Interpreting the Performance of HPF/Fortran 90D

Manish Parashar, Salim Hariri, Tomasz Haupt, and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244-4100
{parashar,hariri,haupt,gcf}@npac.syr.edu

To be presented at
Supercomputing '94, Washington DC

Abstract

In this paper we present a novel interpretive approach for accurate and cost-effective performance prediction in a high performance computing environment, and describe the design of a source-driven HPF/Fortran 90D performance prediction framework based on this approach. The performance prediction framework has been implemented as part of a HPF/Fortran 90D application development environment. A set of benchmarking kernels and application codes are used to validate the accuracy, utility, usability, and cost-effectiveness of the performance prediction framework. The use of the framework for selecting appropriate compiler directives and for application performance debugging is demonstrated.

Keywords: *Performance prediction, HPF/Fortran 90D application development, System & Application characterization.*

1 Introduction

Although currently available High Performance Computing (HPC) systems possess large computing capabilities, few existing applications are able to fully exploit this potential. The fact remains that the development of efficient application software capable of exploiting available computing potentials is non-trivial and is largely governed by the availability of sufficiently high-level languages, tools, and application development environments.

A key factor contributing to the complexity of parallel/distributed software development is the increased degrees of freedom that have to be resolved and tuned in such an environment. Typically, during the course of parallel/distributed software development, the developer is required to select between available algorithms for the particular application; between possible hardware configuration and amongst possible decompositions of the problem onto the selected hardware configuration; between different communication and synchronization strategies; and so on. The set of reasonable alternatives that have to be evaluated is very large and selecting the best alternative among these is a formidable task. Consequently, evaluation tools form a critical part of any software development environment.

In this paper we present a novel interpretive approach for accurate and cost-effective performance prediction in a high performance computing environment, and describe the design of a source-driven HPF[1]/Fortran 90D performance prediction framework based on this approach. The interpretive approach defines a comprehensive characterization methodology which abstracts system and application components of the HPC environment. Interpretation techniques are then used to interpret performance of the abstracted application in terms of parameters exported by the abstracted system. System abstraction is performed offline through a hierarchical decomposition of the computing system. Application abstraction is achieved automatically at compile time. The performance prediction framework has been implemented as a part of the HPF/Fortran 90D application development environment [1] developed at the Northeast Parallel Architectures Center (NPAC), Syracuse University. The environment integrates a HPF/Fortran 90D compiler, a functional interpreter and the source based performance prediction tool; and is supported by a graphical user interface. The current implementation of the environment framework is targeted to the iPSC/860 hypercube multicomputer system.

A set of benchmarking kernels and application

[1] High Performance Fortran

3.1 Systems Module

The systems module abstracts a HPC system by hierarchically decomposing it to form a rooted tree structure called the *System Abstraction Graph (SAG)*. Each node of the SAG is a *System Abstraction Unit (SAU)* which abstracts a part of the HPC system into a set of parameters representing its performance. A SAU is composed of 4 components: (1) Processing Component (P), (2) Memory Component (M), (3) Communication/Synchronization Component (C/S), and (4) Input/Output Component (I/O); each component parameterizing relevant characteristics of the associated system unit.

3.2 Application Module

Application abstraction is performed in two step: Machine independent application abstraction is performed by recursively characterizing the application description into *Application Abstraction Units (AAU's)*. Each AAU represents a standard programming construct (such as iterative, conditional, sequential) or a communication/synchronization operation, and parameterizes its behavior. AAU's are combined to abstract the control structure of the application, forming the *Application Abstraction Graph (AAG)*. The communication/synchronization structure of the application is superimposed onto the AAG by augmenting the graph with a set of edges corresponding to the communications or synchronization between AAU's. The resulting structure is the *Synchronized Application Abstraction Graph (SAAG)*. The second step consists of machine specific augmentation and is performed by the machine specific filter. This step incorporates machine specific information (such as introduced compiler transformations/optimizations) into the SAAG based on a mapping defined by the user.

3.3 Interpretation Engine

The interpretation engine consists of two components; an interpretation function that interprets the performance of an individual AAU, and an interpretation algorithm that recursively applies the interpretation function to the SAAG to predict the performance of the corresponding application. An interpretation function is defined for each AAU type to compute its performance in terms of parameters exported by the associated SAU. Models and heuristics are defined to handle accesses to the memory hierarchy, overlap between computation and communication, and user experimentation with system and run-time parameters. Details of these models and the complete set of interpretation functions can be found in [4].

