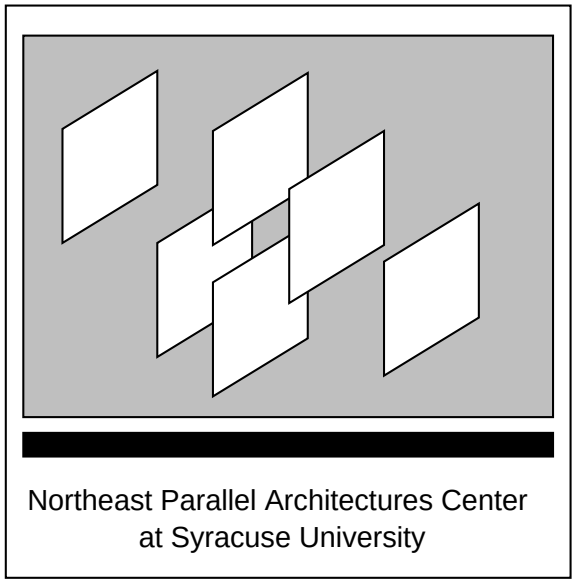
SCCS-453



An Integrated Software Development Model for Heterogeneous High Performance Computing

Manish Parashar, Salim Hariri, Tomasz Haupt and Geoffrey C. Fox
parashar@npac.syr.edu

April 21, 1993

SYRACUSE UNIVERSITY
Northeast Parallel Architectures Center
111 College Place, Room # 3-201 · Syracuse, New York 13244-4100
Tel: (315) 443-1722 · Fax: (315) 443-1973

An Integrated Software Development Model for Heterogeneous High Performance Computing

Manish Parashar, Salim Hariri, Tomasz Haupt and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
parashar@npac.syr.edu, hariri@cat.syr.edu

Contents

An Integrated Software Development Model for Heterogeneous High Performance Computing

Manish Parashar, Salim Hariri, Tomasz Haupt and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
parashar@npac.syr.edu, hariri@cat.syr.edu

Abstract

The last few decades have seen an impressive developments in every aspect of parallel computing technology; viz. processing and storage technology, interconnect technology and software technology. Although these systems incorporate large amount of computing power, they are not general enough to efficiently support today's computation-intensive problems (e.g. the Grand Challenges), that warrant multiple computational models and levels of parallelism. We believe that the future of parallel computing lies in the integration of the plethora of "specialized" architectures into a single Heterogeneous High Performance Computing (HHPC) environment that allows them to cooperate in solving complex problems. Software development in any Parallel/Distributed environment is a non-trivial process and requires a thorough understanding of the application and the architecture. This problem further intensifies as systems evolve into HHPC environments. The objective of this paper is to formalize the software development process for an HHPC environment. The issues and requirements that need to be addressed in HHPC software development are investigated and a model that meets these requirements is proposed. Support required at each stage of the model is also highlighted. The modeling of stock option pricing is used as a running example to validate the applicability of the model. Finally a survey of existing tools and techniques applicable to the different stages is presented.

1 Introduction

The last few decades have seen an impressive development in every aspect of parallel computing technology; viz. processing and storage technology, interconnect technology and software technology. Advances in processing and storage technology can be characterized by advances in device and concurrency technology. Developments in device technology have resulted in faster, more powerful processors with larger storage support and increased functionality, while research in concurrency technology has explored new concurrency paradigms designed to exploit parallelism at different levels and in different ways (e.g. SIMD, shared memory MIMD (SM-MIMD), distributed memory MIMD (DM-MIMD), Dataflow, Vector, Pipelined, etc.). Advances in interconnect technology have introduced (a) high speed, reliable networks capable of providing high transfer rates (e.g. FDDI, DQDB, HIPPI, SONET, ATM, etc.), (b) new, more efficient communication protocols (e.g. NETBLT, VMTP, XTP, Ultranet, etc.) and (c) exotic interconnection topologies

(e.g. FAT Tree, Hypercube, Mesh, Torus, etc.). Advances in software technology have explored new approaches to assist the user in developing parallel software and application. This has included the development of automatic parallelizing/vectorizing compilers, parallel programming languages and language extensions, parallel software development environments, etc. along with support tools such as performance analysis/monitoring tools, problem decomposition/mapping tools, parallel debugging tools, etc.

High performance computer systems today, include SIMD architectures like CM2 and DECmpp, shared memory MIMD, vector and pipelined architectures like the CRAY C90, NEC SX3, and IBM POWER/4, distributed memory MIMD machines like the Paragon XP/S and iPSC/860 from Intel, the CM-5 from TMC, and the KSR1, transputer based machines like the Parsytec GC, special purpose architectures like the BBN MP2000, etc. Each of the above architectures can be thought of as points in the state space of possible alternatives in the design of parallel computers and result from a unique set of trade-off's in system parameters and design decisions. These design trade-off's cause specific architectures to favor certain computational models and thereby deliver maximum performance only to a specific set of applications which lend themselves to one of those computation models. Further, this narrow applicability of current architectures has prevented them from being cost-effective. As a result, although these architectures incorporate large amount of computing power, they are not general enough to efficiently support today's computation-intensive problems, that warrant multiple computational models and levels of parallelism. The U.S. Office of Science and Technology Policy's Committee on Physical, Mathematical, and Engineering Sciences has outlined a set of desired applications and their computing requirements in its report "Grand Challenges: High Performance Computing and Communication" (1991) [1]. The Grand Challenge applications include climate modeling, fluid turbulence, pollution dispersion, human genome, ocean circulation, quantum chormodynamics, semiconductor modeling, superconductor modeling, combustion systems and vision and cognition among others, and are estimated to requires Teraflops (10^{12} flops) of computing power. Tackling applications of this magnitude and diversity would require a general, cost-effective, scalable, yet powerful computing model which will be able to efficiently support its varied computational and communication requirement. It is this realization that has spurred intense research in heterogeneous computing environments [2, 3, 4, 5, 6, 1, 7, 8].

We believe that the future of parallel computing lies in the integration of the plethora of "specialized" architectures into a single Heterogeneous High Performance Computing (HHPC) environment that allows them to cooperate in solving complex problems (Figure 1). The HHPC environment will capitalize on existing architectures and on current advances in computing, networking and communication technology to provide efficient, cost-effective, scalable, high-performance distributed computing.

