

ornl

**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA



3 4456 0366074

ORNL/TM-11900

**Speedup Properties of Phases in
the Execution Profile of
Distributed Parallel Programs**

B. M. Carlson
T. D. Wagner
L. W. Dowdy
P. H. Worley

OAK RIDGE NATIONAL LABORATORY
CENTRAL RESEARCH LIBRARY
CIRCULATION SECTION
4506N ROOM 175

LIBRARY LOAN COPY

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this
report, send in name with report and
the library will arrange a loan.

UCRL7889 (2-1972)

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOD and DOD contractors from the Office of Scientific and Technical Information, P.O. Box 909, Oak Ridge, TN 37831; prices available from (615) 576-8401. DTIC price code 1.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5200 Port Royal Drive, Springfield, VA 22161.
NTIS price code 000-1. DTIC price code 1.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Engineering Physics and Mathematics Division
Mathematical Sciences Section

**SPEEDUP PROPERTIES OF PHASES IN THE EXECUTION PROFILE OF
DISTRIBUTED PARALLEL PROGRAMS**

Brian M. Carlson •
Thomas D. Wagner †
Lawrence W. Dowdy †
Patrick H. Worley ‡

• Computer Systems Research Institute
University of Toronto
Toronto, Ontario M5S1A1
Canada

† Computer Science Department
Vanderbilt University
Box 1679, Station B
Nashville, TN 37235

‡ Oak Ridge National Laboratory
Mathematical Sciences Section
P. O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Date Published: August 1992

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400



Contents

1	Introduction	1
2	Phases of the Execution Profile	2
2.1	Smoothing	4
2.1.1	Moving Averages	6
2.1.2	Rounding	7
2.1.3	Piecewise Polynomial Fits	9
2.2	Identification of Phases	10
3	Speedup of Phases	11
3.1	Execution Signatures	13
4	Conclusions	14
5	References	16

SPEEDUP PROPERTIES OF PHASES IN THE EXECUTION PROFILE OF DISTRIBUTED PARALLEL PROGRAMS

Brian M. Carlson
Thomas D. Wagner
Lawrence W. Dowdy
Patrick H. Worley

Abstract

The execution profile of a distributed-memory parallel program specifies the number of busy processors as a function of time. Periods of homogeneous processor utilization are manifested in many execution profiles. These periods can usually be correlated with the algorithms implemented in the underlying parallel code. Three families of methods for smoothing execution profile data are presented. These approaches simplify the problem of detecting end points of periods of homogeneous utilization. These periods, called phases, are then examined in isolation, and their speedup characteristics are explored. A specific workload executed on an Intel iPSC/860 is used for validation of the techniques described.

1. Introduction

An important question in performance evaluation is – “how will code that exists on a particular system behave in a different environment?” This question might be asked when adding a piece of hardware, such as a math coprocessor, a vector processor, more cache, or additional disk capacity. One might also consider the more complex problem of moving the code to a different type of hardware – perhaps from a conventional architecture to a RISC machine, or from a shared-memory to a distributed-memory multiprocessor. These constitute difficult modeling problems.

A related problem is taking a parallel application, measuring its performance, and then predicting how well it might utilize more processors. The problem is that of predicting scalability. Amdahl recognized that the portion of a parallel application that is inherently sequential limits the amount of gain that is possible by an unlimited number of processors [1]. More recently, lower and upper bounds have been obtained on both the speedup of an application and how efficiently additional processors may be used. Those results emphasize the importance of *average parallelism* – the average number of busy processors through the life of the application [3]. The number of processors utilized as a function of time (called the *parallelism profile*) is an indicator of the inherent parallelism of an application [3], [16]. The parallelism profile assumes that the number of processors exceeds the maximum parallelism of the system. This approach to predicting scalability is most natural in a shared-memory system where a task graph may be executed dynamically. In a distributed-memory environment, the placement of tasks and the number of processors in the system determine program behavior. Consequently, since re-assigning tasks to processors requires non-trivial overhead, determining the proper number of processors to assign to each application is a more reasonable thing to do when studying the behavior of an algorithm in a distributed-memory environment.

The number of busy processors, as a function of time, given a fixed number of available processors is known as the *execution profile* of a system [9]. This study identifies and analyzes the phases of a parallel program that are expressed in the execution profile. The methodology is both descriptive and empirical. Attention is focused on the *average parallelism* of a program in a phase and on the variance within that phase. The *execution signature*¹ is used as the basis for modeling the scalability of a phase in a parallel program [2], [14]. It is argued that it is often possible to decompose a program into phases and study the performance of each phase independently. This is demonstrated for a specific parallel program.

Identification and study of the properties of phases of a parallel program’s execution is worthwhile for several reasons. For example, a large application must be checkpointed periodi-

¹The execution signature of a parallel program is the rate of execution expressed as a function of the number of processors allocated to the program. Section 3.1 will extend the discussion of execution signatures.

cally to aid recovery in the event of processor failure. Phase boundaries, as defined in the next section, are natural checkpoints since nondeterminism in the system is minimal at these times. Also, it may be desirable to suspend the execution of a parallel application at checkpoints to run a higher priority application. It may be appropriate to restart the application on a different number of processors by reallocating processors between phases. If it is more efficient to run the application on a different number of processors from phase to phase, information about phase properties could be used in building an *a priori* schedule off-line. This is particularly useful if the schedules are for real-time workloads. If a phase is identified as too slow, sequential, or unable to speedup adequately as more processors are added, then the corresponding code may be replaced. If a distributed program is to be repartitioned (*i.e.*, remapped to a different set or number of processors), the repartitioning should be done between phases of execution.

One difficulty in studying parallel applications is collecting information that is useful in building quantitative models without unduly perturbing the runtime behavior of the applications. A system that is useful in this regard is PICL (Portable Instrumented Communication Library)[6], [7], [20]. PICL can generate execution trace information on demand, with the volume and detail of the data controlled by the user. The data from PICL can be used to generate execution profiles, which are the object of this study. This study uses distributed code instrumented with PICL running on an Intel iPSC/860 hypercube system.

Section 2 describes phases of execution and mechanisms for smoothing the execution profile of a parallel program. Section 3 applies the discussion of Section 2 to a specific application code. Section 4 discusses open problems and further applications of this type of analysis.

2. Phases of the Execution Profile

The notion of a *phase* of a program is natural in both sequential and parallel environments. For example, a software engineer might design a numerical application from the following abstract specification.