3.4 Output Module

The output module provides an interactive interface through which the user can access estimated performance statistics. The user has the option of selecting the type of information, and the level at which the information is to be displayed. Available information includes cumulative execution times, the communication time/computation time breakup and existing overheads and wait times. This information can be obtained for an individual AAU, cumulatively for a branch of the AAG (i.e. sub-AAG), or for the entire AAG.

4 Design of the HPF/Fortran 90D Performance Prediction Framework

The HPF/Fortran 90D performance prediction framework is based on the HPF source-to-source compiler technology [5] which translates HPF into loosely synchronous, SPMD[3] Fortran 77 + Message-Passing codes. It uses this technology in conjunction with the performance interpretation model to provide performance estimates for HPF/Fortran 90D applications on a distributed memory MIMD multicomputer. Performance prediction is performed in two phases as described below:

4.1 Phase 1 - Compilation

The compilation phase is based on the HPF/-Fortran 90D compiler. Given a syntactically correct HPF/Fortran 90D program, this phase performs the following steps:

1. The first step parses the program to generate a parse tree. Array assignment statement and *where* statement are transformed into equivalent *forall* statements with no loss of information.
2. The partitioning step processes the compiler directives and using these directives, it partitions the data and computation among the processors.
3. The sequentialization step is responsible for converting parallel constructs in the node program into loops or nested loops.
4. The communication detection step detects communication requirements and inserts appropriate communication calls.
5. In the final step, a loosely synchronous SPMD program structure is generated consisting of alternating phases of local computation and global communication.

[3] Single Program, Multiple Data

4.2 Phase 2 - Interpretation

Phase 2 is implemented as a sequence of parses: (1) The abstraction parse generates the application abstraction graph (AAG) and synchronized application abstraction graph (SAAG). (2) The interpretation parse performs the actual interpretation using the interpretation algorithm. (3) The output parse generates the required performance metrics.

Abstraction Parse: The abstraction parse intercepts the SPMD program structure produced in phase 1 and abstracts its execution and communication structures to generate the corresponding AAG and SAAG (as defined in Section 3). A communication table is generated to store the specifications and status of each communication/synchronization.

The abstraction parse also identifies all critical variables in the application description; a critical variable being defined as a variable whose value effects the flow of execution, e.g. a loop limit. The critical variables are then resolved either by tracing their definition paths or by allowing the user to explicitly specify their values.

Interpretation Parse: The interpretation parse performs the actual performance interpretation using the interpretation algorithm. For each AAU in the SAAG, the corresponding interpretation function is used to generate the performance measure associated with it. Performance metrics maintained at each AAU are its computation, communication and overheads times, and the value of the global clock. In addition, cumulative metrics are also maintained for the entire SAAG.The interpretation parse has provisions to take into consideration a set of compiler optimizations (for the generated Fortran 77 + MP code) such as loop re-ordering, etc. These can be turned on/off by the user.

Output Parse The final parse communicates estimated performance metrics to the user. The output interface provides three types of outputs. The first type is a generic performance profile of the entire application broken up into its communication, computation and overhead components. Similar measures for each individual AAU and for sub-graphs of the AAG are also available. The second form of output allows the user to query the system for the metrics associated with a particular line (or a set of lines) of the application description. Finally, the system can generate an interpretation trace which can be used as input to the ParaGraph [6] visualization package. The user can then use the capabilities provided by the package to analyze the performance of the application.

4.3 Abstraction & Interpretation HPF/-Fortran 90D Parallel Constructs

The abstraction/interpretation of the HPF/Fortran 90D parallel constructs i.e. *forall*, array assignments, and *where* is described below:

forall Statement: The *forall* statement generalizes array assignments to handle new shapes of arrays by specifying them in terms of array elements or sections. The element array may be masked with a scalar logical expression. Its semantics are an assignment to each element or section (for which the mask expression evaluates true) with all the right-hand sides being evaluated before any left-hand sides are assigned. The order of iteration over the elements is not fixed. Examples of its use are:

