Models of Parallel Applications with Large Computation and I/O Requirements

Emilia Rosti, *Member*, *IEEE*, Giuseppe Serazzi, *Member*, *IEEE Computer Society*,
Evgenia Smirni, *Member*, *IEEE Computer Society*, and Mark S. Squillante, *Senior Member*, *IEEE*

Abstract—A fundamental understanding of the interplay between computation and I/O activities in parallel applications that manipulate huge amounts of data is critical to achieving good application performance, as well as correctly characterizing the workloads of large-scale high-performance parallel systems. In this paper, we present a formal model of the behavior of CPU and I/O interactions in scientific applications, from which we derive various formulas that characterize application performance. Our model captures the I/O and CPU activity at different levels of granularity, where results from the model are shown to be in excellent agreement with measurement data from a set of I/O-intensive applications. Using the formulas from our model, which explicitly take I/O activity into account, we also present examples of possible applications of the model.

Index Terms—Parallel programs, workload characterization, performance modeling and analysis, performance prediction.

1 Introduction

Scientific and commercial applications that manipulate vast amounts of data have recently emerged as one of the popular consumers of cycles in parallel systems. With the nature of the problems to be solved requiring larger and larger memories and with input/output (I/O) resources belonging to the slowest level of the memory hierarchy, there is an increasing number of scientific and commercial applications that rely heavily on high-performance I/O systems for their execution [33]. As an initial response to this need, parallel file systems and their companion architectures and software support have been the subject of much research and development in recent years [18], [46], [48], [51]. In order to improve the performance of such systems, a range of different solutions and design principles must be considered. However, the cost and time necessary to explore a range of new alternatives are often prohibitive. Formal models and analysis can therefore play an important role in comparing and evaluating various alternatives, possibly complementing more detailed system experiments. Essential to the quality of the results derived with these more formal methods are mathematical models of the applications executed by the system.

- *E. Rosti is with the Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy. E-mail: rose@dsi.unimi.it.*
- *G. Serazzi is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza L. da Vinci 32, 20133 Milano, Italy. E-mail: serazzi@elet.polimi.it.*
- *E. Smirni is with the Department of Computer Science, College of William and Mary, P.O. Box 8795, Williamsburg, VA 23187-8795. E-mail: esmirni@cs.wm.edu.*
- *M. Squillante is with the IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. E-mail: mss@watson.ibm.com.*

Manuscript received 22 June 1999; revised 29 Sept. 2000; accepted 21 Feb. 2001.
Recommended for acceptance by A. Andrews.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 110107.

The modeling of applications comprising the system workload, as well as the modeling of the entire workload, is critical to effective benchmarking, procurement, tuning, and capacity planning studies, which, in turn, can play important roles in the testing and management of large parallel systems and applications. Such performance models are most effective when they provide appropriate levels of flexibility so that users can select the proper balance between accuracy and cost to best address the objectives and needs of their performance study. Further, a mathematical characterization of application behavior can help to formulate design principles and offers valuable insights to both application programmers and system designers, especially with complex systems such as the parallel environments of interest in our study. For the application programmer, a detailed application and workload characterization can be a source of suggestions for code restructuring by identifying system bottlenecks and providing ways to remove them [42]. For the system designer, the characterization is instrumental for the design and analysis of resource management policies, such as processor scheduling or file system policies in parallel systems [38], [24].

A fundamental understanding of the I/O and computation (CPU) requirements of I/O-intensive parallel applications and workloads is also needed to complement and extend previous studies on the design of high-performance parallel I/O systems. To gain a better understanding of the I/O and CPU requirements of parallel scientific applications found in practice and to identify important characteristics that need to be captured in a formal model of parallel application behavior, in this study, we first perform a detailed set of measurements of representative parallel scientific applications from the Scalable I/O Initiative [34] (SIO) suite executed on an existing large-scale parallel system. Based on an analysis of this experimental data, we then move to derive a mathematical abstraction of parallel application behavior based on a set of observations