Program *Linear-Solve*

- a. $X \leftarrow \text{Read-Input file}(\text{BigProblem})$
- b. $\text{LU-Decomposition-Algorithm}(X)$
- c. $\text{Back-Substitution-Algorithm}(X)$
- d. $\text{Write-Results}(X, \text{file}(\text{BigAnswer}))$

Of the specification, parts **a** and **d** are often sequential processes on multiprocessor systems, whereas part **b** may be highly parallelizable, and part **c** may be moderately parallelizable. (See [13] for a discussion of these algorithms and their scalability.) From a performance perspective

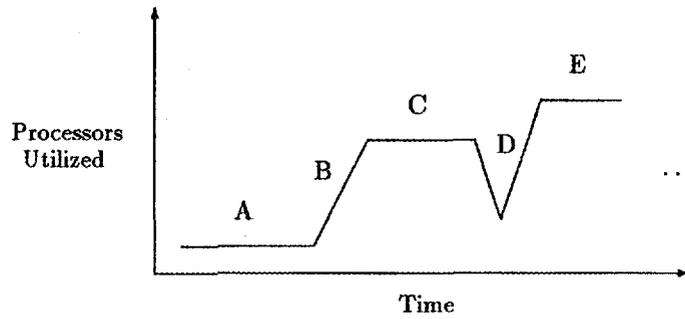


Figure 1: Idealized Execution Profile

there are three problems: 1) minimizing the sequential bottleneck, 2) getting maximum advantage from a multiprocessor when the number of available processors is less than the maximum parallelism of the application code, and 3) efficiently distributing the workload when the number of available processors exceeds the parallelism of the problem. Thus, the runtime behavior can be quite different within the different phases of an application.

In this section the problem of identifying various phases of an application from its execution profile is considered. Each phase of a parallel program has specific properties that distinguish it from other phases. The following definitions are given.

Definition 1 (Stationary Phase). A stationary phase, \mathcal{P}_s , is a subsequence of the execution profile, \mathcal{P} , that shares the same average parallelism and variance.

Definition 2 (Transitional Phase). A transitional phase, \mathcal{P}_t , is a subsequence of \mathcal{P} that bridges two stationary phases. In other words, a transitional phase represents portions of the execution profile that constitute an abrupt change in \mathcal{P} .

Consequence 1. It follows directly from definitions 1 and 2 that an ordered list of \mathcal{P}_s 's and \mathcal{P}_t 's, is a complete description of \mathcal{P} (the execution profile).

It is not always adequate to characterize the performance of a parallel code from decompositions of the execution profile into a sequence of stationary and transitional phases. Sometimes more complicated phase structures must be identified [18]. However, the decomposition into stationary and transitional phases is appropriate for many large scientific codes, especially those with an iterative structure.

Figure 1 illustrates the intent of these definitions. Phases A, C, and E are designated as stationary phases, whereas phases B and D are transitional phases. In many applications, phases are bounded by abrupt changes in utilization. In some cases, abrupt valleys reveal

barrier synchronization². In other cases, more shallow valleys denote changes in the underlying algorithm (*e.g.*, such as going from step b to step c in the earlier example). Unfortunately, stationary phases separated by shallow (or less abrupt) transitional phases may not be clearly identifiable in the execution trace. This is because not all processors reach the transitional phase simultaneously. (If they did, the transitional phase would be of length zero and two stationary phases would be adjacent.) Also, a high variance in the number of utilized processors makes phases difficult to identify. For our purposes, we treat and refer to high variance as noise. While it is not noise in a traditional sense, it represents uninteresting detail that masks underlying trends in the execution profile. This interpretation motivates the techniques for extracting phase structure.

The identification of phases in an execution profile is useful for a number of reasons, as indicated earlier. However, the problem of automatic phase identification is computationally expensive. Often there is no *a priori* knowledge about the number of phases present in an execution profile, and the measured data contains sufficient noise that identification based solely upon local information in \mathcal{P} is not reliable. In this case, a simple search procedure is required. That is, a single phase in the execution profile is first assumed and some measure of goodness is determined. Then, two phases are assumed and a new measure of goodness is determined, etc. The proper number of phases is associated with the best goodness measure [18]. Finally, recognizing *transitional phases* and *stationary phases* effectively requires distinguishing between “transient” and “steady state” behavior in the system, which itself is a difficult task.

While work is continuing on algorithms for the automatic identification of phases, the human visual system is quite good at this task, particularly if the uninteresting high variance components have been removed. Since phase identification is just one step in a larger modeling effort that will probably never be completely automated, the modeler’s direct participation in the phase identification process is a natural option. In this context, the initial step in solving the phase identification problem is the removal of noise. A number of techniques involving polynomial fitting and approximation exist for filtering. This process of filtering is generally known as *smoothing*. Note that automatic techniques for phase identification often require that the data be smoothed first [11]. Smoothing is also useful in other aspects of performance analysis [10].

2.1. Smoothing

Figure 2 illustrates an actual execution profile as measured (*i.e.*, no smoothing). If the noise in the execution profile is removed, visual identification of phases can be simplified. All the

²These would be valleys in the execution profile that drop the parallelism to 1, where only a single processor is utilized.

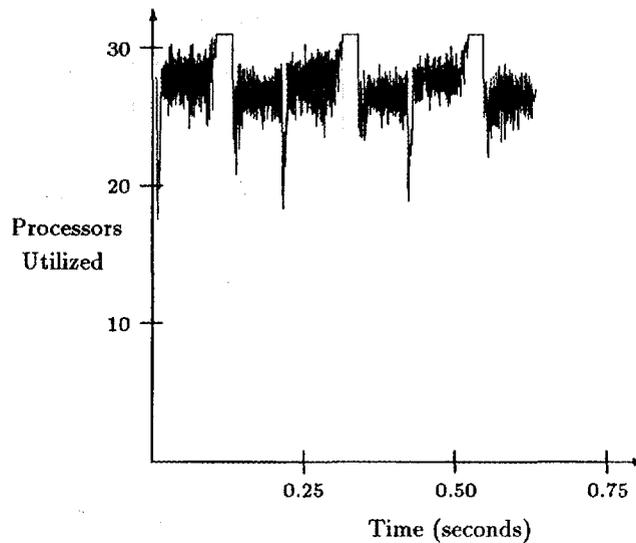


Figure 2: Sample Execution Profile with Distinct Phases

execution profiles shown in this section will use the data described by Figure 2 as input. This particular application is from a parallel implementation of a program that solves the nonlinear shallow water equations on the sphere using the spectral transform method [19]. This program constitutes an algorithmic kernel of a spectral global weather model.³ The specifics of the parallel program are briefly described below.

The spectral transform method used in this application code approximates the solution on both a tensor product grid representing physical coordinates and a triangular grid representing spectral coordinates. The computational kernel of the method is the transformation of the solution between the two representations, requiring both Legendre and Fourier transforms. The parallel implementation decomposes the two grids in such a way that all processors participate in a Legendre transform, but only one processor calculates any given Fourier transform. The resulting code is perfectly load balanced, with all inefficiencies due to the overhead of interprocess communication.