$forall\ (I = 1 : N, J = 1 : N)\ P(I, J)\ =\ Q(I-1, J-1)$

$forall\ (I = 1 : N, Q(I).NE.0.0)\ P(I)\ =\ 1.0/Q(I)$

Phase 1 translates the *forall* statement into a three level structure consisting of a collective communication level, a local computation level and another collective communication level, to be executed by each processor. The processor that is assigned an iteration of the *forall* loop is responsible for computing the right-hand-side expression of the assignment statement, while the processors that owns an array element used in the left-hand side or right-hand side of the assignment statement must communicate that element to the processor performing the computation. Consequently, the first communication level fetches off-processor data required by the computation level. Once this data has been gathered, computations are local. The final communication level writes calculated values to off-processors.

Phase 2 then generates a corresponding sub-AAG using the application abstraction model. The communication level translates into a sequential (Seq(AAU corresponding to index translations and message packing performed, and a communication (Comm) AAU. The computation level generates an iterative (IterD) AAU which may contain a conditional (CondtD) AAU (depending on whether a mask is specified). The abstraction of the *forall* statement is shown in Figure 2. In this example, the final communication phase is not required as no off-processor data needs to be written.

Array Assignment Statements: HPF/Fortran 90D array assignment statements allow entire arrays (or array sections) to be manipulated atomically,

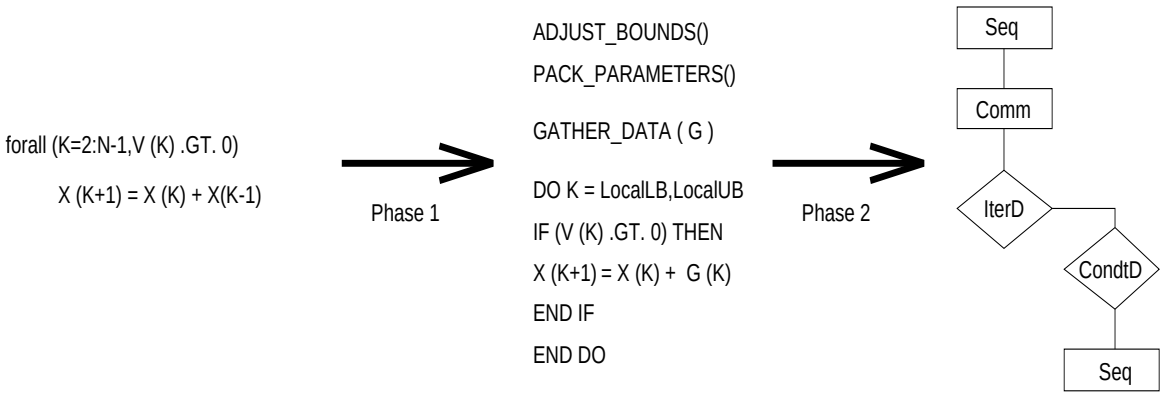

Figure 2: Abstraction of the *forall* Statement

thereby enhancing the clarity and conciseness of the program and making parallelism explicit. Array assignments are special cases of the *forall* statement and are abstracted by first translating them into equivalent *forall* statements. The resultant *forall* statement is then interpreted as described above.

where Statement: Like the array assignment statement, the HPF/Fortran 90D *where* statement is also a special case of the *forall* statement and is handled in a similar way.

4.4 Abstraction of the iPSC/860 System

Abstraction of the iPSC/860 hypercube system to generate the corresponding SAG was performed off-line using a combination of assembly instruction counts, measured timings, and system specifications. The processing and memory components were generated using system specification provided by the vendor, while iterative and conditional overheads were computed using instruction counts. The communication component was parameterized using benchmarking runs. These parameters abstracted both low-level primitives as well as the high-level collective communication library used by the compiler. Benchmarking runs were also used to parameterize the HPF parallel intrinsic library. The intrinsics included circular shift (*cshift*), shift to temporary (*tshift*), global sum operation (*sum*), global product operation (*product*), and the *maxloc* operation which returns the location of the maximum in a distribute array. Characterization of the SRM (host) and the communication channel connecting the SRM to i860 cube was performed in a similar manner.