Software development in any Parallel/Distributed environment is a non-trivial process and requires a thorough understanding of the application and the architecture. This is apparent from the fact that, applications are currently able to achieve only a fraction of peak available performance [7, 1]. The percentage of the peak performance achieved by standard parallel benchmarks on current parallel/distributed systems is

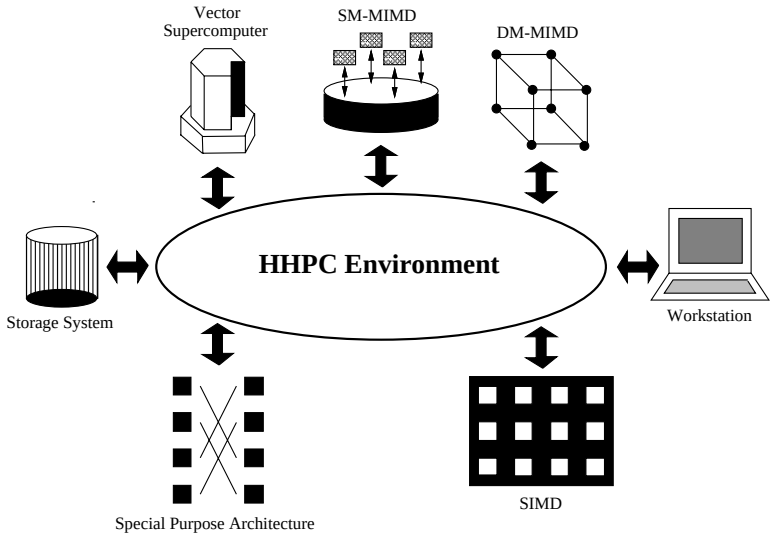


Figure 1: The Heterogeneous High Performance Computing Environment (HHPC)

System Name	Configuration (# Processors)	Peak Speed (Gigaflops)	NAS Parallel Benchmark Efficiency (%)		
			EP (2^{28})	FFT ($256^{28} \times 128$)	CG (2×10^6)
Cray C90	16	16	50%	30%	16%
Intel iPSC/860	128	2.6	15%	20%	3%
TMC CM-200	64 K	20	12%	-	-
TMC CM-2	64 K	14	-	4%	-
TMC CM-2	16 K	3.5	-	-	3%

Note:

NAS = Numerical Aerodynamic Simulation

EP = Highly parallel Monte Carlo Simulation

FFT = 3-D Poisson PDE solver using FFT

CG = Conjugate Gradient linear equation solver for a banded system of equation

Table 1: Current utilization of parallel/distributed systems

shown in Table 1 [1]. The software development problem further intensifies as systems evolve into HHPC environments. The HHPC environment, while increasing the computing power and design flexibility available to the user, provides increased degrees of freedom and therefore requires the developer to make a larger number of design choices. During the course of software development in an HHPC environment, the developer is required to select the optimal hardware configuration for a particular application, the best decomposition and mapping of the problem onto the selected hardware configuration, the best communication and synchronization strategy to be used, etc. Using conventional techniques, this would require extensive experimentation and data collection before these parameters can be resolved. The process is not

always feasible since parallel/distributed systems are expensive resources and usually not freely available for such experimentation. Further, programming, running and data collection on most parallel/distributed systems is a tedious process and exhaustively evaluating the possible alternatives is usually not practical. Most existing evaluation tools post-process traces generated during an execution run. This implies instrumenting source code, executing the application on the actual hardware to generate trace files, post-processing these trace files to gain insight into the execution and overheads in the implementation, refining the implementation and then repeating the process. The process is repeated until all possibilities have been evaluated and the best options for the problem have been identified. Clearly, this development overhead explains the poor exploitation of existing parallel/distributed platforms.

Consequently, there is a need for a software development environment which can assist the developer in uncovering the inherent parallelism in the application, to make efficient use of the underlying computing resources and to exploit the heterogeneity in both, application and hardware. Such an environment should outline the stages involved in the software development process and incorporate tools to support the developer during each stage of application development starting from the specification and design formulation stages through the programming, mapping , distribution, scheduling phases, tuning and debugging stages upto the evaluation and maintenance stages.

The objective of this paper is to formalize the software development process for an HHPC environment and to outline the stages encountered. Section 2 outlines the issues and requirements that need to be addressed in HHPC software development. Section 3 introduces a software development model which meets the requirements outlined and describes the different stages of the model. Support required at each stage is also highlighted. Modeling of stock option pricing is used as a running example to validate the applicability of the proposed model. Section 4 surveys existing tools and techniques applicable to the different stages of the model. Section 5 presents some concluding remarks.

2 Issues and Requirements for HHPC Software Development

This section highlights some of the issues that need to be addressed when designing HHPC software development environments. The first set of issues (Sections 2.1 - 2.7) primarily focus on efficient software development and high performance. This set includes issues pertaining to programming paradigms, application description media, algorithm classification evaluation, and mapping support, implementation/run-time support and visualization/animation support. In addition to these issues, there exists another set of equally important issues (Sections 2.8 - 2.10) that need to be addressed. These issues involve the maintainability, reusability and reliability of the developed application. Some of these issues have been mentioned in [9, 10, 11, 12, 8, 13]. The issues are discussed below:

2.1 General Parallel Programming Models

Although a number of models for parallel programming have been proposed, these models are either too general to be of practical use (e.g. Petrinet, PRAM or Boolean Circuit models) or are bound to specific machines or architectural classes. Models of the former type are primarily theoretical and do not correspond to the behavior of any real machine; while those of the latter type are not general enough to describe algorithms to be implemented on different architectures. As a result, there is a need for a robust software development paradigm which addresses the issues that surface in an HHPC environment and a parallel programming model which is general enough to handle the diversity inherent in such an environment. The model must be architecture independent, must be intuitive, simple to use and must be accurate enough to reflect its cost of execution. The *Paralation Model* presented in [14] attempts to meet these requirements.

2.2 Portable Application Description Medium

A number of application description media, capable of describing and handling parallelism have been proposed. However, they either lack the flexibility to exploit the potential of the underlying hardware (e.g. parallel extensions) or are tied to particular systems and lack portability (e.g. parallel languages). These can be classified as either *parallel extension* or *parallel languages*. The former class consists of classical languages like C, Fortran, Pascal, etc. with appropriate language extensions to handle parallel processes, communication, synchronization, etc. These languages allow easier porting of existing application to parallel platforms. However, parallelism is introduced in these languages as an after-thought which prevents them from being efficient and flexible enough to fully exploit the potential of the underlying hardware. The latter class of application description media, includes new languages designed specifically to handle parallelism, e.g. Occam, PCN, etc. These media are better suited for parallel machines but require re-coding of complete applications. Further, current application description media for parallel/distributed computing are tied to a particular machine or a particular architecture class. For example CMFortran and C* are specific languages for the Connection Machines from TMC, MPFortran is targeted to the DECmpp's, while Occam has been designed for transputer based systems. Hence, there is a need for a portable, yet flexible application description medium which can efficiently support HHPC. HPF [15] proposed by the High Performance Fortran Forum is a step in this direction.