Our experience is that Figure 2 is representative of most execution profiles in that they exhibit phases that are quite apparent visually, but that can be difficult to isolate mathematically. Smoothing is used to help isolate the phases, both visually and algorithmically.

Definition 3 (Smoothing). *Smoothing is the process of removing noise from a sequence while preserving the coarse properties (in particular, transitional and stationary phases) of the original sequence.*

³The serial version of the program was provided by J. J. Hack at the National Center for Atmospheric Research. The distributed version was implemented by P. H. Worley at Oak Ridge National Laboratory.

Determining which aspects of the execution profile are noise can be a subjective process. In order to smooth out the noise automatically, that subjective process must be made an objective one. One method is to make smoothing an issue of determining meaningful granularity. If small local minima and maxima are important to the expression of a function, then removing noise may be detrimental. From Definitions 1 and 2 it is clear that small perturbations in the execution profile have little effect on the end points of a phase. Thus, a fairly coarse level of granularity is preferable in smoothing because it simplifies the problem when a fine level of granularity is not necessary. Numerous techniques are available to accomplish a suitable smoothing. Four computationally inexpensive approaches are presented.

2.1.1. Moving Averages

A common technique for studying data with noise (or, more generally, any experimental data set) is the method of least squares. In the method of least squares, the data is approximated by a linear function that minimizes the sum of squares of the differences between the function and the data [15]. In fitting a line to a collection of data, the following two equations are simultaneously satisfied [17]:

$$\begin{aligned} m \left(\sum_{i=1}^n x_i \right) + nb &= \sum_{i=1}^n y_i \\ m \left(\sum_{i=1}^n x_i^2 \right) + b \sum_{i=1}^n x_i &= \sum_{i=1}^n x_i y_i. \end{aligned}$$

Here, m is the slope and b is the y intercept of the line to be fit to the n observed points (x - y pairs).

In the context of smoothing, each data point is replaced by the value of the linear function that best fits the data in some small neighborhood. Without loss of generality, the local neighborhood of x_i 's may be transformed to be $[-k, -k+1, \dots, 0, \dots, k-1, k]$. Notice that this transformation reduces the two equations to a simpler form since now $\sum_{i=1}^n x_i = 0$. In order to smooth the point located at 0, it is necessary to solve for b in the above two equations. That is,

$$b = \frac{1}{n} \sum_{i=1}^n y_i,$$

which is the simple average of a neighborhood. Figure 3 shows the example data set smoothed using this method where the neighborhood size is 11.

The approach of moving averages generalizes the linear least-squares fit. Any smooth function may be approximated locally to an arbitrary degree of accuracy by a polynomial. The

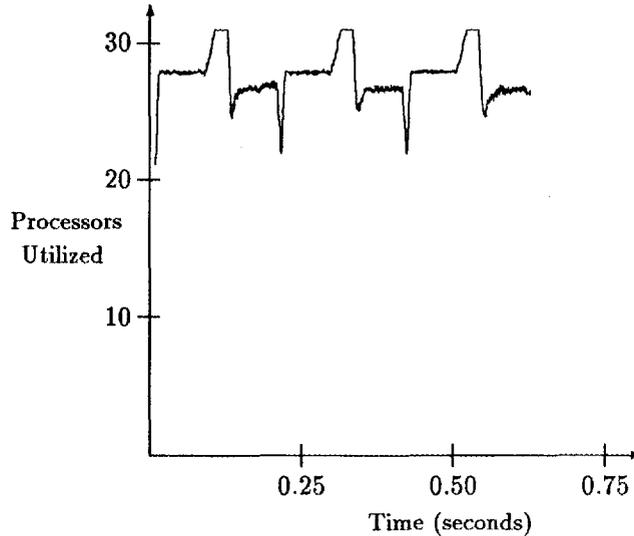


Figure 3: Execution Profile Smoothed by Least Squares

technique of moving averages exploits this fact by fitting $2k + 1$ points of a measured data set with a polynomial. The polynomial is then used to predict the $(k + 1)^{\text{st}}$ smoothed data value. (See [8] for an introductory exposition on moving averages and trend.) Figure 4 shows the smoothed version of Figure 2 using a cubic to fit a neighborhood of 11 points from the data set.

The moving averages strategy is a weighted average of neighboring points. Let f_t be the smoothed data value at time t in the execution profile. Let y_t be the actual observed utilization in the execution profile. The execution profile shown in Figure 4 is constructed as the weighted average of the example data set by:

$$f_t = \frac{1}{429}(-36 y_{t-5} + 9 y_{t-4} + 44 y_{t-3} + 69 y_{t-2} + 84 y_{t-1} + 89 y_t + 84 y_{t+1} + 69 y_{t+2} + 44 y_{t+3} + 9 y_{t+4} - 36 y_{t+5}).$$

2.1.2. Rounding

If the phases to be identified are extremely stratified (*i.e.*, far apart in average parallelism), then it is useful to round the average utilization. For example, each point in a real-valued function may be rounded to the nearest integer or 10^{th} or 100^{th} integer (*i.e.*, to its nearest rounding value). Figure 5 illustrates the execution profile of Figure 2 where utilization has been rounded to the nearest whole number. This technique, as shown in Figure 5, is of little help by itself if the variance within a phase exceeds the rounding value. It is more desirable in practice

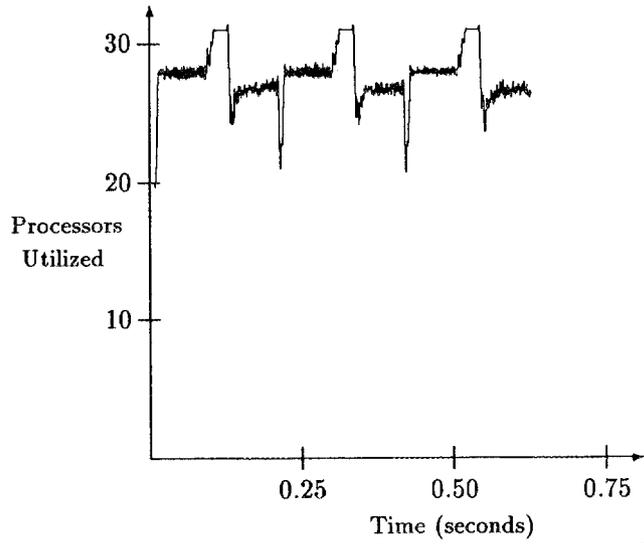


Figure 4: Execution Profile Smoothed by Moving Averages.