5 Validation/Evaluation of the Interpretation Framework

In this section we present numerical results obtained using the current implementation of the HPF/Fortran 90D performance prediction framework. In addition to validating the viability of the interpretive approach, this section has the following objectives:

1. To validate the *accuracy* of the performance prediction framework for applications on a high performance computing system. The aim is to show that the predicted performance metrics are accurate enough to provide realistic information about the application performance and to be used as a basis for design tuning.

2. To demonstrate the utility of the framework and the metrics generated for efficient HPC application development. The results presented illustrate the framework's utility for: (1) Application design and directive selection; and (2) Application performance debugging.

3. To demonstrate the usability (ease of use) of the performance interpretation framework and its cost-effectiveness.

The high performance computing system used is an iPSC/860 hypercube connected to a 80386 based host processor. The particular configuration of the iPSC/860 consists of 8 i860 nodes. Each node has a 4 KByte instruction cache, 8 KByte data cache and 8 MBytes of main memory. The node operates at a clock speed of 40 MHz and has a theoretical peak performance of 80 MFlop/s for single precision and 40 MFlop/s for double precision. The validation application set was selected from the NPAC HPF/Fortran

Name	Description
	Livermore Fortran Kernels (LFK)
LFK 1	Hydro Fragment
LFK 2	ICCG Excerpt (Incomplete Cholesky; Conj. Grad.)
LFK 3	Inner Product
LFK 9	Integrate Predictors
LFK 14	1-D PIC (Particle In Cell)
LFK 22	Planckian Distribution
	Purdue Benchmarking Set (PBS)
PBS 1	Trapezoidal rule estimate of an integral of f(x)
PBS 2	Compute $e^* = \sum_{i=1}^{n} \prod_{j=1}^{m} \left(1 + \frac{0.5}{-\lvert i-j \rvert + 0.001}\right)$
PBS 3	Compute $S = \sum_{i=1}^{n} \prod_{j=1}^{m} a_{ij}$
PBS 4	Compute $R = \sum_{i=1}^{n} \frac{1}{x_i}$
PI	Approximation of π by calculating the area under the curve using the n-point quadrature rule
N-Body	Newtonian gravitational n-body simulation
Finance	Parallel stock option pricing model
Laplace	Laplace solver based on Jacobi iterations

Table 1: Validation Application Set

90D Benchmark Suite [7]. The suite consists of a set of benchmarking kernels and "real-life" applications and is designed to evaluate the efficiency of the HPF/Fortran 90D compiler and specifically, automatic partitioning schemes. The selected application set includes kernels from standard benchmark sets like the Livermore Fortran Kernels and the Purdue Benchmark Set, as well as real computational problems. The applications are listed in Table 1.

5.1 Validating Accuracy of the Framework

Accuracy of the performance prediction framework is validated by comparing estimated execution times with actual measured times. For each application, the experiment consisted of varying the problem size and number of processing elements used. Measured timings represent an average of 1000 runs. The results are summarized in Table 2. Error values listed are percentages of the measured time and represent maximum/minimum absolute errors over all problem sizes and system sizes. For example, the N-Body computation was performed for 16 to 4094 bodies on 1, 2, 4, and 8 nodes of the iPSC/860. The minimum absolute error between estimated and measured times was 0.09% of the measured time while the maximum absolute error was 5.9%.

The obtained results show that in the worst case, the interpreted performance is within 20% of the measured value, the best case error being less than 0.001%. The larger errors are produced by the benchmark kernels which have been specifically coded to task the compiler. Further, it was found that the interpreted performance typically lies within the variance of the measured times over the 1000 iterations. This indicates that the main contributors to the error are the tolerance of the timing routines and fluctuations in the system load. Predicted metrics typically serve either as the first-cut performance estimate of an application or as a relative performance measure to be used as a basis for design tuning. In either case, the interpreted performance is accurate enough to provide the required information.

5.2 Validating Utility of the Framework

The utility of the performance prediction framework is validated through the following experiments; (1) selecting the appropriate HPF/Fortran 90D directives based on the predicted performance, and (2) using the tool to analyze different components of the execution time and their distributions with respect to the application. These experiments are described below:

5.2.1 Appropriate Directive Selection

To demonstrate the utility of the interpretive framework in selecting HPF compiler directives we compare the performance of the Laplace solver for 3 different distributions (DISTRIBUTE directive) of the template, namely (BLOCK,BLOCK), (BLOCK,X) and (X,BLOCK), and corresponding alignments (ALIGN directive) of the data elements to the template. These three distributions (on 4 processors) are shown in Figure 3. Figures 4 & 5 compare the performance of each of the three cases for different system sizes using both, measured times and estimated times. These graphs can be used to select the best directives for a particular problem size and system configuration. For the Laplace solver, the (Block,X) distribution is the appropriate choice. Further, since the maximum absolute error between estimated and measured times is less than 1%, directive selection can be accurately performed using the interpretive framework. Using the interpretive framework is also significantly more cost-effective as will be demonstrated in Section 5.3.

