1994

A Study of Software Development for High Performance Computing

Manish Parashar
Syracuse University

Salim Hariri
Syracuse University

Tomasz Haupt
Syracuse University

Geoffrey C. Fox
Syracuse University

A Study of Software Development for High Performance Computing

Manish Parashar, Salim Hariri, Tomasz Haupt and Geoffrey Fox
Northeast Parallel Architectures Center
Syracuse University

Published in
Programming Environments for Massively Parallel Distributed Systems
Birkhauser Verlag, Basel, Switzerland, August, 1994
Also, presented at
IFIP WG10.3 Working Conference on Programming Environments
for Massively Parallel Distributed Systems, 1994

Abstract

Software development in a High Performance Computing (HPC) environment is non-trivial and requires a thorough understanding of the application and the architecture. The objective of this paper is to study the software development process in a high performance computing environment and to outline the stages typically encountered in this process. Support required at each stage is also highlighted. The modeling of stock option pricing is used as a running example in the study.

1 Introduction

Software development in any High Performance (Parallel/Distributed) Computing (HPC) 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 currently achieve only a fraction of peak available performance [Zor92]. HPC software development requires the developer to resolve and tune a large number of available design options. For example, during the course of software development, 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, data collection and post-processing. The set of reasonable

alternatives that have to be evaluated is very large and selecting the best among these is a formidable task. As a result the exploitation of the vast potential of HPC systems will largely be governed by the availability of suitable tools and application development environments to support application developers.

The objective of this paper is to study the software development process in a high performance computing environment and to outline the stages encountered. Further, the nature of supporting tools that can assist the developer at each stage are identified. Parallel modeling of stock option pricing is used as an illustrative example in the study. The rest of the document is as organized follows: Section 2 presents the study of HPC software development process and outlines the stages (subsections 2.3 - 2.7). Section 3 presents some conclusions.

2 HPC Software Development

The HPC software development process is described as a set of stages which correspond to the phases typically encountered by a developer. 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 1) forming a development pipeline. 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 pipeline is a running application. Feedback loops present at some stages signify step-wise refinement and tuning. Related discussions pertaining to parallel computing environments and spanning parts of the software development process can be found in [BM91, BBDK91, RL88]. A survey of existing tools and techniques corresponding to the developemnt stages is presented in [PHHF93a]. The stages in the HPC software development process are described in the following sections. Parallel modeling of Stock Option Pricing [MCV+92] is used as an illustrative, running example in the discussion.

2.1 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 requires three classes of inputs: **Market Variables** which include the current stock price, call price, exercise price and time to maturity. **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. **User Inputs** which specify the na-