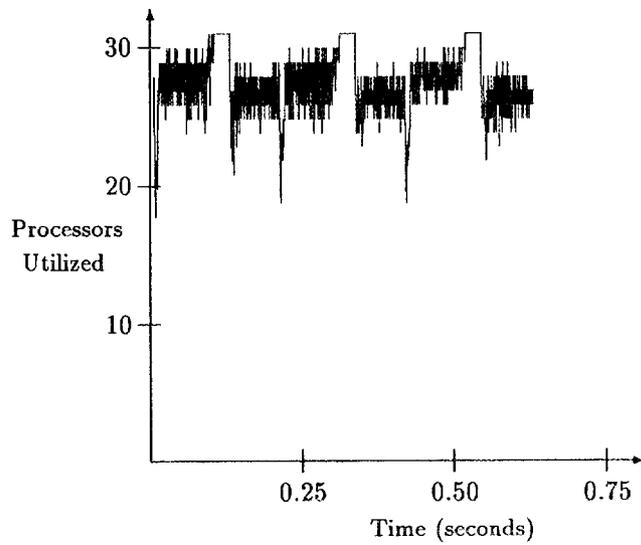


Figure 5: Execution Profile Smoothed by Simple Rounding

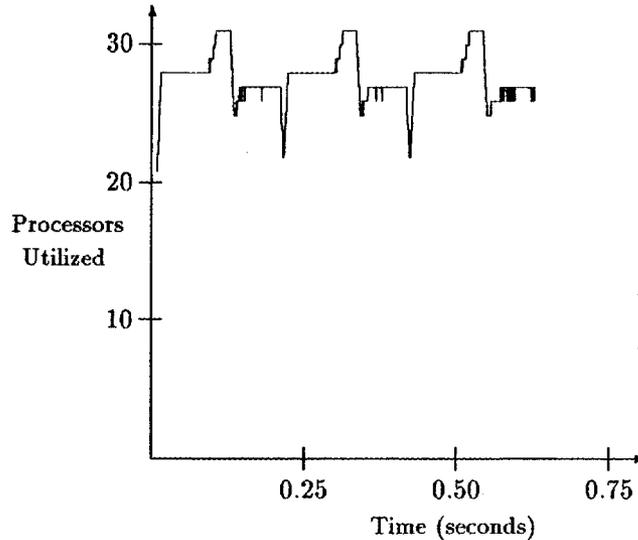


Figure 6: Execution Profile Smoothed by Rounding and Averaging Neighborhoods

to combine rounding with one of the other averaging techniques. Figure 6 combines rounding with moving averages.

2.1.3. Piecewise Polynomial Fits

Moving averages and rounding are both effective smoothing techniques for the example data set. However, since they use only local information, these techniques can sometimes smooth away important detail. In particular, a short transitional phase between stationary phases with similar mean and variance can be lost, leading to the merging of the three phases. An alternative is to calculate the optimal piecewise polynomial fit to the data, using a large number of pieces. In this approach, the data domain is partitioned into segments and the data is fit by polynomials in each segment separately. Both the endpoints of the segments and the individual fits are chosen to minimize the error in the approximation. Since this is a global optimization process, the algorithm is more robust when identifying transitions.

Calculating the piecewise polynomial that minimizes the least-squares error is prohibitively expensive. Since the endpoints of the segments, as well as the polynomials between the endpoints must be calculated, a large nonlinear least-squares problem must be solved. Fortunately, there are inexpensive algorithms for approximating the least-squares solution that are quite effective [4], [5]. A more robust solution is to calculate a piecewise polynomial that solves the *segment approximation problem* [12]. For this problem, the least-squares error between the data and the approximating function is calculated over each segment individually. The segments and fits are chosen to minimize this maximum *piecewise* error. This measure of the error is par-

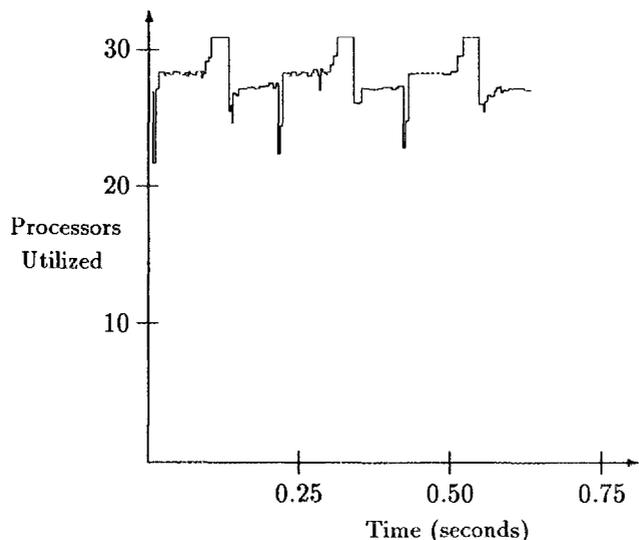


Figure 7: Execution Profile Smoothed by a Piecewise Constant Fit

ticularly suitable for the identification of phases in the processor profile, and a fast algorithm exists for computing the optimal fit [18].

Whereas, in moving averages, the amount of smoothing is determined by the size of the averaging neighborhood, the amount of smoothing when using a piecewise polynomial fit is a function of the number of pieces. For example, in the example data set there are 1157 data points. Thus, in a piecewise polynomial fit, using 105 pieces has similar smoothing properties to using an averaging neighborhood of 11 (since $1157/105 \approx 11$). Figure 7 shows the 105-piece piecewise constant function fit to the example data set of Figure 2.

The choice of which smoothing technique to use is data-dependent. For the execution profile used in this study, all four of the smoothing techniques discussed are effective. There are also many other smoothing techniques that could be used that are not discussed here. In the graphs and discussion that follow, the smoothing technique chosen is that of moving averages.

2.2. Identification of Phases

In this study, the identification of transitional phases and three special types of stationary phases are of interest. One special type of stationary phase is where utilization is equal to the number of allocated processors and the variance in the utilization over the phase is 0. These stationary phases typically scale almost linearly. For example, doubling the number of processors decreases the length of the phase by half. This type of phase represents intervals of pure computation with no interfering communication.

A second special type of stationary phase is where the average utilization is 1 (*i.e.*, only a

single processor is utilized) and the variance in utilization over the time interval is 0. These stationary phases correspond to the sequential portion of the parallel program. They represent intervals that exhibit no speedup as additional processors are allocated. The sum of the time periods of these stationary phases divided by the run time of the entire program is the *fraction sequential* of the program [1]. According to Amdahl's law, a program can not speedup more than the inverse of the fraction sequential of the program. Amdahl's definition of sequential is quite broad since a sequential portion of code could be arbitrarily short (*e.g.*, the bottom of a transitional phase). Phases, as defined here, are typically longer. It is assumed that sequential code occurs over an entire phase. From a practical perspective, this is not a limiting assumption.

