Performance Forecasting: Towards a Methodology for Characterizing Large Computational Applications*

Brian Armstrong Rudolf Eigenmann
School of Electrical and Computer Engineering
Purdue University

Abstract

We present a methodology that can identify and formulate performance characteristics of a computational application and uncover program performance trends on very large, future computer architectures and problem sizes. Based on this methodology we present "performance forecast diagrams" that predict the scalability of a large seismology application suite on a terabyte data set. We find that the applications scale well up to a large number of processors, given an interconnection network similar to the one of the SGI/Cray Origin architecture. However we find that if we increase the computation-to-communication speed ratio by a factor of 100, the different applications of the seismic suite start exhibiting architectural "sweet spots", at which the communication overhead starts to dominate computation time.

The presented methodology has proven to be useful in characterizing large computational applications. It is being applied in a project to create a repository of realistic programs and their characteristics.

1. Introduction

The motivation for the work presented in this paper is twofold. One of our long-term goals is to develop facilities that enable computer systems research teams to use large computational applications for evaluating and guiding their work. Such applications need to be characterized in meaningful ways. Methodologies for performing such characterizations are not well developed, and there is a lack of tools that help gather the necessary information from programs and their machine environments. In this paper we will contribute to the development of such methodologies and tools.

*This work was supported in part by U. S. Army contract DABT63-92-C-0033, NSF award ASC-9612133, and an NSF CAREER award. This work is not necessarily representative of the positions or policies of the U. S. Army or the Government.

A second goal is to develop next-generation computer architectures. We need methods for predicting performance trends of relevant applications on new machine concepts and system configurations. In related work we are using simulator tools to evaluate new architectures [5]. Simulators can perform detailed evaluations of small to mid-size programs. However, while it is already challenging to execute large-scope applications with large data sets on current high-performance machines (e.g, Dataset 6 requires 100 GB of disk space), the simulation of such programs on future architectures and problem sizes is not feasible. This paper develops methods to find performance trends where simulations are beyond reach.

For our project, we studied a large computational application suite used by the petroleum industry in the search for oil and gas. The application, called "Seismic," is also part of SPEChpc96, a benchmark suite of large-scope industrial applications [7]. Relatively little is known about the performance behavior of these codes. Our work will contribute to the characterization of this suite, hence facilitating its use for both our projects and those of related groups. We will make specific use of these results in our project to evaluate very-high-performance computer architectures that may be built within the next 10–20 years [4].

The specific contributions presented in this paper are

- a new method for describing and formulating performance trends of computational applications, parameterized by dataset sizes and variables of the underlying architectures,
- a description of tool technology that can help determine these trends,
- a discussion of the accuracy of these methods and their comparison with measured application timings on current computer systems,
- discussion of several "performance forecast diagrams" that show the behavior of the seismic pro-

cessing suite on future architectures under several "hardware assumptions."

Our work differs from related projects in several respects. Our goal is to develop methods and techniques that apply to large-scope computational applications. Many methods that apply to small and mid-range problems are not feasible for large programs and datasets. One example is the limitation of simulation methods, mentioned above. Second, for our performance prediction methods we make use of advanced symbolic program analysis techniques available in optimizing compilers. We attempt to determine program characteristics from the given application, even where other approaches may resort to user queries [3]. Third, our performance prediction is based on the computed volume of computation, communication, and I/O plus factoring in measured effective parameters of sample program runs. This contrasts with related approaches that use kernel benchmarks for determining computation and communication speeds [2] or that are based on counting the number of program statements [14]. It also contrasts with approaches that concentrate of the measurement of parallel overhead factors in order to predict extrapolated performance [6]. Fourth, an important goal of our prediction methodology is to capture performance trends for future computer architectures. This is different from and complementary to approaches that model performance with the goal of improving application speeds [1], capturing communication behavior [15], benchmarking current machines [8], and creating performance measurement tools [9, 12].

2. SPECseis96: A Seismic Processing Application Suite

In this paper we concentrate on the seismic processing suite SPECseis96. This suite is available as part of a benchmarking effort by the SPEC/High-Performance Group [7] for both machine benchmarking and scientific study. It is a code used in the petroleum industry for the prospecting of oil and gas, consisting of four applications, referred to as four "phases," which perform the seismic processes: *data generation, stacking of data, time migration*, and *depth migration*. Each of the four phases have distinct computation, communication, and disk IO characteristics. The entire suite contains 20,000 lines of Fortran and C and includes intensive communication as well as intensive disk IO.