In the above experiment, performance interpretation was source driven and can be automated. This exposes the utility of the framework as a basis for an intelligent compiler capable of selecting appropriate directives and data decompositions. Similarly, it can also enable such a compiler to select code optimizations such as the granularity of the computation phase

Name	Problem Sizes (data elements)	System Size (# procs)	Min Abs Error (%)	Max Abs Error (%)
LFK 1	128 - 4096	1 - 8	1.3%	10.2%
LFK 2	128 - 4096	1 - 8	2.5%	18.6%
LFK 3	128 - 4096	1 - 8	0.7%	7.2%
LFK 9	128 - 4096	1 - 8	0.3%	13.7%
LFK 14	128 - 4096	1 - 8	0.3%	13.8%
LFK 22	128 - 4096	1 - 8	1.4%	3.9%
PBS 1	128 - 4096	1 - 8	0.05%	7.9%
PBS 2	256 - 65536	1 - 8	0.6%	6.7%
PBS 3	256 - 65536	1 - 8	0.8%	9.5%
PBS 4	128 - 4096	1 - 8	0.2%	3.9%
PI	128 - 4096	1 - 8	0.00%	5.9%
N-Body	16 - 4096	1 - 8	0.09%	5.9%
Financial	32 - 512	1 - 8	1.1%	4.6%
Laplace (Blk-Blk)	16 - 256	1 - 8	0.2%	4.4%
Laplace (Blk-X)	16 - 256	1 - 8	0.6%	4.9%
Laplace (X-Blk)	16 - 256	1 - 8	0.1%	2.8%

Table 2: Accuracy of the Performance Prediction Framework

per communication phase in the loosely synchronous computation model.

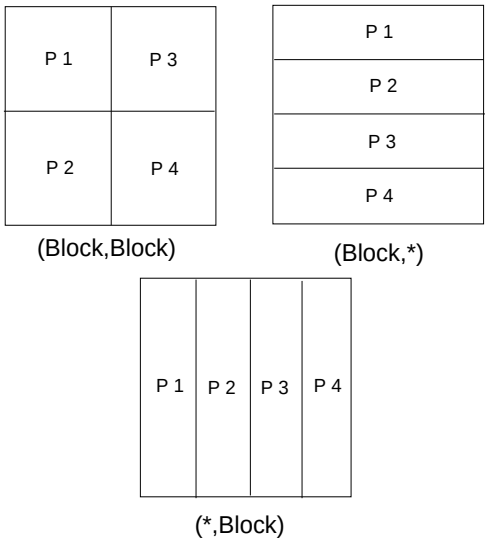

Figure 3: Laplace Solver - Data Distributions

5.2.2 Application Performance Debugging

The performance metrics generated by the framework can be used to analyze the performance contribution of different parts of the application description and to identify bottlenecks. A performance profile for the phases (Figure 6) of the parallel stock option pricing application is shown in Figure 7. Phase 1 creates the (distributed) option price lattice while Phase 2, which requires no communication, computes the call prices of stock options.

Application performance debugging using conventional means involves instrumentation, execution and data collection, and post-processing this data. Further, this process requires a running application and has to be repeated to evaluate each design modification. Using the interpretive framework, this information (at all levels required) is available during application development (without requiring a running application).