The remaining stationary phases are considered to be of the same type. However, each phase has different scaling properties. That is, as additional processors are allocated, each phase is characterized by its own speedup curve. These phases are characterized by their average parallelism and the variance in parallelism. In the next section, the program shown in Figure 2 is characterized by these two parameters for 8, 16, 32, and 64 processors. The use of average parallelism to characterize parallel programs is not new. Both [3] and [16] study the properties of these parameters thoroughly for use in scheduling and in studying efficiency. Our approach differs by considering subsequences of the execution profile and aggregating them to make speedup predictions for the entire program.

Phase identification within a parallel program is important since phases appear to be associated with the parallel algorithm rather than with the number of processors allocated. That is, phases are often independent of the number of allocated processors. This implies that if one is able to associate a phase of the execution profile to some portion of the underlying algorithm, then it is possible to identify the parts of the execution profile that do not scale well, and replace them with algorithms that have better scaling properties. In the next section, speedup of phases is examined as the program is executed on a varied number of processors. Figure 8 illustrates the smoothed execution profile for 8, 16, 32, and 64 processors for the algorithm described in Section 2.

3. Speedup of Phases

As noted earlier, the identification of transitional phases is difficult. It is generally true, however, that transitional phases occur between two stationary phases. It is also true, by definition, that transitional phases are relatively short in duration. The peaks and troughs of transitional phases are easier to identify after smoothing. The approach taken in this study is to first identify the stationary phases that fully utilize the available processors. This can be done before smoothing. After smoothing using the moving averages technique, the peaks and troughs of the execution

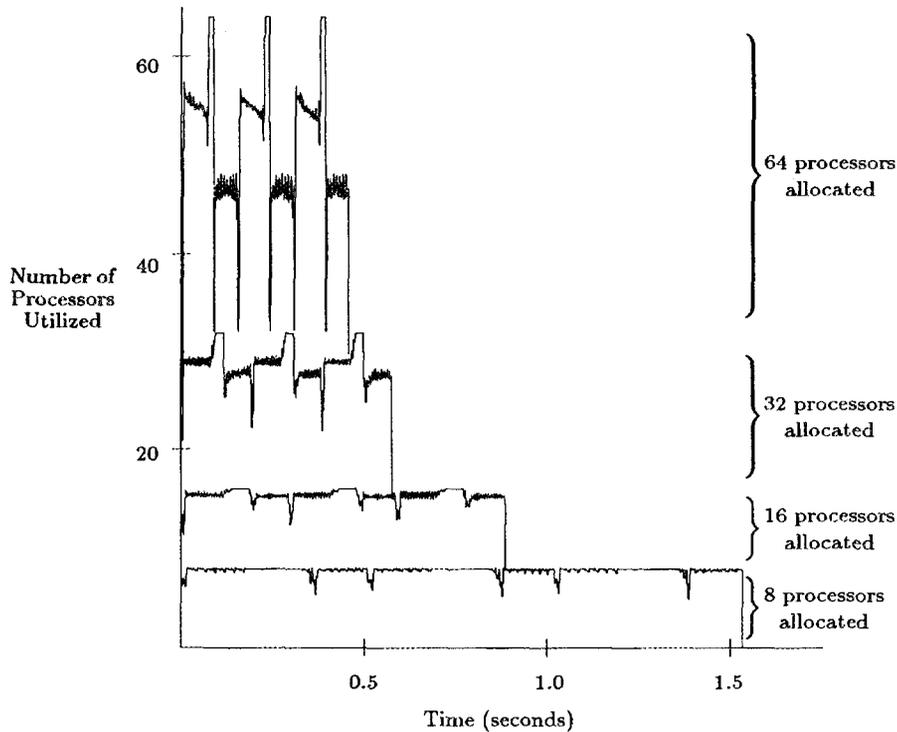


Figure 8: Smoothed Execution Profile for 8, 16, 32 and 64 Processors

profile are identified by ranking all local minima (maxima) by the average difference of each local minimum (maximum) with each data point in its neighborhood. The size of the neighborhood chosen depends upon the desired granularity. The local minima (maxima) with the highest rank are taken to be the troughs (peaks) of transitional phases. After the beginning and end points of stationary phases with full utilization and the peaks and troughs of the transitional phases are identified, the execution profile is then decomposed into phases. Phases are identified to begin at either a peak or trough of a transitional phase or at the end of a stationary phase with maximum utilization. Of course, the stationary phases with maximum utilization are retained.

Using the same application discussed in the previous sections, phases were identified as just described. The identification was carried out on *PICL* trace files of the application running on 8, 16, 32, and 64 processors. A total of 9 stationary phases were identified in each execution trace. The phases identified are shown in Figure 9. The phases are labeled, in order, \mathcal{P}_1 through \mathcal{P}_9 . The lengths of those phases, their mean utilizations, and their variance in utilization are tabulated in Table 1. Of the nine phases identified, phases \mathcal{P}_1 , \mathcal{P}_4 , and \mathcal{P}_7 have similar behavior. The same is true for phases \mathcal{P}_3 , \mathcal{P}_6 , and \mathcal{P}_9 . The similarities in each of those groups can be seen in the mean utilizations and variance. These similarities become more obvious as the number of processors is increased. Phases \mathcal{P}_2 , \mathcal{P}_5 , and \mathcal{P}_8 are the phases of full utilization.

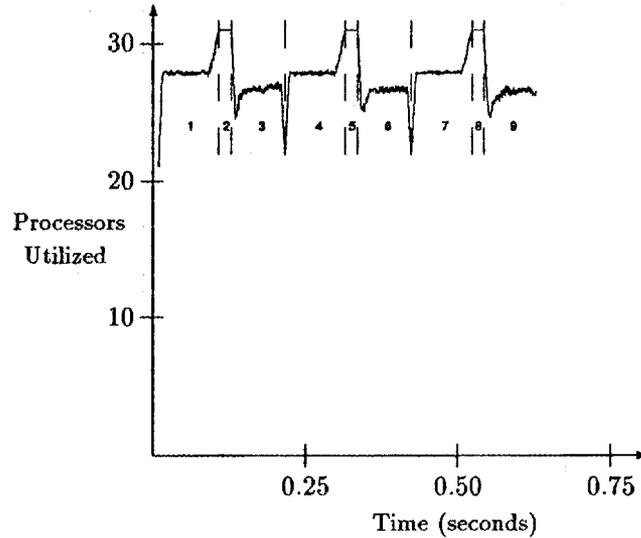


Figure 9: Phases Identified in a Smoothed Execution Profile

There is a repeating pattern whose first occurrence is the sequence of phases \mathcal{P}_1 , \mathcal{P}_2 , and \mathcal{P}_3 . Not surprisingly, this repeating pattern can be associated with the underlying structure of the parallel shallow water equation code.