0098-5589/02/$17.00 © 2002 IEEE

regarding the I/O requirements of the corresponding applications. This formal abstraction models the key characteristics and interactions of I/O and CPU activity within a parallel application, from which we derive various formulas that characterize application performance. This includes application scalability measures such as speedup, efficiency and a form of Amdahl's Law [2], each of which explicitly accounts for the I/O demands of a parallel application in addition to CPU demands. By exploiting an accurate and compact mathematical model that unifies previous experimental work together with the measurement data collected as part of this study, our work complements and extends existing characterization studies and provides a useful abstraction for the behavior of I/O-intensive scientific applications. While our primary focus in this paper is on scientific applications, our formal methods and results may apply more generally to commercial applications with similar characteristics.

Our model of parallel application behavior is validated against measurement data collected for a class of parallel applications from the SIO suite and is shown to be in excellent agreement. Our results further show that the model captures the fact that, depending on application CPU and I/O demands, allocating more CPU resources may increase contention on the disks and eventually increase the application execution time, thus adversely affecting application speedup. Disk contention may be caused by hardware (e.g., disk rotational speeds) and/or software bottlenecks (e.g., specific file system implementations of I/O operations [42], [44]). Given that the parallel scientific applications of interest are inherently very complex, it becomes extremely difficult to characterize in detail the exact interactions of these two causes of disk contention. Our model captures the key characteristics of this interaction in the form of a relatively simple mathematical abstraction. We also derive a generalization of standard flow-equivalent service centers [23], based on phase-type distributions and matrix-analytic methods, to capture the contention for disk resources in our parallel application behavior model when measurements of these effects are not available.

The characterization of application performance with our model can be used to explore a broad spectrum of performance issues and trade-offs in research studies having a wide range of objectives. When little or no measurement data are available, our model can be used together with rough estimates for the model parameters to investigate high-level performance issues in a manner analogous to the ways that Amdahl's law has proven effective in practice when detailed measurement data are not directly available (e.g., see [15]). As a very simple example, given rough estimates of parallel application scaling factors, users can determine a reasonable number of processors and disks on which to execute their parallel applications. System administrators can similarly use our model to gain insights into the potential cause of problems in a parallel system based on rough estimates of parallel application behavior comprising the workload and the number of processors and disks requested by users. We note that most users and administrators at high-performance supercomputing centers have sufficient information to exploit our model in these ways. Furthermore, by combining our application behavior model with other mathematical models and methods, related performance issues can be investigated more effectively; refer to [38] for a specific example.

A more detailed characterization of application performance with our model, based on measurement data or other sources of accurate application behavior, can be used to gain a better understanding of various performance issues in an existing system and/or in instances of the system that do not exist. This in turn can be used to make decisions regarding program restructuring and/or performance optimizations, e.g., grouping the I/O operations into larger or smaller chunks interleaved with more or less computation, identifying an optimal number of disks given that a specific number of processors is assigned to the application, and identifying the optimal application operational point in terms of number of processors and disks so as to maximize the application performance. We therefore consider a few specific examples of using our general model to characterize the behavior of real parallel applications. This includes the characterization of application scalability by means of functional forms based on our model and measurement data obtained as part of this study, which can be exploited for parallel application performance characterization when either measurements are not available or a synthetic application model is required. An example of the usefulness of these methods is illustrated by an analysis of the forecasted impact of hardware upgrades on application scalability as part of a capacity planning study. We further note that these specific applications of our model provide values and trends of aspects of the model components for the class of parallel applications considered in our study. In particular, the model parameter values and trends presented in this paper can be used together with our model to investigate various high-level performance issues for parallel applications with characteristics that are similar to those considered in our study when little or no measurements of these other applications are available, possibly together with other mathematical models and methods [38].