5.3 Validating Usability of the Framework

The interpreted performance estimates for the experiments described above were obtained using the interpretive framework running on a Sparcstation 1+. The framework provides a friendly menu-driven, graphical user interface to work with and requires no special hardware other than a conventional workstation and a windowing environment. Application characterization is performed automatically (unlike most approaches) while system abstraction is performed offline and only once. Application parameters and directives were varied from within the interface itself. Typical experimentation on the iPSC/860 (to obtained measured execution times) consisted of editing code, compiling and linking using a cross compiler (compiling on the front end is not allowed to reduce its load), transferring the executable to the iPSC/860 front end, loading it onto the i860 node and then finally running it. The process had to be repeated for each instance of each experiment. Relative experimentation times for different implementation of the Laplace Solver (Section 5.2.1) using measurements and the performance interpreter are shown in Figure 8. Experimentation using the interpretive approach required

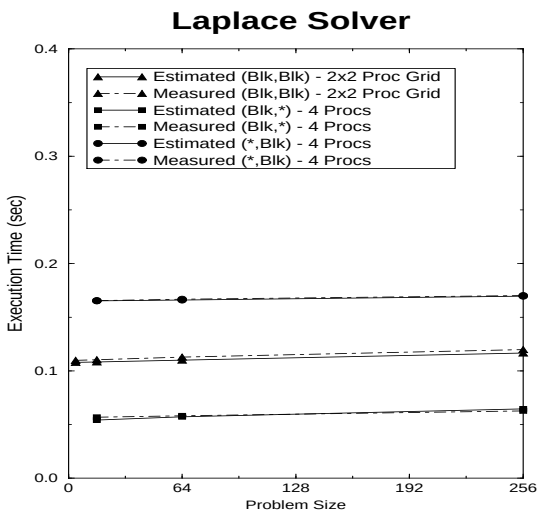

Figure 4: Laplace Solver (4 Procs) - Estimated/Measured Times

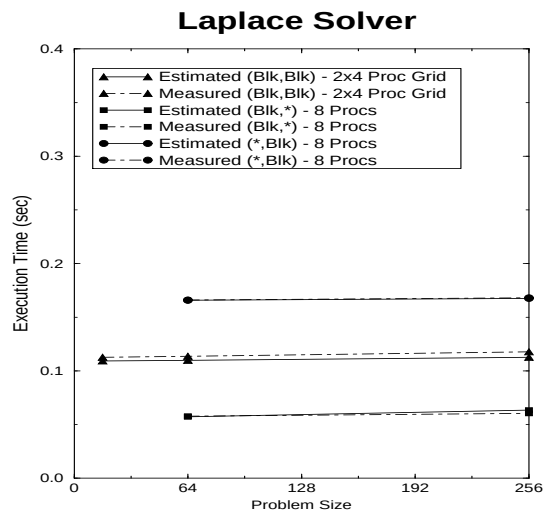

Figure 5: Laplace Solver (8 Procs) - Estimated/Measured Times

approximately 10 minutes for each of the three implementation. Experimentation using measurements however took a minimum 27 minutes (for the (Blk,*) implementation) and required almost 1 hour for the (*,Blk) case. Clearly, the measurements approach is not feasible, specially when a large number of options have to be evaluated. Further, the iPSC/860, being an expensive resource, is shared by various development groups in the organization. Consequently, its usage can be restrictive and the required configuration may not be immediately available. The comparison above validates the convenience and cost-effectiveness of the framework for experimentation during application development.

6 Related Work

Existing performance prediction approaches and models for multicomputer systems can be broadly classified as analytic, simulation, monitoring or hybrid (which make use of a combination of the above techniques along with possible heuristics and approximations)

Analytic techniques use mathematical models to abstract the system and application, and solve these models to obtain performance metrics. A general approach for analytic performance prediction for shared memory systems has been proposed by Siewiorek et al. in [8] while probabilistic models for parallel programs based on queueing theory have been presented in [9]. The above approaches require users to explicitly model the application along with the HPC system.

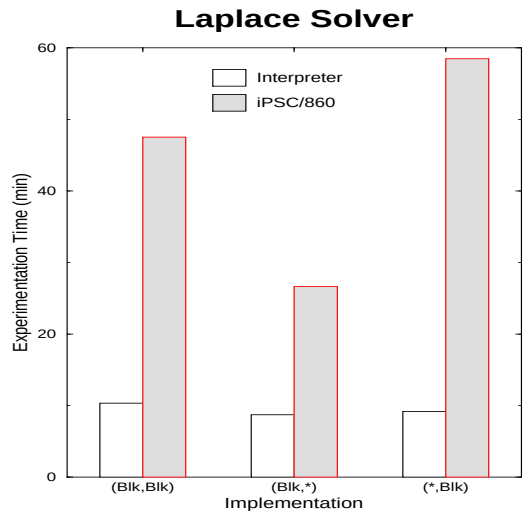

Figure 8: Experimentation Time - Laplace Solver

A source based analytic performance prediction model for Dataparallel C has been developed by Clement et al [10]. The approach uses the a set of assumptions and specific characteristics of the language to develop a speedup equation for applications in terms of system costs.