3.1. Execution Signatures

Having identified the phases in this application, the speedup properties of the phases were examined. The characterization of speedup chosen for this study was the execution signature [2], [14].

Definition 4 (Execution Signature). *The execution signature of a (phase of a) parallel program is a function whose abscissa is the number of processors allocated, and whose ordinate is the rate at which the program executes in the absence of other programs.*

The runtime of the program may be obtained directly from the execution signature since runtime is the reciprocal of rate. The functional form of the execution signature E_i of parallel program i used is:

$$E_i(p_i) = \frac{P_i}{k_{i1}p_i + k_{i2}} \quad (1)$$

where p_i is the number of processors assigned to i . The parameters k_{i1} and k_{i2} are specific to the program and architecture. The functional form in Equation (1) passes through the point $(0,0)$ and is concave increasing as a function of p_i . As more processors are allocated, programs

Phase	Runtime(0.5ms)				Mean Utilization				Variance in Utilization			
	Processors				Processors				Processors			
	8	16	32	64	8	16	32	64	8	16	32	64
\mathcal{P}_1	447	270	185	149	7.85	15.29	28.70	53.74	0.39	0.84	2.17	6.21
\mathcal{P}_2	217	102	48	26	8.00	16.00	32.00	64.00	0.00	0.00	0.00	0.00
\mathcal{P}_3	379	222	154	136	7.71	15.16	27.60	45.78	0.50	0.61	1.52	4.52
\mathcal{P}_4	425	263	182	144	7.89	15.33	28.81	54.03	0.31	0.74	1.95	5.16
\mathcal{P}_5	218	99	46	25	8.00	16.00	32.00	64.00	0.00	0.00	0.00	0.00
\mathcal{P}_6	374	219	154	137	7.72	15.17	27.55	45.76	0.52	0.60	1.28	4.32
\mathcal{P}_7	430	271	184	143	7.88	15.32	28.81	54.00	0.40	0.82	1.79	5.11
\mathcal{P}_8	216	98	44	26	8.00	16.00	32.00	64.00	0.00	0.00	0.00	0.00
\mathcal{P}_9	303	224	153	133	7.77	15.18	27.41	46.07	0.51	0.61	1.34	4.22
Total	3009	1768	1150	919	7.85	15.38	28.65	51.21	0.40	0.71	2.12	7.26

Table 1: Length, Mean Utilization, and Variance for 9 Phases

Phase	Runtime(0.5ms)		Error
	Actual	Pred.	
\mathcal{P}_1	149	142	4.5%
\mathcal{P}_2	26	24	7.7%
\mathcal{P}_3	136	118	13.2%
\mathcal{P}_4	144	141	2.1%
\mathcal{P}_5	25	23	8.0%
\mathcal{P}_6	137	119	13.1%
\mathcal{P}_7	143	142	0.7%
\mathcal{P}_8	26	23	11.5%
\mathcal{P}_9	133	124	6.8%

Table 2: Actual and Predicted Runtimes and Error on 64 Processors

execute faster but incur added communication overhead and synchronization costs. Forms other than Equation (1) are possible as long as they possess these properties [14]. Using the reciprocals of the runtimes of the phases identified above for 8, 16 and 32 processors, execution signatures of the above form were calculated from a linear least-squares fit. (It is apparent from Table 1 that phases \mathcal{P}_2 , \mathcal{P}_5 , and \mathcal{P}_8 exhibit speedup behavior that is different from the other phases. For these three stationary phases with full utilization a linear execution signature was calculated, again using a least-squares fit.) Using these execution signatures, a prediction of the runtimes of all nine phases on 64 processors can be made. Figure 10 presents the actual execution signatures of each phase as solid lines. The dashed segment in each graph represents the predicted values for 64 processors. The predicted runtimes, along with the actual runtimes and the error, can be found in Table 2.

4. Conclusions

The execution profile of a distributed-memory multiprocessing system illustrates the number of busy processors in an application as a function of time. This study has observed that, within