2.3 Algorithm Classification Support

An HHPC environment provides architectural support for parallelism at multiple levels (e.g. instruction level, procedure level, etc,) and of different types (e.g. data parallelism, control parallelism, etc.). and presents the developer with the potential of exploiting corresponding levels and types of parallelism within the application. This can be done by mapping relevant portions (tasks) of the application onto processing

elements of the HHPC environment which are best suited for it. In-order to achieve this, it is necessary to be able to classify an algorithm on the basis of its computational/communication needs. Such a classification must be able to cover the largest possible set of applications while retaining sufficient detail to enable selection of the best hardware configuration.

2.4 Algorithm Evaluation Support

It is critical to HHPC software development, to be able to obtain a realistic estimate of the complexity or potential performance of an algorithm. This allows the developer to evaluate different algorithms for the problem and to make proper design choices early in the application development process.

2.5 Algorithm Mapping Support

The mapping of the algorithm onto the right hardware configuration in the HHPC environment is critical to exploiting its potential performance. Any such mapping support should make use of algorithm evaluation and classification support along with benchmarking data and the performance specifications of the underlying hardware, to assist the developer in identifying the most suited hardware configuration for the particular application. A knowledge-base of rules and heuristics can be used for this purpose.

2.6 Program Implementation and Run-Time Support

In current HHPC environments, even after having selected the right algorithm and system level mapping for an application, the developer has to spend a considerable amount of time understanding the underlying hardware and resolving system specific aspects like selection of synchronization strategies, routing, data decomposition, load balancing, vectorization strategies, pipelining strategies, etc., to get the best possible performance. Following this, additional time and considerable effort required for actually programming the application and debugging the implementation. As a result, there is a critical need for a programming environment providing parallel language support, intelligent compilers and cross-compilers, parallel debuggers, syntax-directed editors, configuration management tools as well as other programming aids to assist the user with every aspect of application development. In addition, such an environment must provide extensive evaluation support for performance prediction and estimation as well as performance evaluation, to enable the developer to evaluate different design choices and tradeoff's. These tools must account for the non-deterministic nature of communication as well as the lack of global ordering of events which are characteristic of distributed systems, and for the heterogeneity in the environment. Run-Time support for HHPC environments includes providing efficient, parallel run-time libraries, dynamic scheduling and load-balancing support, as well as support for non-intrusive monitoring and profiling of application execution. As a part of this category, we can also mention the need for a truly distributed operating system which provides low-level support for HHPC and provides the user with a familiar user interface.

2.7 Visualization/Animation Support

Since, the HHPC environment can process huge amounts of data at high speeds, there is a need for visualization and animation support to enable the user to interpret the data. Further, visualization and animation support enable the user to obtain insight into the actual execution of the application and the existing inefficiencies.

2.8 Maintainability Issues

Maintainability issues include ensuring that the developed software continues to meet its specifications and handling any faults or bugs that might surface during its lifetime. It deals with issues like software testability, software quality evaluations and fault handling and recovery. HHPC environments introduce a great deal of non-determency in the application execution, specially due to asynchronous behavior and synchronization problems, which have to be handled.

2.9 Reliability Issues

Reliability and software fault-tolerance are specially critical to HHPC environments due to the lack total ordering of events in the system as well as lack to repeatability of event execution. Fault detection and recovery in such a system is complex since the order of computation cannot be easily determined. This is further intensified by the fact that multiple autonomous computing elements are involved.

2.10 Reusability Issues

Software reusability issues, as with sequential computing, deal with productivity and development costs. However, as software development for HHPC environments involve higher costs and much greater efforts, these issues have added significance to such an environment and must be addressed.

3 A Model for HHPC Software Development

The HHPC software development model described in this section is defined as a set of stages, which correspond to phases typically encountered in the software development process. At each stage, a set of support tools which can assist the developer are identified. The stages can be viewed as a set of filters in cascade (see Figure 2). The input to this system of filters is the application description and specification which is generated from the application itself (if it is a new problem) or from existing sequential code (porting of dusty decks). The final output of the model is a running application. All intermediate stages are managed within the environment, with user interaction. Feedback loops are present at some stages to enable step-wise refinement and tuning. Descriptions of models for traditional parallel computing environments spanning parts of the software development process can be found in [11, 9, 16]. The stages in