Several datasets are available, ranging from a 17 MB dataset to sizes that are larger than current machines can handle (4 TB.) The data space consists of seismic samples in traces along a number of so-called lines [10]. Temporary disk space is required for the file of traces, which are records of signal amplitudes at specific 3D coordinates. The data size depends on the number of samples in a trace, the number of traces in a line, and the number of lines. The traces are collected into groups according to a traces-per-group parameter allowing the data to be passed between seismic processes as a two-dimensional array of traces.

The phases are designed to execute in sequence, though the third and fourth phases can be executed simultaneously. Within each phase, a main-loop performs a series of functions on each trace. Phases 2, 3, and 4 require data to be processed and passed across the processors throughout the phase. Phase 1 requires communication only to decompose at the start of the phase and join the data at the phase's completion. Each processor writes a disjoint segment of the data file allowing for writing to occur simultaneously among processors without blocking.

Communication is implemented using a message-passing layer. Sends and receives are blocking, hence the total time a processor spends communicating can be separated from the time it spends computing. Thus, both communication and disk IO can be captured using time-stamps at the beginning and end of the read and write functions. All time spent outside of the disk IO and communication routines is considered time spent in computation.

3. Methodology

We have defined analytical models for the components of the execution time, which relate the loads placed on the machine's resources with the code structures that create these loads. We simplify the loads placed on a machine's resources into three categories: computation, communication, and disk IO. Our breakdown of the execution time makes the simplifying assumption that the number and size of processor-to-processor messages and disk reads and writes contribute the majority of overheads. All other effects are not separated from the above three in our analysis. Specifically, our analysis only implicitly models the behavior of the cache. While the presented measurements will show good accuracy of our predictions, extending the models is an ongoing effort, which will increase the range of applicable programs.

3.1. Modeling Computation

The computation time is modeled for each loop in a program based on the execution time of the loop body and the number of loop iterations. The iteration number of a loop is expressed in terms of the application's input parameters and the number of processors. This allows us to scale the number of iterations a loop executes with respect to meaningful dataset and architecture parameters. The forecasted execution time of a

loop i ($FORE\_Time_{loop-i}$) is the number of times the loop is expected to execute ($FORE\_Calls_{loop-i}$) multiplied by an expression describing the average number of iterations the loop executes per call (named $FORE\_Iter_{loop-i}$) combined with the average *measured* time to execute a single iteration, (referred to as a base measurement, $BASE_{loop-i}$.) We obtain measured times by surrounding the loop with time stamps and summing the loop times exclusive of any time spent in inner-loops, for a specific program run. This sum ($MEAS\_Time_{loop-i}$,) is divided by the recorded number of iterations executed over the entire phase ($MEAS\_Iter_{loop-i}$,) to give a measured, average time per iteration of the loop. For *Seismic*, we used times from a run of Dataset 3a on four processors of the SGI/Cray Origin2000[1] as a base measurement.

$$BASE_{loop-i} = \frac{MEAS\_Time_{loop-i}}{MEAS\_Iter_{loop-i}}$$

$$FORE\_Time_{loop-i} = FORE\_Calls_{loop-i} \times FORE\_Iter_{loop-i} \times BASE_{loop-i}$$

3.2. Modeling External Resources

Loads on resources other than the CPU that are accessed using explicit external commands, (such as communication sends/receives and disk reads/writes,) are described by (1) their position within the loop structure, (2) the size of the data that the commands operate on, and (3) architectural parameters, such as the number of processors. Commands to external resources are located within the loop structure of the application, yielding the expression that describes the number of times these commands occur during program execution. We only consider explicit communication and disk IO commands.

External commands also have parameters that describe the size of the data they operate on, (the size of the message or of the read/written data.) These parameters are expressed in terms of the application and the program's input parameters, just like the loop range expressions. The time taken to access an external resource over the course of the program is defined using the function for the number of commands which access this resource and the size of the data accessed by each command.

The performance of the interconnect and the disk are greatly affected by their runtime environment. Since we are approaching performance from a static perspective, we do not simulate or measure dynamic events, which make up such performance issues as communication contention. Yet, we can grasp the volume of explicit disk accesses and communications. Our models combine characteristics of the code with the structure of the machine.