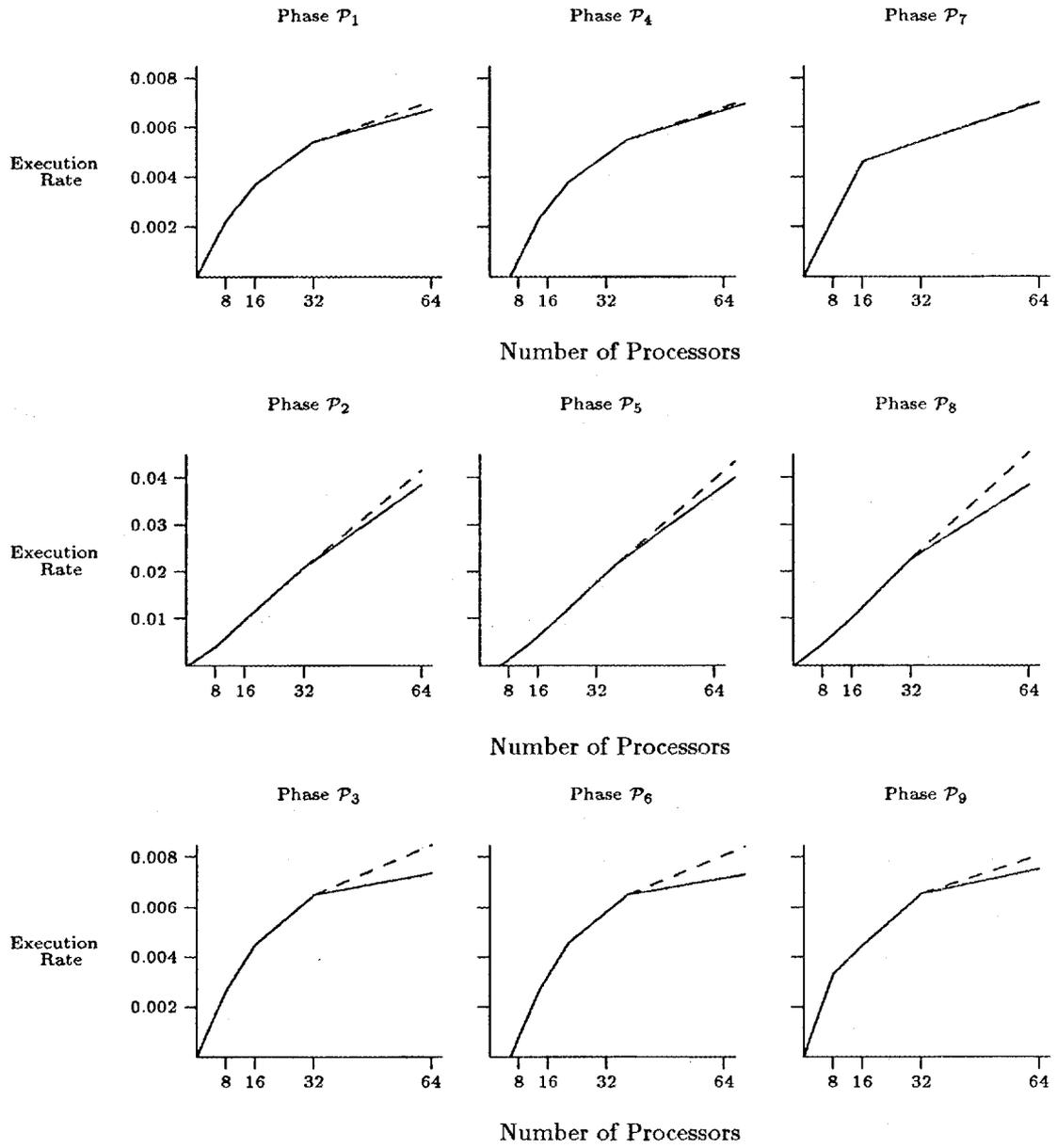


Figure 10: Execution Signatures of the 9 Phases

the execution profile, phases exist that are periods of roughly uniform processor activity separated by abrupt transitions in processor utilization. These phases of uniform processor activity (stationary phases) may be studied in isolation with respect to their speedup behavior. The resulting analysis may be aggregated, providing information regarding the speedup properties of the parallel application as a whole.

There are a number of potential applications for both the software designer as well as an adaptive operating system. For example, phases that are identified as those that scale very poorly may indicate portions of the program that should be redesigned. For an operating system, if remapping a program to a different number of processors is feasible, the execution profile identifies times in the application when remapping should occur (*i.e.*, during transitional phases). One can envision taking the four execution profiles of Figure 8, rearranging phases and constructing a new (reconfigurable) application that increases the efficiency of the system's processors. The identification of transitional phases also suggests times that the operating system might checkpoint at minimal cost (since the fewest number of busy processors would be interrupted).

One extension of this research is to compare the execution profile against communication volume as a function of time and attempt to detect phases given two dimensions instead of just one. This approach has been used successfully in a similar context [11]. Another extension is to incorporate variance into the execution signature as a method to minimize error in predicting phase speedup.

5. References

- [1] G. AMDAHL, *The validity of the single processor approach to achieving large scale computing capabilities*, AFIPS Conference Proceedings, 30 (1967), pp. 483-485.
- [2] L. W. DOWDY AND M. R. LEUZE, *On modeling partitioned multiprocessor systems*. under revision, 1991.
- [3] D. EAGER, J. ZAHORJAN, AND E. D. LAZOWSKA, *Speedup versus efficeincy in parallel systems*, IEEE Trans. Comput., 38 (1989), pp. 408-423.
- [4] W. D. FISHER, *On grouping for maximum homogeneity*, Journal of the American Statistical Association, 53 (1953), pp. 789-798.
- [5] J. H. FRIEDMAN AND B. W. SILVERMAN, *Flexibile parsimonious smoothing and additive modeling*, Technometrics, 31 (1989), pp. 3-21.

- [6] G. A. GEIST, M. T. HEATH, B. W. PEYTON, AND P. H. WORLEY, *PICL: a portable instrumented communication library, C reference manual*, Tech. Report ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.
- [7] ———, *A users' guide to PICL: a portable instrumented communication library*, Tech. Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.
- [8] M. KENDALL, *Time-Series*, Charles Griffith and Company, London, 1973.
- [9] M. KUMAR, *Measuring parallelism in computation intensive scientific/engineering applications*, IEEE Trans. Comput., 37 (1988), pp. 1088–1098.
- [10] T. J. LEBLANC, J. M. MELLOR-CRUMMEY, AND R. J. FOWLER, *Analyzing parallel program executions using multiple views*, J. Par. Dist. Comp., 9 (1990), pp. 203–217.
- [11] B. P. MILLER, M. CLARK, J. HOLLINGSWORTH, S. KIERSTEAD, S.-S. LIM, AND T. TORZEWSKI, *IPS-2: the second generation of a parallel program measurement system*, IEEE Trans. Par. Dist. Sys., 1 (1990), pp. 206–217.
- [12] G. NÜRNBERGER, M. SOMMER, AND H. STRAUSS, *An algorithm for segment approximation*, Numerische Mathematik, 48 (1986), pp. 463–477.
- [13] J. M. ORTEGA, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.
- [14] K.-H. PARK, *Dynamic Processor Partitioning for Multiprogrammed Multiprocessor Systems*, Ph.D. thesis, Vanderbilt University, Nashville, TN, August 1990.
- [15] M. B. PRIESTLEY, *Spectral Analysis and Time Series*, Academic Press, New York, 1981.
- [16] K. C. SEVCIK, *Characterization of parallelism in applications and their use in scheduling*, Performance Evaluation Review, 17 (1989), pp. 171–180.
- [17] G. THOMAS AND R. FINNEY, *Calculus and Analytic Geometry*, Addison-Wesley Publishing Company, Reading, Massachusetts, 5th ed., 1981.
- [18] P. H. WORLEY, *Modeling histogram data with piecewise polynomials*, Tech. Report ORNL/TM-11637, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.
- [19] P. H. WORLEY AND J. B. DRAKE, *Parallelizing the spectral transform method – part I*, Tech. Report ORNL/TM-11747, Oak Ridge National Laboratory, Oak Ridge, TN, February 1991.

- [20] P. H. WORLEY AND M. T. HEATH, *Performance characterization research at Oak Ridge National Laboratory*, in *Parallel Processing for Scientific Computing*, J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 431--436.

INTERNAL DISTRIBUTION

- | | |
|--------------------|--|
| 1. B. R. Appleton | 17-21. R. F. Sincovec |
| 2. E. F. D'Azevedo | 22-26. R. C. Ward |
| 3-4. T. S. Darland | 27-31. P. H. Worley |
| 5. J. J. Dongarra | 32. Central Research Library |
| 6. T. H. Dunigan | 33. ORNL Patent Office |
| 7. G. A. Geist | 34. K-25 Applied Technology Li-
brary |
| 8. M. R. Leuze | 35. Y-12 Technical Library |
| 9. C. E. Oliver | 36. Laboratory Records - RC |
| 10. G. Ostrouchov | 37-38. Laboratory Records Department |
| 11-15. S. A. Raby | |
| 16. T. H. Rowan | |