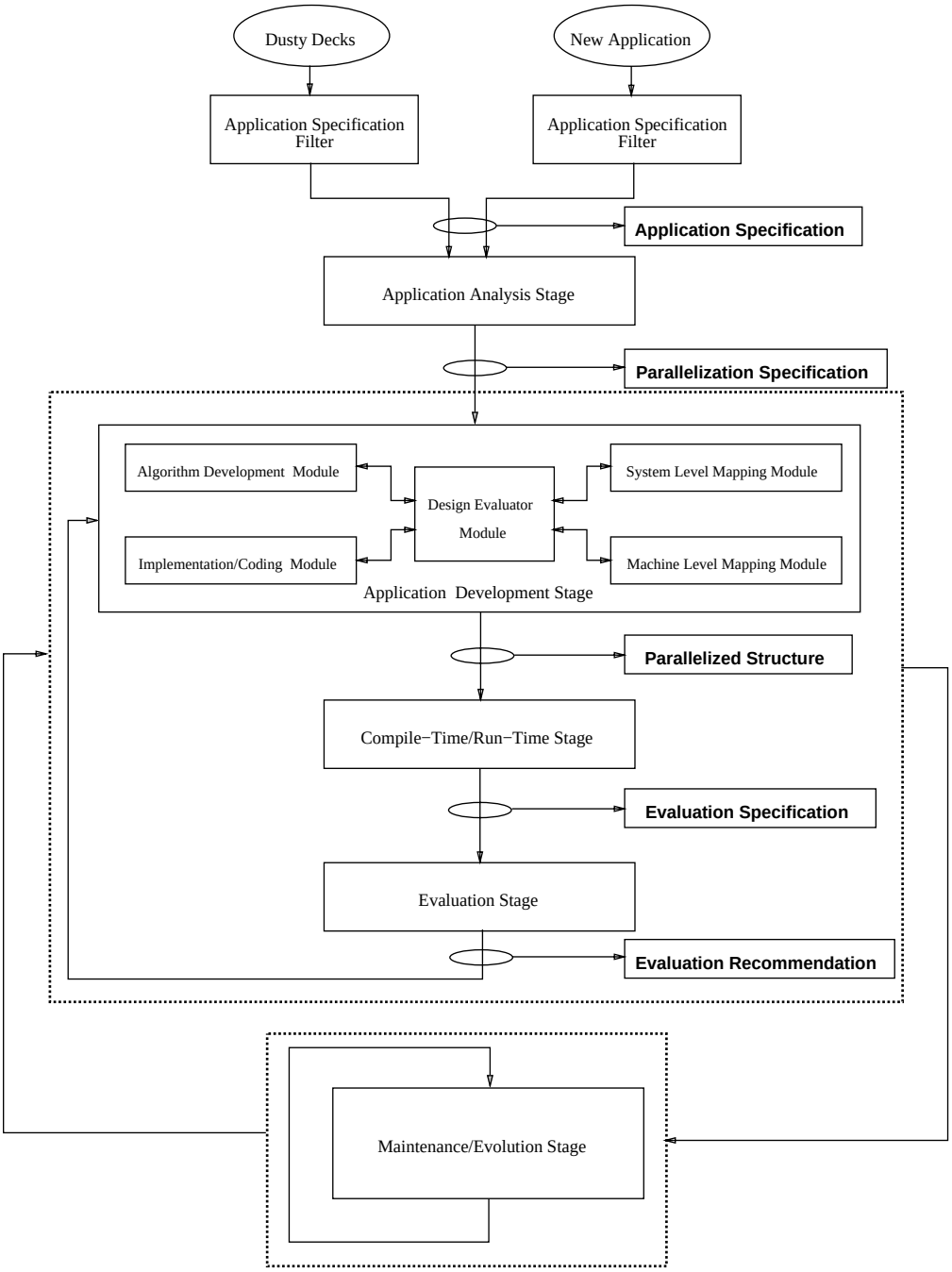


Figure 2: A Model for HHPC Software Development

the HHPC software development model are described in the following sections. To illustrate the validity of the proposed model and the application of the various stages of the model, we use the Modeling of Stock Option Pricing [17] as a running example through the discussion below.

3.1 Parallel Modeling of Stock Option Pricing

Stock options are contracts that give the holder of the contract the right to buy or sell the underlying stock at some time in the future for an agreed upon striking or exercise price. Option contracts are traded just as stocks and models that quickly and accurately predict their prices are valuable to the traders. Stock option pricing models estimate the price for an option contract based on historical market trends and current market information. The model required three classes of inputs: (1) **Market Variables** which include the current stock price, call price, exercise price and time to maturity. (2) **Model Parameters** which include the volatility of the asset (variance of the asset price over time), variance of the volatility and the correlation between asset price and volatility. These parameters cannot be be directly observed and must be estimated from historical data (using optimization techniques). (3) **User Inputs** which specify the nature of the required estimation; e.g. American/European call, constant/stochastic volatility, time of dividend payoff, and other constraints regarding acceptable accuracy and running times. A number of option pricing models have been developed using varied approaches, e.g. non-stochastic analytic models, Monte Carlo simulation models, binomial models, binomial models with forced recombination, etc. Each of these models involve a set of tradeoff's in the nature and accuracy of the estimation and suit different user requirements. In addition, these models make varied demands in terms of programming models and computing resources.

3.2 Model Inputs

The HHPC software development model presented in this section addresses two classes of applications:

1. **"New" Application Development:** This class of applications involves solving new problems using the resources of an HHPC environment. Developers of this class of applications have to start from scratch using a textual description of the problem.

2. **Porting of Exiting Applications (Dusty-Decks):** This class includes developers attempting to port existing codes written for single processor or closely-coupled multiprocessor systems, to an HHPC environment. Developer of this class applications start off with huge listings of (hopefully) commented source code.

The input to the HHPC software development model is an application specification in the form of a functional flow description of the application and its requirements. The functional flow description is a very high-level flow diagram of the application outlining the sequence of functions that have to be performed. Each node (termed as functional module) in the functional flow diagram is a black-box and contains information about (1) its input(s), (2) the function to be performed, (3) the desired output(s) and (4) the requirements at each node. Implementation issues like the approach or algorithm to be used to realize a function or the nature of data representation to be used, are not included in this specification.

Figure 4: Stock Option Pricing Model: Parallelization Specifications

information and historical data and generates the three classes of inputs required by the model. (2) The estimation module consists of the actual model and generates the stock option pricing estimates. (3) The output module provides a graphical display of the estimation to the user. The feedback from the output module to the input module represents tuning of the user specification based on the output displayed.

3.3 Application Analysis Stage

The first stage of the HHPC software development model is the application analysis stage. The input to this stage is the application specification as described in Section 3.2. The function of this stage is to thoroughly analyze the application with the sole objective of achieving the most efficient implementation. An attempt is made, in this stage, to uncover any parallelism inherent in the application. Functional modules which can be executed concurrently are identified and the dependencies between these modules are analyzed. In addition, the application analysis stage attempts to identify standard computational modules which can later be matched with a database of optimized templates in the application development stage (for example, nodes in the application specification performing a Fast Fourier Transform can be clustered and tagged so that they can be matched with an appropriate FFT template in the application development stage). The output of this stage is a detailed process flow graph called the "Parallelization Specification" where the nodes represent functional components and the edges represent interdependencies. Thus, the problems dealt with in this stage can be summarized as: (1) module creation problem, i.e. identification of tasks which can be executed in parallel; (2) module classification problem i.e. identification of standard modules; and (3) module synchronization problem, i.e. analysis of mutual interdependencies. This stage corresponds to the "design phase" in standard software life-cycle models and its output corresponds to the "design document".

The tools which can assist the user at this stage of software development are: (1) smart editors which can interactively generate directed graph models from the application specifications; (2) intelligent tools with learning capabilities which can use the directed graphs to analyze dependencies, identify potentially parallelizable modules and attempt to classify the functional modules into standard modules; and (3) problem specific tools equipped with a database of transformations and strategies applicable to the specific problem.

The parallelization specification for the running example is shown in Figure 4. The Input functional module is subdivided into two functional components: (1) analyzing historical data and generating model parameters; and (2) accepting market information and user inputs to generate market variables and estimation specifications. The two components can be executed concurrently. The Estimation module is identified as a standard computational module and is retained as a single functional component (to avoid getting into the details of financial modeling in this paper). The Output functional module consists of two independent functional components: (1) rendering the estimated information onto a graphical display; and (2) writing it onto disk for subsequent analysis.