To account for the overhead of a message-passing layer or a file access, we use measured data as a basis. We determine the latencies used in our communication and disk IO performance models for each application phase separately. This gives us an effective communication latency, or a measure of the overhead in executing a communication call within a certain application (a phase of *Seismic*.)

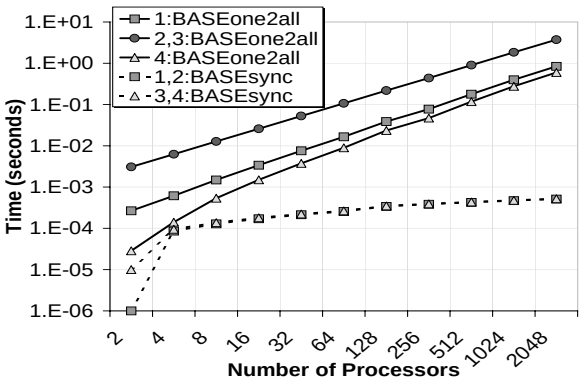

Figure 1. The base times for one-to-all broadcasts and barrier synchronizations are shown in this graph for each phase. The phase is given to the left side in the legend. These times are recorded in a lookup table for each phase. Then, a communication latency is calculated by the type of communication (all-to-all, synchronization, or master-slave gather,) and the number of processors.

Modeling Communication We include three latencies in our model, based on the SGI/Cray Origin hypercube-like interconnect:

$COMM_{su}$: a startup latency for a single communication command, obtained by calculating the average latency per message from measured data; a measure of the overhead for a communication call not including the size of the system.

$COMM_{hub}$: a Hub latency for a processor-to-processor connection, obtained by fitting forecasted to measured data.

$COMM_{router}$: a router latency for a hub-to-hub connection, also obtained by fitting forecasted to measured data.

Given a network topology, we sum the number of Routers ($One2All - Routers_p$) and Hubs ($One2All -$

[1] We used modi3, at the time, a 32-processor node of an SGI/Cray Origin2000 dedicated to single-user jobs, courtesy of NCSA at the University of Illinois.

Example: A forecast formula for the execution time of a loop is given below. *loop-273* in subroutine `vfill` steps through a vector, with a given stride, setting elements of the vector to a given constant. This loop is called to initialize the working array for accumulating a local image sum in order to propagate the wave field one depth step—part of the depth migration of the fourth phase of Seismic. The following values give the measured time and recorded number of iterations that *loop-273* executes during Phase 4 as well as the expressions used to forecast the number of calls and iterations of this loop.

$MEAS\_Time_{loop-273}$	4.643	milliseconds
$MEAS\_Iter_{loop-273}$	486,400	iterations
$FORE\_Calls_{loop-273}$	$1 + \lfloor \frac{ZMAX}{10} \rfloor$	calls
$FORE\_Iter_{loop-273}$	$NX \times NLINE$	iterations-per-call

These values and expressions are used to formulate our model of *loop-273*'s execution time.
$BASE_{loop-273} = \frac{0.004643\ seconds}{486{,}400\ iterations} = 9.55\ nanoseconds$
$FORE\_Time_{loop-273} = \left(1 + \lfloor \frac{ZMAX}{10} \rfloor\right) \times (NX \times NLINE) \times (9.55\ nanoseconds)$
This expression describing the execution time of *loop-273* can then be used to forecast how long the loop will execute for Dataset 8 by plugging in the values for the application-specific parameters accordingly: $ZMAX \rightarrow$ 10,000, $NX \rightarrow$ 2,168, $NLINES \rightarrow$ 1,024. The forecasted time for *loop-273* with Dataset 8, $FORE\_Time_{loop-273}$, is 21 seconds.

$Hubs_p$) that all the messages originating from a communication command executed on one processor must pass through. We describe how the latency of a communication command varies with the number of processors using these sums. Only two types of communication are done within *Seismic*: barrier synchronizations and all-to-all broadcasts.