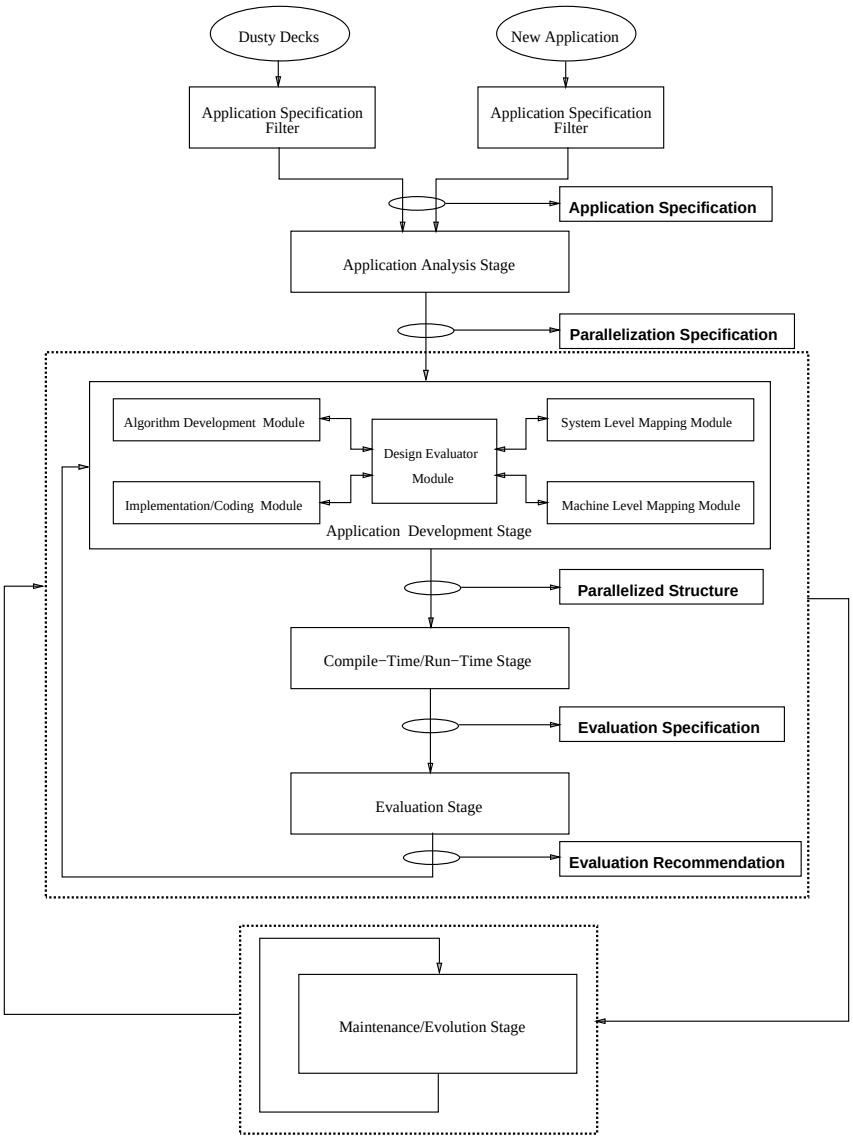

Figure 1: The HPC Software Development Process

ture 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.

2.2 Inputs

The HPC software development process presented in this section addresses "new" application development as well as the porting of exiting applications (Dusty-Decks) to HPC environments. The input to the development pipeline is the application specification in the form of a functional flow description, which is a very high-level flow diagram of the application outlining the sequence of functions 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 resource requirements at each node. The application specification can be thought of as corresponding to the "user requirement document" in a traditional life-cycle models.

In the case of new applications, the inputs are generated from the textual description of the problem and its requirements. In the case of dusty decks code porting, the developer is required to analyze the existing source code. In either case, expert system based tools and intelligent editors, both equipped with a knowledge base to assist in analyzing the application, are required. In Figure 1, these tools are included in the "Application Specification Filter" module.

The stock price modeling application comes under the first class of applications (i.e. new applications). The application specifications based on the textual description presented in Section 2.1, is shown in Figure 2. It consists of three functional modules: (1) The input module which accepts user specification, market 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.

2.3 Application Analysis Stage

The first stage of the HPC software development pipeline is the application analysis stage. The input to this stage is the application specification as described in Section 2.2. The function of this stage is to thoroughly analyze the application with the sole objective of achieving the most efficient implementation. The problems dealt with in this stage are: (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. The output of this stage is a detailed process flow graph called the "Parallelization Specification" where the nodes represent

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. 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.

2.4 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. The modules are described below:

2.4.1 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. 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 3), 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. 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.

2.4.2 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 a distributed (possibly heterogeneous) HPC 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 HPC environment.

The algorithms for stock option pricing have been efficiently implemented on architectures like the CM2 and the DECmpp-12000 [MCV+92]. Thus, an appropriate mapping for the estimation functional component in the parallelization specification in Figure 3 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.

2.4.3 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 a 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. 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.

2.4.4 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 transformations, optimized libraries and 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.

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 [MCV+92].

2.4.5 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 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 configuration. 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 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. A novel approach which uses interpretive techniques to realize a performance prediction framework that can meet these requirements, is presented in [PHHF93b].

2.5 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. 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.

2.6 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, synchronization overheads and existing idle times at every execution level (application level, node level, procedure level, etc.). It uses this evaluation to identify regions in the implementation where performance improvement is possible. 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 requirement of this stage is the ability to provide desired accuracy and granularity of evaluation while maintaining tractability and non-intrusiveness. Support applicable to the evaluation stage include different analytic tools, monitoring tools, simulation tools and prediction/estimation tools.

2.7 Maintenance/Evolution Stage

In addition to the above described stages encountered during the development and execution of HPC applications, there is an additional stage in the life-cycle of this software which involves its maintenance and evolution. 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, and system configuration and configuration evaluation tools.

3 Conclusions

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. This paper studies the software development process for in a High Performance Computing environment. It describes the stages typically involved in this process and outlines the support required at each stage. The development of a parallel model for stock option pricing is used as a running example.

References

[BBDK91] 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)*, 207(3-5):141 – 165, 1991.

[BM91] Victor R. Basili and John D. Musa. The Future Engineering of Software: A Management Perspective. *IEEE Computer*, 24(9):90–96, September 1991.

[MCV+92] 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*, December 1992.

[PHHF93a] Manish Parashar, Salim Hariri, Tomasz Haupt, and Geoffrey C. Fox. An Integrated Software Development Model for Heterogeneous High Performance

Computing. Technical Report SCCS-453, Northeast Parallel Architectures Center, Syracuse University, Syracuse NY 13244-4100, April 1993.

[PHHF93b] Manish Parashar, Salim Hariri, Tomasz Haupt, and Geoffrey C. Fox. An Interpretive Framework for Application Prediction. *Procs of the 1993 Int'l Conference On Parallel and Distributed Systems*, 668–672, Dec. 1993.

[RL88] 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*, 306–307, 1988.

[Zor92] Glenn Zorpette. Teraflops Galore. *IEEE Spectrum*, 29(9):26–76, sep 1992.