3.4 Application Development Stage

The application development stage receives as its input the Parallelization Specifications and produces the Parallelized Structure which can then be compiled and executed. This stage is made up of 5 modules: (1) Algorithm Development Module; (2) System Level Mapping Module; (3) Machine Level Mapping Module; (4) Implementation/Coding Module; and (5) Design Evaluator Module. It should be noted, however, that these modules are not executed in any fixed sequence or a fixed number of times. There exists instead, a feedback system from each module to the other modules through the design evaluator module. This allows the development as well as the tuning to proceed in an iterative manner using step-wise refinement. A typical sequence of events in the application development stage can be outlined as follows:

- The Algorithm Development Module uses an initial system level mapping (possibly specified via user directives) to select appropriate algorithms for the functional components.
- The Algorithm Development Module then uses the services of the Design Evaluator Module to evaluate applicable algorithms and to tune the selection of the algorithmic implementations.
- The System Level Mapping Module uses feedback provided by the Design Evaluator Module and the Algorithm Development Module to tune the initial mapping.
- The Machine Level Mapping Module selects an appropriate machine level distribution and mapping for the particular algorithmic implementation and system level mapping. Once again, feedback from the Design Evaluator Module is used to select between alternate mappings.
- This process of step-wise refinement and tuning is continued until some termination criterion is met (e.g. until some acceptable performance is achieved or up to a maximum time limit).
- The selected algorithm, system level mapping and machine level mapping are realized by the Implementation/Coding Module which generates the parallelized structure.

3.4.1 Algorithm Development Module

The function of the algorithm development module is to assist the developer in identifying functional components in the parallelization specification and selecting appropriate algorithmic implementations. The input information to this module includes: (1) the classification and requirements of the components specified in the parallelization specification; (2) hardware configuration information; and (3) mapping information generated by the system level mapping module. It then uses this information to select the best algorithmic implementation and the corresponding implementation template from its database. It also analyzes the requirements of the selected algorithm (e.g. communication requirements, synchronization requirements, storage requirements, etc.). The algorithm development module uses the services of the design evaluator module to select between possible algorithmic implementations. Tools needed during

this phase include an intelligent algorithm development environment (ADE) equipped with a database of optimized templates for different algorithmic implementations, an evaluation of the requirements of these templates and an estimation of their performance on different platforms.

The algorithm chosen to implement the Estimation Component of the stock option pricing model (shown in Figure 4), depends on the nature of the estimation (constant/stochastic volatility, American/European calls/puts, dividend payoff time, etc) to be performed and the accuracy/time constraints. For example, models based on Monte Carlo simulation provide high accuracy. However, these models are computationally intensive and slow and thereby cannot be used in real-time systems. Further they are not suitable for American calls/puts when early dividend payoff is possible. Binomial models are less accurate than Monte Carlo models but are more tractable and can handle early exercise. Models using constant volatility (as opposed to treating volatility as a stochastic process) lack accuracy but are simplistic and easy to compute. Modeling American calls where in the option can be exercised anytime during the life of the contract (as opposed to European calls which can only be exercised at maturity) is more involved and requires sophisticated and computationally efficient model (e.g. binomial approximation with forced recombination).

The algorithmic implementations of the input and output functional components must be capable of handling terminal and disk I/O at rates specified by the time constraint parameters. Further, the output display must provide all information required by the user.

For an illustration of the operation of this module for a predefined mapping, consider a functional component which requires the solution of a system of linear equations. If it is mapped onto an SIMD architecture, a direct parallelization of the Gauss-Jordan algorithm is applicable. However, if the target machine has a MIMD architecture, the blocked Gauss-Seidel algorithm will be more efficient.

3.4.2 System Level Mapping Module

The function of the system level mapping module is to use the information provided by the algorithm development module to appropriately map the functional components of the application to the appropriate computing elements of the HHPC environment. The objective is to map each functional component to the computing element that maximizes the performance of the application. Some data and load distribution issues may have to be resolved in this module. In addition, this module may also cluster functional component nodes specified in the parallelization specifications to obtain a better mapping. The system level mapping module uses feedback from the evaluation module to select between different mapping candidates.

System level mapping can be accomplished in an interactive mapping environment equipped with intelligent tools for analyzing the requirements of the functional components, and a knowledge base consisting of analytic benchmarks for the different computing elements and interconnection media in the HHPC en-

vironment. The tools use this information to interactively select an appropriate mapping of algorithmic implementations to the computing elements.

The algorithms for stock option pricing have been efficiently implemented on architectures like the CM2 and the DECmpp-12000 [17]. Thus, an appropriate mapping for the estimation functional component in the parallelization specification in Figure 4 is an SIMD architecture. The input and output interfaces (Input/Output Component-A) require graphics capability with support for high speed rendering (output display) and must be mapped to an appropriate graphics stations. Finally, Input/Output Component-B requires high speed disk I/O and must be mapped to an I/O server with such capabilities.

3.4.3 Machine Level Mapping Module

The machine level mapping module performs the mapping of the functional component(s) onto the processor(s) of the computing elements. This stage resolves issues like data partitioning, load distribution, control distribution, etc. and makes transformations specific to that computing element. It uses the feedback from the design evaluator module to select between possible alternatives. Machine level mapping can be accomplished in an interactive mapping environment similar to that described for the system level mapping module, but equipped with information pertaining individual computing elements of a specific computer architecture.

The performance of the stock option pricing models are very sensitive to the layout of data onto the processing elements. The optimal layout is dictated by the input parameters (e.g. time of dividend payoff, terminal time, etc.) and by the specification of the architecture onto which the component is mapped. For example, in the binomial model, the continuous time processes for stock price and volatility are represented as discrete up/down movements forming a binary lattice. Such lattice is generally implemented as asymmetric arrays which are distributed onto the processing elements. It has been found that the default mapping of these arrays (i.e. in two dimensions) on architectures like the DECmpp-12000, lead to poor load balancing and performance, specially for extreme values of the dividend payoff time [18]. Further the performance in case of such a mapping, is very sensitive to this value and has to be modified for each set of inputs. Hence, in this case it is favorable to explicitly map them as one dimensional arrays. This is done by the machine level mapping module. As another example of the mapping performed by this module, consider the case of a functional component performing linear algebra which is allocated to a hypercube architecture. The function of the machine level mapping module would be to decide whether to distribute the matrix in a block or cyclic fashion and whether to perform this distribution in a column or row major fashion.