The number of Hubs and Routers traversed by all messages sent in a one-to-all broadcast is determined for every number of processors (of a power of two) and placed in a lookup table. The base time for a one-to-all broadcast, ($BASE_{one2all-P}$,) is calculated from the number of Hub, Router, and startup latencies required to perform a one-to-all broadcast. $FORE\_Time_{all2all-i}$, the time for an all-to-all broadcast, is found by multiplying the base time by a factor of $2 \times (P-1)$, because the all-to-all communication command consists of a blocking send and receive with every other processor. Messages longer than a threshold are divided into multiple messages using the message size, $FORE\_Size_{all2all-i}$, and the maximum allowable message size, $COMM_{max-size}$, which we set to the 4 KB threshold of our MPI implementation. The time for one all-to-all broadcast is multiplied by the number of times this specific, explicit command is called within the seismic phase, labeled $FORE\_Calls_{all2all-i}$.

$$FORE\_Time_{all2all-i} = FORE\_Calls_{all2all-i} \times 2 \times (P-1) \times \left(\begin{array}{c} BASE_{one2all-P} \\ + \\ \lceil \frac{FORE\_Size_{all2all-i}}{COMM_{max-size}} \rceil \end{array} \right)$$

Barrier synchronizations take advantage of the hypercube structure to synchronize all processors in a tree fashion. Their latencies are calculated for every number of processors (of a power of two) by counting the number of Routers and Hubs traversed by the messages. The resulting model for the time of a synchronization command, ($FORE\_Time_{sync-i}$,) includes the number of times the specific synchronization command is executed, ($FORE\_Calls_{sync-i}$,) and the base time, ($BASE_{sync-P}$.)

$$FORE\_Time_{sync-i} = FORE\_Calls_{sync-i} \times BASE_{sync-P}$$

The base measurements for the all-to-all and synchronization commands for all four phases are graphed in Figure 1, showing how the measured averages affect the effective latencies.

Modeling Disk IO Our disk model considers explicit IO commands within the code. They are modeled as a startup latency, which depends on the logarithm of the number of processors, and a transmission latency per byte of the disk access. The startup and transmission latencies are calculated separately for each application phase and also separately for reads and writes from measured data. Separating latencies for reads from latencies for writes is significant in Phase 2, which reads from the disk file in large strides (using "transposed" reads where one sample is read from each trace.) The startup latency per read is the average time per read ($READ_{Startup}$) times the logarithm of the number of processors. The transmission latency per read ($READ_{Transmit}$) is the average time per byte of the read accesses. Latencies for writes are calculated in the same manner.

$$FORE\_Time_{read-i} = FORE\_Calls_{read-i} \times \left(\begin{array}{l} \\ + \end{array} \begin{array}{c} READ_{Startup} \times (\log P + 1) \\ \frac{FORE\_Size_{read-i}}{READ_{Transmit}} \end{array} \right)$$

Our communication and disk IO performance models represent simple approximations, corresponding to our goal of capturing important aspects of an application's performance behavior. Though our model cannot incorporate detailed runtime effects, it gives trends for the scalability of the features we can calculate statically and estimate in terms of application and machine-relevant parameters. The analytical models are used to determine how the loads produced by the application and the underlying machine configuration scale with the size of the dataset, structure of the data, and machine parameters.

4. Precision of Forecasts

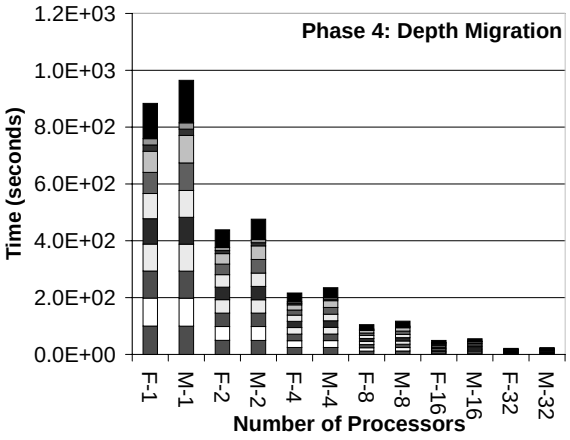

Figure 2. Loop-by-Loop Comparison of Forecasted to Measured Computation Times, dependent on Machine Size. These values are for Phase 4 of *Seismic* using Dataset 1, run on one 32 processor node of an SGI/Cray Origin 2000. The forecasts are compared to the measured times for 1,2,4,8,16, and 32 processors. The bars labeled "F-" correspond to the forecasts; the bars labeled "M-" are measured values. The top ten loops are given as different patterns and the black segment refers to the remaining loops. Discrepancies between our forecasts and what we measure can be localized within loop boundaries.