Simulation techniques simulate the hardware and the actual execution of a program on that hardware. These techniques are typically expensive in terms of the time and computing resource required. A simulation based approach is used in the SiGLe system (Simulator at Global Level) [11] which provides spe-

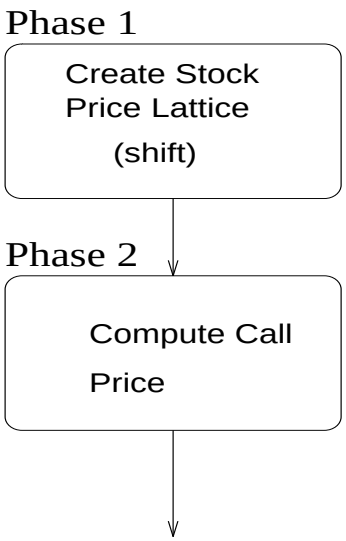

Figure 6: Financial Model - Application Phases

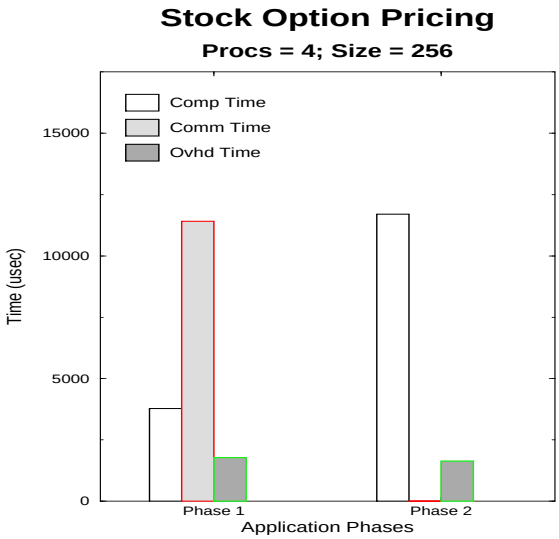

Figure 7: Financial Model - Interpreted Performance Profile

cial description languages to describe the architecture, application and the mapping of the application onto the architecture.

The PPPT system [12] uses monitoring techniques to profile the execution of the application program on a single processor. Obtained information is then used by the static parameter based performance prediction tool to estimate performance information for the parallelized (SPMD) application program on a distributed memory system. A similar evaluation approach based on instrumentation, data collection and post-processing has been proposed by Darema et al. [13]. Balasundaram et al. [14] use 'training routines" to benchmark the performance of the architecture and then use this information to evaluate different data decompositions.

A hybrid approach is presented in [15] where the runtime of each node of a stochastic graph representing the application is modeled as a random variable. The distributions of these random variables are then obtained using hardware monitoring.

The layered approach presented in [16] uses a methodology based on application and system characterization. The developer is required to characterize the application as an execution graph and define its resource requirements in this system.

7 Conclusions and Future Work

Evaluation tools form a critical part of any software development environment as they enable the developer to evaluate the different design choices available at various stages of application development, and to make the most appropriate selection.

In this paper, we described a novel interpretive approach for accurate and cost-effective performance prediction on high performance computing systems. A comprehensive characterization methodology is used to abstract the system and application components of the HPC environment into a set of well defined parameters. An interpreter engine then interprets the performance of the abstracted application in terms of the parameters exported by the abstracted system. A source-driven HPF/Fortran 90D performance prediction framework based on the interpretive approach has been implemented as part of the HPF/Fortran 90D integrated application development environment. The current implementation of the environment framework is targeted to the iPSC/860 hypercube system.

Numerical results using benchmarking kernels and application codes from the NPAC HPF/Fortran 90D Benchmark Suite, were presented to validate the accuracy, utility, and usability of the performance prediction framework. The use of the framework for selecting appropriate compiler directives, and for application performance debugging was demonstrated.