EXTERNAL DISTRIBUTION

39. Donald M. Austin, 6196 EECS Building, University of Minnesota, 200 Union Street, S.E., Minneapolis, MN 55455
40. Robert G. Babb, Oregon Graduate Center, CSE Department, 19600 N.W. Walker Road, Beaverton, OR 97006
41. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185
42. Adam Beguelin, Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890
43. Robert E. Benner, Parallel Processing Division 1413, Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185
44. Philippe Berger, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France
45. Donna Bergmark, 745 E & TC Building, Hoy Road, Cornell University, Ithaca, NY 14853
46. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street Cambridge, MA 02138
47. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
48. Greg Burns, Trollius Project Leader, Academic Computing, The Ohio State University, 1224 Kinnear Rd., Columbus, OH 43212
49. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

50. Maria Calzarossa, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrosso 209, I-27100 Pavia, Italy
- 51-55. Brian M. Carlson, Computer Systems Research Institute, University of Toronto, Toronto, Ontario M5S 1A1, Canada
56. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024
57. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709
58. Siddhartha Chatterjee, RIACS, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035-1000
59. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550
60. Alva Couch, Department of Computer Science, Tufts University, Medford, MA 02155
61. George Cybenko, Center for Supercomputing Research and Development, University of Illinois, 104 South Wright Street, Urbana, IL 61801-2932
62. Helen Davis, Computer Science Department, Stanford University, Stanford, CA 94305
63. Michel Dayde, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France
64. John J. Dorning, Department of Nuclear Engineering Physics, University of Virginia Reactor Facility, Charlottesville, VA 22901
65. Craig Douglas, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598-0218
- 66-70. Larry Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235
71. Iain S. Duff, Atlas Centre, Rutherford Appleton Laboratory, Chilton, Oxon OX11 0QX, England
72. Dannie Durand, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, FRANCE
73. Derek Eager, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
74. Stanley Eisenstat, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
75. Edward Felten, Department of Computer Science, University of Washington, Seattle, WA 98195
76. Charles Fineman, Ames Research Center, Mail Stop 269/3, Moffet Field, CA 94035
77. Jon Flower, Parasoft Corporation, 2500 E. Foothill Boulevard, Suite 205, Pasadena, CA 91107

78. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100
79. Chris Fraley, Statistical Sciences, Inc., 1700 Westlake Avenue N, Suite 500, Seattle, WA 98119
80. Joan M. Francioni, Computer Science Department, University of Southwestern Louisiana, Lafayette, LA 70504
81. Offir Frieder, George Mason University, Science and Technology Building, Computer Science Department, 4400 University Drive, Fairfax, Va 22030-4444
82. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
83. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401
84. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
85. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305
86. Eric Grosse, AT&T Bell Labs 2T-504, Murray Hill, NJ 07974
87. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020
88. Ann H. Hayes, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545
89. Michael T. Heath, National Center for Supercomputing Applications, 4157 Beckman Institute University of Illinois, 405 North Mathews Avenue, Urbana, IL 61801-2300
90. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305
91. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
92. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
93. Leah H. Jamieson, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907
94. Gary Johnson, Scientific Computing Staff, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
95. Lennart Johnsson, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214
96. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309
97. Malvyn Kalos, Cornell Theory Center, Engineering and Theory Center Building, Cornell University, Ithaca, NY 14853-3901

98. Alan H. Karp, HP Labs 3U-7, Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, CA 94304
99. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001
100. Michael Langston, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
101. Richard Lau, Office of Naval Research, 1030 E. Green Street, Pasadena, CA 91101
102. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709
103. E. D. Lazowska, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
104. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815
105. Ted Lewis, Research Director, Oregon Advanced Computing Inst., 19500 NW Gibbs Boulevard #101, Beaverton, OR. 97006
106. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
107. Ivo de Lotto, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrosso 209, I-27100 Pavia, Italy
108. Ewing Lusk, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, MCS 221 Argonne, IL 60439-4844
109. Allen D. Malony, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403
110. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550
111. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Boulevard, Pasadena, CA 91125
112. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801
113. Richard Muntz, Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90024
114. Jacek Myczkowski, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142
115. David Nelson, Director, Scientific Computing Staff, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
116. Randolph Nelson, IBM, P.O. Box 704, Room H2-D26, Yorktown Heights, NY 10598

117. Joseph Olinger, Computer Science Department, Stanford University, Stanford, CA 94305
118. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901
119. Steve Otto, Oregon Graduate Institute, Department of Computer Sci. & Eng., 19600 NW von Neumann Drive, Beaverton, OR 97006-1999
120. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
121. Peter C. Patton, Special Consulting Services, Inc., 1990-A Christensen Avenue, West St. Paul, MN 55118
122. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931
123. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801
124. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
125. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore National Laboratory, Livermore, CA 94550
126. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
127. Diane T. Rover, 155 Engineering Building, Dept. of Electrical Engineering, Michigan State University, East Lansing MI 48824
128. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455
129. Joel Saltz, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-522
130. Jorge Sanz, IBM Almaden Research Center, Department K53/802, 650 Harry Road, San Jose, CA 95120
131. Robert B. Schnabel, Department of Computer Science, University of Colorado at Boulder, ECOT 7-7 Engineering Center, Campus Box 430, Boulder, CO 80309-0430
132. Robert Schreiber, RIACS, MS 230-5, NASA Ames Research Center, Moffet Field, CA 94035
133. James L. Schwarzmeier, Cray Research, Inc., 900 Lowater Road, Chippewa Falls, WI 54729
134. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
135. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881

136. Charles L. Seitz, Department of Computer Science, California Institute of Technology, Pasadena, CA 91125
137. Giuseppe Serazzi, Dipartimento di Scienze della Informazione, Università di Milano, Via Moretto da Brescia 9, I20133 Milano, Italy
138. Kenneth C. Sevcik, Computer Systems Research Institute, 10 King's College Road, University of Toronto, Toronto, Ontario M5S 1A1, Canada
139. Margaret L. Simmons, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545
140. Horst D. Simon, NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035
141. Tony Skjellum, Lawrence Livermore National Laboratory, L-316, P. O. Box 808, Livermore, CA 94550
142. Burton Smith, Tera Computer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103
143. Marc Snir, IBM T.J. Watson Research Center, Department 420/36-241, P. O. Box 218, Yorktown Heights, NY 10598
144. Larry Snyder, Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195
145. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
146. Steven Suhr, Computer Science Department, Stanford University, Stanford, CA 94305
147. Wei Pai Tang, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
148. Bernard Tourancheau, LIP, ENS-Lyon, 69364 Lyon cedex 07, France
149. Mary Vernon, Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706
150. Robert G. Voigt, National Science Foundation, Room 417, 1800 G Street N.W., Washington, DC 20550
151. Michael D. Vose, 107 Ayres Hall, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
152. Phuong Vu, Cray Research, Inc., 19607 Franz Road, Houston, TX 77084
- 153-157. Thomas Wagner, Computer Science Department, Vanderbilt University, Nashville, TN 37235
158. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251
159. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545

160. John Zahorjan, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
161. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600
- 162-171. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831