To verify the method we used in characterizing the performance of a large application suite we compared the scalability forecasted by our execution time model with actual measurements. Our model's scalability was tested as processors were added, keeping the data space

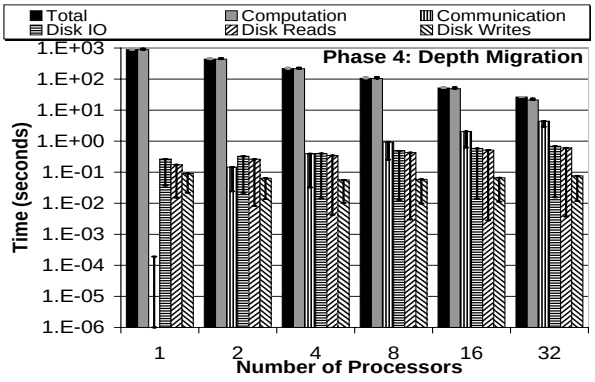

Figure 3. Comparison of Forecasted to Measured Times for the Components of Execution Time, dependent on Machine Size. These values are also for Dataset 1, run on a 32-processor node of an SGI/Cray Origin2000. Unlike the previous figure, here we include the total execution time broken down by component: computation, communication, disk reads, and disk writes. The graph shows results for the fourth seismic phase as the number of processors is increased from 1 to 32. The bars display the forecasted times of: Total Time, Computation Time, Disk IO Time, Disk Read Time, and Disk Write Time. The difference between the forecasted and measured times for each component are shown as error bars at the top of each bar. From the graphs we see that computation time is the most accurately modeled component.

constant. Some results of these tests are displayed in Figure 2 and Figure 3.

Since we predict the performance of each loop in the program individually, we can pinpoint the loops whose forecasts do not scale the same as our measurements. Figure 2 shows that for the majority of loops measured and predicted times agree well, while for a few loops there are discrepancies. Further refining of the performance expressions for the problematic loops can be done to make the forecasts more accurate. Cache effects and operating system latencies are indirectly included in the execution times for an iteration of each loop, which comes from measured data (Dataset 3a run on 4 processors.) While our tests using the first three datasets and a 32-processor machine show the predictions to be reasonably precise, improving the models is an ongoing project. Of particular interest is the more accurate formulation of cache effects and network contention.

The less predictable aspects of the *Seismic*'s performance are the times for communication and disk IO. Figure 3 shows a breakdown of Phase 4's execution time into time spent in computations, communications, and disk accesses. Separating computation time from time spent servicing external devices (communication

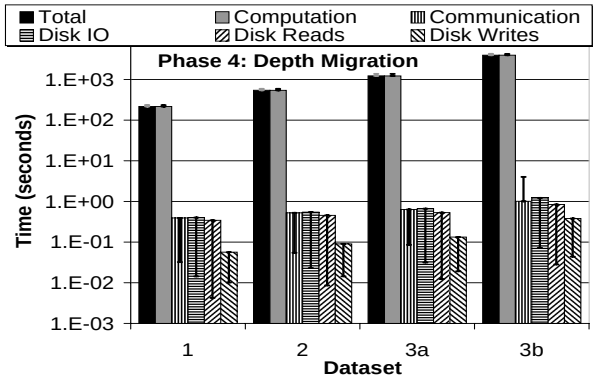

Figure 4. Comparison of Forecasted to Measured Times for the Components of Execution Time, dependent on Data Size. The dataset is varied with increasing dataset size; Dataset 1 requires 17 MB to store all the traces and Dataset 3b uses 137 MB to do so. In each case, four processors of a 32-processor SGI/Cray Origin2000 node are used. The graph shows results for the fourth seismic phase. The total execution time is broken down into four components: computation, communication, disk read, and disk write time. The difference between the forecasted and measured times for each component are shown as error bars at the top of each bar. Dataset 3a and Dataset 3b are very similar except for a few parameters: Dataset 3b has twice as many samples per trace and half as many groups per line. Also, Dataset 3b's x-y-z values for velocity sampling are larger than those of Dataset 3a's and its number of depth steps is increased by one half (this parameter only affects this seismic phase.)

and disk IO) reveals that the computation time can be accurately predicted by our simple model. The differences between the forecasted and measured times, seen in Figure 2, are consistent as the number of processors is increased—i.e., the error in the forecasts for loop execution times stays within tight bounds as the number of processors is increased.