3.4.4 Implementation/Coding Module

The function of the implementation/coding module is to handle all code generation and perform the code filling of selected templates so as to produce parallel code which can then be compiled and executed on the target computer architecture. This module incorporates all machine specific codes, handles the introduction of calls to communication and synchronization routines and takes care of the distribution of data among the processing elements. It also handles any input/output redirection that may be required. Machine specific transformations and calls to optimized machine specific libraries are inserted by this module.

With regard to the pricing model application, the implementation/coding module is responsible for introducing the machine specific communication routines. For example, the binary estimation model makes use of the "end-of-shift" function for its nearest-neighbor communication. The corresponding function call in C^* (CM2) or MPL (DECmpp-12000) are introduced by this module. A possible machine specific optimization that can be introduced by this module is to reduce communication by making use of in-processor arrays. This optimization can improve performance by about two orders of magnitude [17].

3.4.5 Design Evaluator Module

The design evaluator module is a critical component of the application development stage. Its function is to assist the developer in evaluating different options (e.g. algorithms, implementations, system level mappings, machine level mappings including data partitioning, etc.), available to each of the other modules, and identifying the option that provides the best performance. It receives information about the hardware configuration, the application structure, the requirements of the selected algorithms and the mappings. This input information is then used to estimate the performance of the application on the target computer. Further, it provides insight into the computation and communication costs, the existing idle times and the overheads. This information can be used by the other modules to identify regions where further refinement or tuning is required. The module can evaluate the correctness of the implementation using performance debugging as a criterion and can detect synchronization error like deadlocks. Finally, many runtime scenarios can be evaluated (e.g. system load, network contention). The keys features of this module are: (1) the ability to provide evaluations with the desired accuracy, with minimum resource requirements and within a reasonable amount of time; (2) the ability to automate the evaluation process; and (3) the ability to perform the evaluation within an integrated workstation environment without running the application on the target computers. Support applicable to this module consists primarily of performance prediction and estimation tools. Simulation approaches can also be used to achieve some of the required functionality.

3.5 Compile-Time & Run-Time Stage

The compile-time/run-time stage handles the task of executing the parallelized application generated by the development stage to produce the required output. The input to this stage is the parallelized source

code (parallelized structure). The compile-time portion of this stage consists of set of cross compilers for the computing elements and tools for scheduling and allocation. These compilers have appropriate optimization capabilities and can introduce machine-specific optimizing transformation into the parallelized structure. The compile-time software also handles the loading of the executables onto appropriate computing elements. The run-time portion of this stage handles run-time functions like debugging, scheduling, dynamic load balancing, migration, irregular communications, etc. It also enables the user to (non-intrusively) instrument the code for profiling and debugging and allows checkpointing for fault-tolerance. During the execution of the application, it accepts outputs from the different computing elements and directs them for proper visualization. It intercepts error messages generated and provides proper interpretation.

3.6 Evaluation Stage

In the evaluation stage, the developer, retrospectively evaluates the design choices made during the design process and looks for ways to improve the performance. The evaluation stage performs a thorough evaluation of the execution of the entire application, detailing communication and computation times, communication and synchronization overheads and existing idle times at every execution level (application level, node level, process level, procedure level, etc.). It uses this evaluation to identify regions in the implementation where performance improvement is possible. The evaluation procedure is accurate, non-intrusive and does not alter the execution order of the application. Further, it allows a cost-effective evaluation (in terms of time and resources) of the application for a representative inputs set as well as the effect of various run-time parameters like system load, network contention, on performance. The scalability of the application with machine and problem size is also evaluated. The key features of this stage are: (1) the ability to provide desired accuracy and granularity of evaluation while maintaining tractability and non-intrusiveness; and (2) the ability to perform the evaluation within a friendly workstation environment without requiring the actual hardware. Support applicable to the evaluation stage include different analytic tools, monitoring tools, simulation tools and prediction/estimation tools.

3.7 Maintenance/Evolution Stage

In addition to the above described stages encountered during the development and execution of HHPC applications, there is an additional stage in the life-cycle of this software which involves its maintenance and evolution. Software maintenance is an important part of the software life-cycle and is known to span around 70% of this cycle. Maintenance includes monitoring the operation of the software and ensuring that it continues to meet its specifications. It involves detecting and correcting bugs as they surface. The maintenance stage also handles modifications needed to incorporate changes in the system configuration. Software evolution deals with improving the software, adding additional functionality, incorporating new optimizations, etc. Another aspect of evolution is the development of more efficient algorithms and corresponding algorithmic templates and the incorporation of new hardware architectures. To support such

a development, the maintenance/evolution stage provides tools for the rapid prototyping of hardware and software and for evaluating the new configuration and designs without having to implement them. Other support required during this stage includes tools for monitoring the performance and execution of the software, fault detection and recovery tools, system configuration and configuration evaluation tools and prototyping tools.

4 Existing Software Support

Development Stage	Required Tools	Existing Tools/Approaches
Application Specification Filter	SA/SD CASE Tools	SAMTOP (TOPSYS)
Application Analysis Stage	Intelligent Editors, Problem Specific Databases	Sigma (FAUST), SAMTOP (TOPSYS) Parafrase-2
Application Development Stage		
(a) Algorithm Development Module	Intelligent ADE's, Databases, Optimized Templates	SCHEDULE, SKELETONS
(b) System Level Mapping Module	Intelligent Mapping Tools, Analytic Benchmarks	SAMTOP (TOPSYS), MARC, Paralex, TEACHER 4.1
(c) Machine Level Mapping Module	same as system level mapping	ParaScope
(d) Implementation/Coding Module	Code Generation Tools, Code Optimizers	Proteus, SUPERB, S. Bhatt et al.
(e) Design Evaluator Module	Performance Prediction Tools	V. Balasundaram et al. A. Sussman et al. M. Gupta et al.
Compile-Time/Run-Time Stage	Intelligent Optimizing Compilers, Dynamic Load-Balancing Tools, Debuggers, Profilers, Visualization Tools, Error-handling Support, etc.	Parafrase-2, DETOP (TOPSYS), FAUST
Evaluation Stage	Performance Analysis Tools, Performance Monitoring Tools, Performance Simulation Tools, Performance Prediction Tools	PATOP, VISTOP (TOPSYS), SIMPLE, IPS-2, FAUST, CPPP, RPPT
Maintenance/Evolution Stage	Monitoring Tools, Fault Detection/Recovery Tools, System Configuration Tools, Prototyping Tools, Predictive Evaluation Tools	PAWS, SiGle, Proteus.