We are currently working on developing an intelligent HPF/Fortran 90D compiler based on the source based interpretation model. This tool will enable the compiler to automatically evaluate directives and transformation choices and optimize the application at compile time. Future development of the framework will involve moving it to high performance distributed computing systems and exploiting its potential as a

system design evaluation tool.

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] Manish Parashar, Salim Hariri, Tomasz Haupt, and Geoffrey C. Fox, "Design of An Interpretive Toolkit for HPF/Fortran 90D Application Development", Technical report, Northeast Parallel Architectures Center, Syracuse University, Syracuse NY 13244-4100, Apr. 1994.

[2] High Performance Fortran Forum, *High Performance Fortran Language Specifications, Version 1.0*, Jan. 1993, Also available as Technical Report CRPC-TR92225 from Center for Research on Parallel Computing, Rice University, Houston, TX 77251-1892.

[3] Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koebel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu, "Fortran D Language Specifications", Technical Report SCCS 42c, Northeast Parallel Architectures Center, Syracuse University, Syracuse NY 13244-4100, Dec. 1990.

[4] Manish Parashar, Salim Hariri, Tomasz Haupt, and Geoffrey C. Fox, "An Interpretive Framework for Application Performance Prediction", Technical Report SCCS-479, Northeast Parallel Architectures Center, Syracuse University, Syracuse NY 13244-4100, Apr. 1993.

[5] Zeki Bozkus, Alok Choudhary, Geoffrey Fox, Tomasz Haupt, and Sanjay Ranka, "Compiling HPF for Distributed Memory MIMD Computers", in David Lilja and Peter Bird, editors, *Impact of Compilation Technology on Computer Architecture*. Kluwer Academic Publishers, 1993.

[6] J. A. Etheridge M. Heath, "Paragraph", Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Oct 1991.

[7] A. Gaber Mohamed, Geoffrey C. Fox, Gregor von Laszewski, Manish Parashar, Tomasz Haupt, Kim Mills, Ying-Hua Lu, Neng-Tan Lin, and Nang kang Yeh, "Application Benchmark Set for Fortran-D and High Performance Fortran", Technical Report SCCS-327, Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY 13244-4100., June 1992.

[8] Dalibor F. Vrsalovic, Daniel P. Siewiorek, Zary Z. Segall, and Edward F. Gehringer, "Performance Prediction and Calibration for a Class of Multiprocessors", *IEEE Transactions on Computers*, **37**(11):1353–1365, Nov. 1988.

[9] A. Kapelnikov, R. R. Muntz, and M. D. Ercegovac, "A Methodology for Performance Analysis of Parallel Computations with Looping Constructs", *Journal of Parallel and Distributed Computing*, **14**:105–120, 1992.

[10] Mark J. Clement and Micheal J. Quinn, "Analytic Performance Prediction on Multicomputers", Technical report, Department of Computer Science, Oregon State University, Mar. 1993.

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

[12] Thomas Fahringer and Hans P. Zima, "A Static Parameter based Performance Prediction Tool for Parallel Programs", *Proceedings of the 7th ACM International Conference on Supercomputing, Japan*, July 1993.

[13] Frederica Darema, "Parallel Applications Performance Methodology", in Margaret Simmons, Rebecca Koskela, and Ingrid Bucher, editors, *Instrumentation for Future Parallel Computing Systems*, chapter 3, pp. 49–57. Addison-Wesley Publishing Company, 1988.

[14] Vasanth Balasundaram, Geoffrey Fox, Ken Kennedy, and Ulrich Kremer, "A Static Performance Estimator in the Fortran D Programming System", in Joel Saltz and Piyush Mehrotra, editors, *Languages, Compilers and Run-Time Environments for Distributed Memory Machines*, pp. 119–138. Elsevier Science Publishers B.V., 1992.

[15] Franz Sötz, "A Method for Performance Prediction of Parallel Programs", in H. Burkhart, editor, *Joint International Conference on Vector and Parallel Processing, Proceedings, Zurich, Switzerland*, pp. 98–107. Springer, Berlin, LNCS 457, Sep. 1990.

[16] E. Papaefstathiou, D. J. Kerbyson, and G. R. Nudd, "A Layered Approach to Parallel Software Performance Prediction: A Case Study", *Massively Parallel Processing Applications and Development, Delft*, 1994.