The remainder of this paper is organized as follows: Section 2 presents an overview of previous work in the characterization of I/O-intensive parallel applications and how our research complements these experimental studies. We then present our model and analysis of parallel application behavior including derivations of formulas for application scalability measures. Section 3 discusses general applications of our mathematical model to explore various performance tradeoffs in research studies having a wide range of objectives. We then consider in Sections 5 and 6 various specific examples of exploiting our model to characterize the behavior of real parallel applications in practical situations. Section 7 presents our concluding remarks.

2 Related Work

In scientific application domains where manipulating large volumes of data is a necessity, scalable parallel I/O is critical for performance. Moreover, because of the disparity in the

speeds of existing file system technologies and computation/communication hardware, there is a pressing need for a detailed understanding of the exact I/O requirements of scientific applications. During the past decade, there have been a number of studies focusing on collecting and analyzing empirical data on I/O access patterns in parallel environments. From the system perspective, detailed characterization of the I/O demands of scientific applications can lead to the development of effective file system policies [34], [17], [37], [5] or the development of frameworks that allow distinct system components (e.g., the processor scheduler and the parallel file system) to work in collaboration to achieve high performance (e.g., refer to [38]). From the application perspective, detailed knowledge of the I/O access patterns can lead to application restructuring so as to eliminate software bottlenecks [42].

The first I/O characterization efforts of scientific applications on vector supercomputers concluded that I/O behavior is regular, recurrent, and predictable [31], [32]. Similarly, Miller and Katz [25] measured a workload of mostly computational fluid dynamics applications on a Cray Y-MP and characterized the I/O behavior as highly regular, cyclical, and bursty. They first proposed a high-level I/O classification based on compulsory, checkpoint, and data staging I/O operations. Pasquale and Polyzos [31] used clustering and regression analysis to examine I/O demands and their relationship to the elapsed computation time of the Cray Y-MP workload at the San Diego Supercomputing Center (SDSC). They also isolated and analyzed two I/O-intensive applications on a Cray 90 at SDSC [32]. By focusing on both physical resource usage and functional application composition, Pasquale and Polyzos concluded that supercomputer I/O behavior was recurrent and predictable, based on the iterative nature of many scientific applications.

In contrast to the results on vector systems, recent analyses on the Intel Paragon XP/S [9], the Intel iPSC/860 [20], and the CM-5 [35], showed greater irregularity in I/O access patterns, with the majority of file requests being small but with the greatest data volume generated by a few large requests. We note that the work presented in [20], [35], [30] does not distinguish applications from one another, but rather measures all traffic to a parallel I/O system and, thus, shows how the file system is used by *all* applications.

From the application perspective, detailed characterization of the I/O demands of individual applications can lead to the identification of bottlenecks and possibly even suggest ways to circumvent them via application restructuring [1], [42]. These works underscore the interplay between application access patterns and file system features and propose a methodology for tuning the application I/O requests in order to better match the characteristics of the underlying file system and achieve high performance. A series of experimental studies [9], [43], [44] concentrated on determining the most common application I/O access patterns, focusing on their temporal and spatial characteristics, and analyzed their interactions with current parallel I/O systems. These studies have proven instrumental in guiding the development of new application programming interfaces for parallel file systems.

At the file system level, a detailed characterization of the physical I/O patterns, as seen by the device drivers and their correlation with the logical, i.e., application, I/O patterns, is presented in [40]. The conclusion of this work is that disk hardware features coupled with application I/O patterns makes identifying an ideal single file system policy or data distribution an almost impossible task. The interaction of application access patterns with the file system hardware can alternatively be measured via simulation. Indeed, disk simulators [21], [13] allow the development of detailed file system and data distribution policies and can be driven by events that capture the application behavior. However, we note that such models are by their nature prohibitively time consuming.

Our work builds on the experimental studies reported in [44] by proposing a *simple* analytic model that can nevertheless capture the application temporal I/O behavior as a function of both computational (i.e., processors) and I/O (i.e., disks) resources assigned to the application. Our model is inspired by experimental observations at the application level. This stochastic model captures contention at the disk level through a set of random variables that can be parameterized by direct measurements or by the use of other models. Our model characterizes the application scalability as a function of the allocated resources to explore a broad spectrum of performance issues and tradeoffs in research studies having a wide range of objectives. This information can, for example, provide a basis for intelligent management of system resources [38].