Table 2: HHPC Software Development Model - Requirements and Existing Support

In this section we survey some existing tools which provide support at different stages of the software

development process. Our objective is twofold: (1) demonstrate the nature of support needed at each stage of the HHPC software development model; and (2) illustrate the fact that, although a large number of individual tools or systems have been developed, there is a lack of an integrated environment which can support the developer through the entire software development process and assist in fully utilizing the potential of an HHPC environment. Table 2 summarizes the tool requirements and existing support at each stage of the HHPC software development model proposed in this paper. In what follows, we briefly discuss these tools.

Application Specifications Filter The SAMTOP tool, which is proposed to be a part of the TOPSYS [19] system, will provide provide the functionality required by this stage. In addition, existing SA/SD (Structured Analysis/Structured Design) CASE tools can be used at this stage.

Application Analysis Stage The Sigma editor, which part of the FAUST [20] parallel programming environment, provides the support required by this stage for shared memory architectures. It provides intelligent, interactive editing and parallelizing capabilities and incorporates a performance predictor. Another existing system applicable to this stage is Parafrase-2 [21]. The SAMTOP tool discussed above will also provide some analysis capabilities.

Application Development Stage At the application development stage, tools like SCHEDULE [22] and SKELETONS assist the user during algorithm development while MARC, Paralex [23] and TEACHER 4.1 [24] provide mapping support. SKELETONS and and MARC are part of an integrated application development environment and run-time environment for transputer based systems [9]. Existing approaches which provide some of the functionality of the design evaluator module include methodologies proposed by V. Balasundaram et al. [25], A. Sussman [26] and M. Gupta et al. [27]. Support for implementation and coding is provided by the Proteus system [28], SUPERB [29], and by the system proposed by S. Bhatt et al. [30].

Other tools providing support during application development include ParaScope [31] and SPADE [9]. SAMTOP and Sigma systems also provide some functionality need by the stage.

Compile-Time & Run-Time Stage Support required by this stage of software development is provided by the FAUST and TOPSYS systems discussed above. TOPSYS provides debugging support (DETOP) while FAUST incorporates a compile-time and run-time environment. Another tool applicable to this stage is Parafrase-2.

Evaluation Stage Existing evaluation systems include PATOP and VISTOP from TOPSYS, the IPS-2 system [32], the SIMPLE environment [33], and RPPT [34]. FAUST and RPPT [34] specifically provide evaluation support for the CEDAR computer system.

Maintenance/Evolution Stage The PAWS systems [35] presents an approach for machine evaluation and can be used during the maintenance/evolution stage. System prototyping capabilities are provided by SiGle [36] and Proteus [28].

5 Conclusions

Current trends in parallel/distributed computing indicate that the future of parallel computing lies in the integration of the plethora of existing "specialized" architectures into a single Heterogeneous High Performance Computing (HHPC) environment that allows them to cooperate in solving complex problems. The HHPC environment will capitalize on existing architectures and on current advances in computing, networking and communication technology to provide efficient, cost-effective, scalable, high-performance distributed computing.

Software development in any Parallel/Distributed environment is a non-trivial process and requires a thorough understanding of the application and the architecture. This apparent from the fact that currently, applications are able to achieve only a fraction of peak available performance. The software development problem further intensifies as systems moves to an HHPC environment. The HHPC environment, while increasing the computing power and design flexibility available to the user, provides increased degrees of freedom and requires the developer to make a larger number of design choices.

This paper formalizes the software development process for an HHPC environment. It describes a software development model and outlines the support required at each stage of the model. The development of a parallel model for stock option pricing is used as a running example to demonstrate the applicability of the model. Existing tools techniques corresponding to the different stages are also surveyed. The ultimate goal of the ongoing High Performance Fortran project at the Northeast Parallel Architectures Center, Syracuse University, is to realize such an environment. Currently our efforts are focussed on the application development and the compile-time/run-time stages. An interpretive performance prediction framework is also being developed which meets the requirements of the Design Evaluator Module of the application development stage and of the Evaluation Stage.

Acknowledgment

The presented research has been jointly sponsored by DARPA under contract #DABT63-91-k-0005 and by Rome Labs under contract #F30602-92-C-0150. The content of the information does not necessary reflect the position or the policy of the sponsors and no official endorsement should be inferred.

References

[1] Glenn Zorpette, "Teraflops Galore", *IEEE Spectrum*, vol. 29, pp. 26–76, sep 1992.

[2] Salim Hariri, Manish Parashar, Jong Baek Park, Fang-Kuo Yu, and Geoffrey Fox, "A Case for Heterogeneous High Performance Computing", Technical Report SCCS-417, Northeast Parallel Architectures Center, 111 College Place, Room # 3-201, Syracuse NY 13244-4100, 1992.

[3] Salim Hariri, Manish Parashar, JongBaek Park, and Fang-Kuo Yu, "An Environment for High-Performance Distributed Computing", Technical Report SCCS-418, Northeast Parallel Architectures Center, Syracuse University, 111 College Place Room # 3-201, Syracuse NY 13244-4100, 1992.

[4] Salim Hariri, Manish Parashar, JongBaek Park, Fang-Kuo Yu, and Geoffrey Fox, "A Message Passing Interface for Parallel and Distributed Computing", To be presented at the 2nd International Symposium on High Performance Distributed Computing, Spokane, Washington, July 1993.

[5] Richard F. Freund and D. Sunny Conwell, "Superconcurrency: A Form of Distributed Heterogeneous Supercomputing", *Supercomputing Review*, vol. , pp. 47–50, oct 1990.

[6] Norris Parker Smith, "The Future of High-Performance Computing: The 1990 Federal Assessment", *Supercomputing Review*, vol. , pp. 52–53, oct 1990.

[7] Gordon Bell, "Ultracomputers: A Teraflop Before Its Time", *Communications of the ACM*, vol. 35, pp. 27–47, aug 1992.

[8] Ashfaq Khokar, Viktor K. Prasanna, Mohammad Shaaban, and Cho-Li Wang, "Heterogeneous Supercomputing: Problems and Issues", *Heterogeneous Processing Workshop, IPPS '92*, vol. , 1992.

[9] J. E. Boillat, H. Burkhart, K. M. Decker, and P. G. KROPF, "Parallel Computing in the 1990's: Attacking the Software Problem", *Physics Report (Review Section of Physics Letters)*, vol. 207, pp. 141 – 165, 1991.