We also varied the dataset size in Figure 4. The data space is increased from 17 MB to 137 MB. The data space is scaled consistently from Dataset 1 to Dataset 3a. Dataset 3b differs from Dataset 3a by keeping the overall amount of space required to store all traces relatively constant, increasing one dimension (the number of samples in a single receiver's trace) and decreasing another (the number of groups of traces in a line.) We include both Dataset 3a and Dataset 3b to investigate how well our forecasting model grasps changes to individual application parameters as opposed to increasing the data space in a consistent manner. The computation time does not vary much (relative to the times for Dataset 3a) for the first three phases despite this modification. However, Phase 4's time increases because Phase 4's computations are dependent upon the number of depth steps, which is 50% larger than with Dataset 3a.

5. Performance Forecast for *Seismic*

Given the methodology and tools introduced in the previous sections we now predict the performance of *Seismic* when using a very large dataset of several terabytes (Dataset 8.) The totals per phase are shown in Figure 5. The forecasts reveal that the computation time remains the major component of the execution time with large datasets, even when using up to 2,048 processors, meaning that *Seismic* is expected to perform very well under aggressive parallelization. Extrapolating for Dataset 8 distinguishes the major characteristics of the seismic processing phases.

- Phase 1 is highly parallel and performs a significant amount of disk writing.
- Phase 2 performs hours of communication as well as spends much time in disk IO. The majority of disk IO is spent reading from the disk.
- Phase 3 remains the shortest phase. Its communication depends upon two synchronizations. Its disk IO occurs within a matter of minutes.
- Phase 4 is computationally intensive, taking approximately ten times longer than the computations in Phase 2. The communication of Phase 4 is approximately ten times that of Phase 2, but their communication curves exhibit similar characteristics. The disk IO remains relatively constant, in minutes.

Current trends in processor technology favors CPU speed over communication speed; i.e., processor performance is increasing with new technology quicker than communication speed and the speed of disk accesses. To model these trends we scale up the computation speed by a factor of 100 without changing the communication performance. The forecast of *Seismic* under these hardware assumptions is given in Figure 6. The characteristics we can extract from this study are:

- Phase 1 and 4 benefit from higher processing power.
- The performance of Phases 1 and 2 are dependent upon the disk access latencies. However, Phase 1 requires twice as much computation and half the disk IO as that of Phase 2.
- Also, Phase 2's disk IO consists of reads while Phase 1's consists of writes.

