct. 2000, pp. 76–96.
4. D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proc. 4th ACM SIGPLAN Symposium on PPoPP*, May 1993, pp. 1–12.
5. T. Fahringer, "Estimating and optimizing performance for parallel programs," *IEEE Computer*, Nov. 1995, pp. 47–56.
6. H. Gautama and A.J.C. van Gemund, "Static performance prediction of data-dependent programs," in *ACM Proc. on The Second International Workshop on Software and Performance (WOSP'00)*, Ottawa, ACM, Sept. 2000, pp. 216–226.
7. A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM Int'l Conf. on Supercomputing*, Tokyo, 1993, pp. 318–327.
8. A.J.C. van Gemund, *Performance Modeling of Parallel Systems*. PhD thesis, Delft University of Technology, The Netherlands, Apr. 1996.
9. A.J.C. van Gemund, "Automatic cost estimation of data parallel programs," Tech. Rep. 1-68340-44(2001)09, Delft University of Technology, The Netherlands, Oct. 2001.
10. N. Götz, U. Herzog and M. Rettelbach, "Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras," in *Proc. SIGMETRICS/PERFORMANCE'93*, LNCS 729, Springer, 1993.
11. H. Jonkers, A.J.C. van Gemund and G.L. Reijns, "A probabilistic approach to parallel system performance modelling," in *Proc. 28th HICSS, Vol. II*, IEEE, Jan. 1995, pp. 412–421.
12. C.L. Mendes and D.A. Reed, "Integrated compilation and scalability analysis for parallel systems," in *Proc. PACT '98*, Paris, Oct. 1998, pp. 385–392.
13. H. Schwetman, "Object-oriented simulation modeling with C++/CSIM17," in *Proc. 1995 Winter Simulation Conference*, 1995.
14. L. Valiant, "A bridging model for parallel computation," *CACM*, vol. 33, 1990, pp. 103–111.
15. C. van Reeuwijk, A.J.C. van Gemund and H.J. Sips, "Spar: A programming language for semi-automatic compilation of parallel programs," *Concurrency: Practice and Experience*, vol. 9, Nov. 1997, pp. 1193–1205.
16. K-Y. Wang, "Precise compile-time performance prediction for superscalar-based computers," in *Proc. ACM SIGPLAN PLDI'94*, Orlando, June 1994, pp. 73–84.