[10] Joseph P. Cavano, Geoffrey C. Fox, Carl Murphy, and Lucian Russel, "Panel Session: Software Development Issues for Parallel Processing", *Proceedings of the* 12^{th} *Annual International Computer Software and Applications Conference*, vol. , pp. 299–307, 1988.

[11] Victor R. Basili and John D. Musa, "The Future Engineering of Software: A Management Perspective", *IEEE Computer*, vol. 24, pp. 90–96, Sep. 1991.

[12] D. B. Skillicorn, "Models for Practical Parallel Computation", *International Journal of Parallel Programming*, vol. 20, pp. 133–158, 1991.

[13] Grady Booch, *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, 2 edition, 1986.

[14] Gary W. Sabot, *The Paralation Model: Architecture-Independent Parallel Programming*, The MIT Press, Cambridge, Massachusetts, 1988.

[15] Massively Parallel System Group, *High Performance Fortran*, Digital Equipment Corporation, report ml01-5/u46 edition, 1992.

[16] Lucian Russell and R. N. C. Lightfoot, "Software Development Issues for Parallel Processing", *Proceedings of the* 12^{th} *Annual International Computer Software and Applications Conference*, vol. , pp. 306–307, 1988.

[17] Kim Mills, Gang Cheng, Michael Vinson, Sanjay Ranka, and Geoffrey C. Fox, "Software Issues and Performance of a Parallel Model for Stock Option Pricing", *Proceedings of the* 5^{th} *Australian Supercomputing Conference, Melbourne, Australia*, vol. , Dec. 1992.

[18] Kim Mills, Gang Cheng, Michael Vinson, and Geoffrey C. Fox, "Expressing Dynamic, Asymmetric, Two-Dimensional Arrays for Improved Performance on the DECmpp-12000", Technical Report SCCS-261, Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100, Oct. 1992.

[19] Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, and Roland Wismüller, *The Design and Implementation of TOPSYS*, Technische Universität München, Institut Für Informatik, July 1991, Ver 1.0.

[20] D. Gannon, Y. Gaur, V. Guarna, D. Jablonowski, and A. Malony, "FAUST: An Integrated Environment for Parallel Programming", *IEEE Software*, vol. , pp. 20–27, July 1989.

[21] Constantine D. Polychronopoulos, Milind Girkar, Mohammad Reza Haghighat, Chia Ling Lee, and Bruce Leung, "Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors", *Proceedings of the International Conference on Parallel Processing*, vol. 2, pp. 39–48, Aug. 1989.

[22] J. J. Dongarra and D. C. Sorensen, "SCHEDULE: Tools for Developing and Analyzing Parallel Fortran Programs", in L. H. Jamieson, D. B. Gannon, and R. J. Douglas, editors, *The Characteristics of Parallel Algorithms*, vol. . MIT Press, 1987.

[23] Özalp Babaoğlu, Lorenzo Alvisi, Alessandro Amoroso, Renzo Davoli, and Luigi Alberto Giachini, "Paralex: An Environment for Parallel Programming in Distributed Systems", Technical report, Department of Mathematics, University of Bologna, Piazza Porta S. Donato, 5, 40127 Bologna, Italy, 1991.

[24] Arthur Ieumwananonthachai, Akiko N. Aizawa, Steven R. Schwartz, Benjamin W. Wah, and Jerry C. Yan, "Intelligent Mapping of CommunicationProcesses in Distributed Computing Systems", *Supercomputing '91, Proceedings*, vol. , pp. 512–521, 1991.

[25] Vasanth Balasundaram, Geoffrey Fox, Ken Kennedy, and Ulrich Kremer, "An Interactive Environment for Data Partitioning and Distribution", *Proceedings of the* 5^{th} *Distributed Memory Computing Conference, Charleston, South Carolina*, vol. , pp. 1160–1170, Apr. 1990.

[26] Alan Sussman, "Execution Models for Mapping Programs onto Distributed Memory Parallel Computers", Technical Report 189613, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia 23665-5225, Mar. 1992.

[27] Manish Gupta and Prithviraj Banerjee, "Compile-Time Estimation of Communication Costs in Multicomputers", Technical report, Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1101 W. Springfield Avenue, Urbana IL 61801, .

[28] Peter H. Mills, Lars S. Nyland, Jan F. Prins, John H. Reif, and R. W. Wagner, "Prototyping Parallel and Distributed System in Proteus", *Proceedings of the* 3^{rd} *IEEE Symposium on Parallel and Distributed Processing*, vol. , 1991.

[29] H. Zima, H. Bast, and M. Gerndt, "SUPERB: A Tool for Semi-Automatic SIMD/MIMD Parallelization", *Parallel Computing*, vol. , 1988.

[30] Sandeep Bhatt, Marina Chen, Cheng-Yee Lin, and Pangfeng Liu, "Abstractions for Parallel N-body Simulations", Technical Report DCS/TR-895, Yale University, 1992.

[31] Vasanth Balasundaram, Ken Kennedy, Ulrich Kremer, K. McKinley, and J Subhlok, "The ParaScope Editor: An Interactive Parallel Programming Tool", *Supercomputing '89, Reno, Nevada*, vol. , Nov. 1989.

[32] Barton P. Miller, Morgan Clark, Jeff Hollingsworth, Steven Kierstead, Sek-See Lim, and Timothy Torzewski, "IPS-2: The Second Generation of a Parallel Program Measurement System", *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 206–217, Apr. 1990.

[33] Bernd Mohr, "SIMPLE: A Performance Evaluation Tool Environment for Parallel and Distributed Systems", *Proceedings of the* 2^{nd} *European Distributed Memory Computing Conference (EDMCC2)*, vol. , pp. 80–89, Apr. 1991.

[34] R. C. Covington, S. Madala, V. Mehta, J. R. Jump, and J. B. Sinclair, "The Rice Parallel Processing Testbed", 1988 ACM 0-89791-254-3/88/0005/0004 pp 4-11, 1988.

[35] Daniel Pease, Arif Gafoor, Ishfaq Ahmad, David L. Andrews, Kamal Foudil-Bey, Thomas E. Karpinski, Mohammad A. Mikki, and Mohammad Zerrouki, "PAWS: A Performance Evaluation Tool for Parallel Computing Systems", *IEEE Computer*, vol. , pp. 18–29, Jan. 1991.

[36] F. Andre and A. Joubert, "SiGle: An Evaluation Tool for Distributed Systems", *Proceedings of the International Conference on Distributed Computing Systems*, vol. , pp. 466–472, 1987.