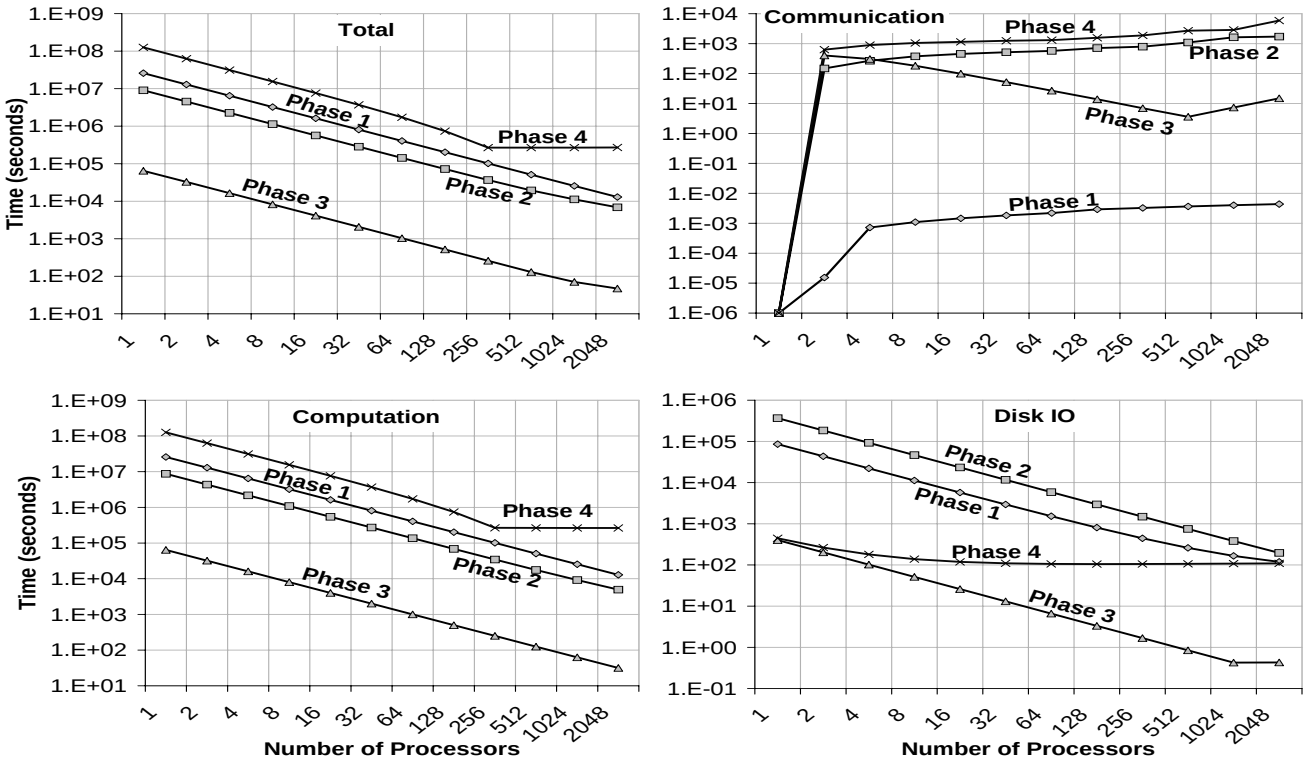

Figure 5. Forecasted Times for the Components of Execution Time, Extrapolating Machine Size. The forecasts are for Dataset 8 (which requires 4 terabytes to store all the traces.) The machine being forecasted is an SGI/Cray Origin2000 which allows configurations of up to 2,048 processors. Each graph displays the trend of each seismic phase for a component of the execution time as the number of processors is increased. These graphs show how intensive the computation, communication, and disk IO of the phases are in relation to each other. The communication of phases 2 and 4 occurs throughout the execution of these phases, which is why their communication patterns are similar. In contrast, phases 1 and 3 do a fixed number of communications for a given dataset, regardless of the number of processors. Using *transposed* reads (where a single trace is read from each group instead of every trace in a group being read sequentially) in Phase 2 makes its disk IO time the highest of all the phases.

- Phases 2 and 4 will eventually become communication dominated as more processors are added.
- Disk IO can be parallelized in all phases except in Phase 4, where it decreases to a limit around 32 processors.

The "sweet spots" in this figure are the minimum total execution times for each phase. Each phase has a sweet-spot after which its communication starts dominating the execution time. These points vary among the phases since the relative importance (in terms of time) of the execution time components differs across the phases.

6. Conclusions

We have developed a methodology for characterizing large computational applications in terms of their performance trends for large datasets and on large system configurations. We will use this methodology for analyzing and documenting a series of applications, which will be made available to other research groups. "Performance forecasting" is one of several methodologies for describing application programs such that research teams developing computer systems technology may find it easier to work with large, realistic applications and to do performance evaluation based on these programs. A repository that makes this information publicly available is being built [11].

Tools to support the described methodology in an automatic manner are partially available. Currently, several manual steps must be performed for creating performance diagrams as shown in this paper. In part, these manual steps compensate for current shortcomings of available tools, (such as limited compiler analysis techniques,) as well as combine the results of the

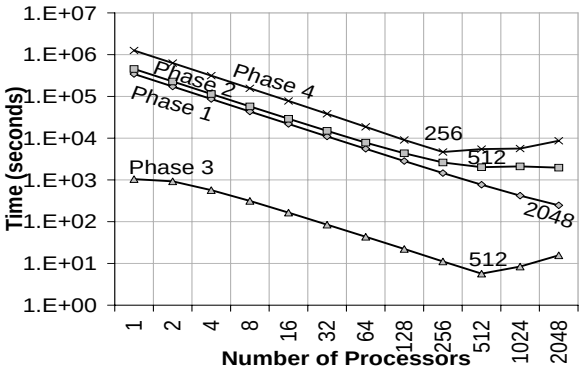

Figure 6. Forecasted Execution Time for Dataset 8 on a machine with Faster CPU Performance. The CPU performance is improved by 100 times, (roughly corresponding to 20-gigahertz CPU's,) and the totals for all four seismic processing phases are given in a single graph. The interprocessor network is the same as the hypercube network used by the SGI/Cray Origin2000 design. This models trends that CPU performance increases faster than communication performance.

various tools. Building an integrated environment is a long-term goal.

Using the described methodology we have identified performance trends of a large-scope seismic processing application suite. We have done this analysis using datasets that are larger than could be executed on any existing computer system. We have found that the applications scale well up to about 2000 processors, after which the current data partitioning scheme limits the available parallelism. When assuming a computation-to-communication speed ratio of 100 times the one of a SGI/Cray Origin2000 machine we have seen that the application begins to exhibit "sweet spots". These distinct, best-architecture points for the various seismic processing phases point out the key bottlenecks in each phase for architectures in which processing power dominates communication speed. Increasing the number of processors beyond these points will negatively impact the overall performance due to communication and IO overheads.

Extending the described models to a larger application class is an ongoing effort. *Seismic* is a regular application, for which it is relatively easy to determine and formulate the volumes of computation, communication and I/O. Furthermore, we have included cache behavior only implicitly, by factoring it into the overall computation speed. There is good agreement for our applications between predicted performance and the measurements up to 32 processors on the SGI/Cray Origin2000. Further studies with other applications are needed to demonstrate the applicable scope of the described methods and identify future extensions.

References

[1] G. Abandah and E. S. Davidson. Configuration independent analysis for characterizing shared-memory applications. In *12th Int'l. Parallel Processing Symp.*, Mar. 1998.

[2] G. A. Abandah and E. S. Davidson. Modeling the communication performance of the IBM SP2. 10th Int'l Parallel Processing Symp. (IPPS'96), Apr. 1996.

[3] V. Balasundaram, G. Fox, and et al. A static performance estimator to guide data partitioning decisions. 3rd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, pages 213–221, Apr. 1991.

[4] Z. Ben-Miled, R. Eigenmann, and et al. Hierarchical processors-and-memory architecture for high performance computing. In *Proc. of Frontiers'96 Conf.*, pages 355–362, Oct. 96.

[5] Z. Ben-Miled, J. A. B. Fortes, R. Eigenmann, and et al. A simulation-based cost-efficiency study of hierarchical heterogeneous machines for compiler and hand parallelized applications. *9th Int. Conf. on Par. and Dist. Computing and Systems*, pages 168–175, Oct. 1997.

[6] M. E. Crovella and T. J. LeBlanc. Parallel performance prediction using lost cycles analysis. In IEEE, editor, *Proc., Supercomputing '94: Washington, DC, November 14–18, 1994*, Supercomputing, pages 600–609. IEEE Computer Society Press, 1994.

[7] R. Eigenmann and S. Hassanzadeh. Benchmarking with real industrial applications: The SPEC High-Performance Group. *IEEE Computational Science & Engineering*, 3(1):18–23, Spring 1996.

[8] J. Gustafson and Q. Snell. HINT: A new way to measure computer performance. In *Proc. of the Twenty-eight Annual Hawaii Int'l Conf. on System Sciences*, volume II, pages 392–401, 95.

[9] B. P. Miller, M. D. Callaghan, and et al. The Paradyn parallel performance measurement tools. *IEEE Computer*, 28(11), Nov. 1995.

[10] C. C. Mosher and S. Hassanzadeh. ARCO seismic processing performance evaluation suite, User's Guide. ARCO, Plano, TX, 1993.

[11] I. Park and R. Eigenmann. Ursa Major: Exploring Web technology for design and evaluation of high-performance systems. In *Int'l Conf. on High-Performance Computing and Networking, HPCN Europe'98*, pages 535–544, Apr. 98.

[12] D. A. Reed. Experimental performance analysis of parallel systems: Techniques and open problems. In *7th Int' Conf on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 25–51, 1994.

[13] R. H. Saavedra and A. J. Smith. Performance characterization of optimizing compilers. *IEEE Trans. on Software Engineering*, 21(7):615–628, July 1995.

[14] R. H. Saavedra and A. J. Smith. Analysis of benchmark characteristics and benchmark performance prediction. *ACM Trans. on Comp. Sys.*, 14(4), Nov. 1996.

[15] M. R. Steed and M. J. Clement. Performance prediction of PVM programs. 10th Int'l Parallel Processing Symp., Honolulu, HI, Apr. 1996, pages 803–807.