3 The Parallel Application Model

In this section, we formulate a general stochastic model of parallel application (and workload) behavior and we derive expressions for various measures of application performance. Our approach is motivated by a set of observations based on measurement data of the I/O requirements of a representative class of parallel scientific applications. After describing these applications, measurements, and observations, we then present our model that captures the typical characteristics and interactions of I/O and CPU activity within a parallel application. This section ends by describing a series of application performance metrics and formulas that are based on our stochastic model and are used throughout the rest of our study.

3.1 Characteristics of Parallel Program Behavior

We seek to gain a better understanding of the I/O and CPU requirements of parallel scientific applications found in practice and to identify important characteristics that need to be captured in our model. To do so, we consider scientific applications from the Scalable I/O Initiative [34] (SIO) suite. The purpose of the SIO is to collect a set of I/O-intensive applications of national importance, conduct a detailed characterization of their I/O requirements, and use this characterization to design and evaluate resource management policies for parallel file systems. Each of the applications in the SIO suite consists of one or more programs and, thus, in our study, we adopt this

SIO terminology in which a single parallel *application* is composed of a set of *programs*.

The SIO applications selected for our study are representative of the I/O behavior and requirements that have been identified as typical of many scientific applications [25], [32], [42], [44]. As an example of this application behavior, Fig. 1 illustrates the I/O and CPU activity over time for four applications: Fig. 1a: an implementation of the Schwinger Multichannel method for calculating low-energy electron-molecule collisions (ESCAT) [8]; Figs. 1b and 1c: two different implementations of electronic structure calculations using the Hartree-Fock algorithm (NWCHEM and MESSKIT) [16]; and Fig. 1d: a quantum chemical reaction dynamics code (QCRD) [53]. Fig. 1 plots the duration of each I/O operation as a function of the application execution time. All applications are executed on a 64 processor partition of a 512-node Intel Paragon XP/S, accessing data striped across 64 4GB Seagate disks. (Additional details on the system environment used for our experiments can be found in Section 5.1.1.)

The ESCAT application is a parallel implementation of the Schwinger multichannel method for calculating low-energy electron-molecule collisions. The code is written in C, FORTRAN, and assembly language. The scattering probabilities are obtained by solving linear systems whose terms must be evaluated by numerical quadrature. From an I/O perspective, there are four distinct execution phases. After initialization data are read from the disks, all nodes participate in the calculation and storage of the requisite quadrature data set that is energy dependent. Then, the quadrature data are read from the disks and energy dependent data structures are generated and combined with the reloaded quadrature data. Finally, the end results are written to disk.

The MESSKIT and the NWCHEM applications are two distinct FORTRAN implementations of the Hartree-Fock self-consistent field method that calculates the density of each electron in a molecule in the collective field of the others. From an I/O perspective, both applications have three execution phases. After an initial I/O phase where the relative geometry of the atomic centers is read from the disks, the basis sets are computed and written to the disks and atomic integrals are computed over these basis sets. Then, each processor writes the integral it computes to disk. In the final phase, all processors repeatedly read the integral data from a disk, construct the Fock matrix, and use the self-consistent field method to compute the molecular density and write final results to disk. Although both applications solve the same problem using the same logical sequence of steps, they use different integral evaluation libraries resulting in a different number of iterations and data volume, even if both applications process the same input.

The QCRD application is written in C and solves the Schrödinger equation for the cross sections of the scattering of an atom by a diatomic molecule. Parallelism is achieved by data decomposition and all processors execute the same code on different data portions of the global matrices with data elements equally distributed among processors. Due to the prohibitively large size of the global matrices to be stored in memory and the iterative nature of the numerical algorithms used by QCRD, the I/O activity on the disks is intense, with a repetitive, cyclic pattern.

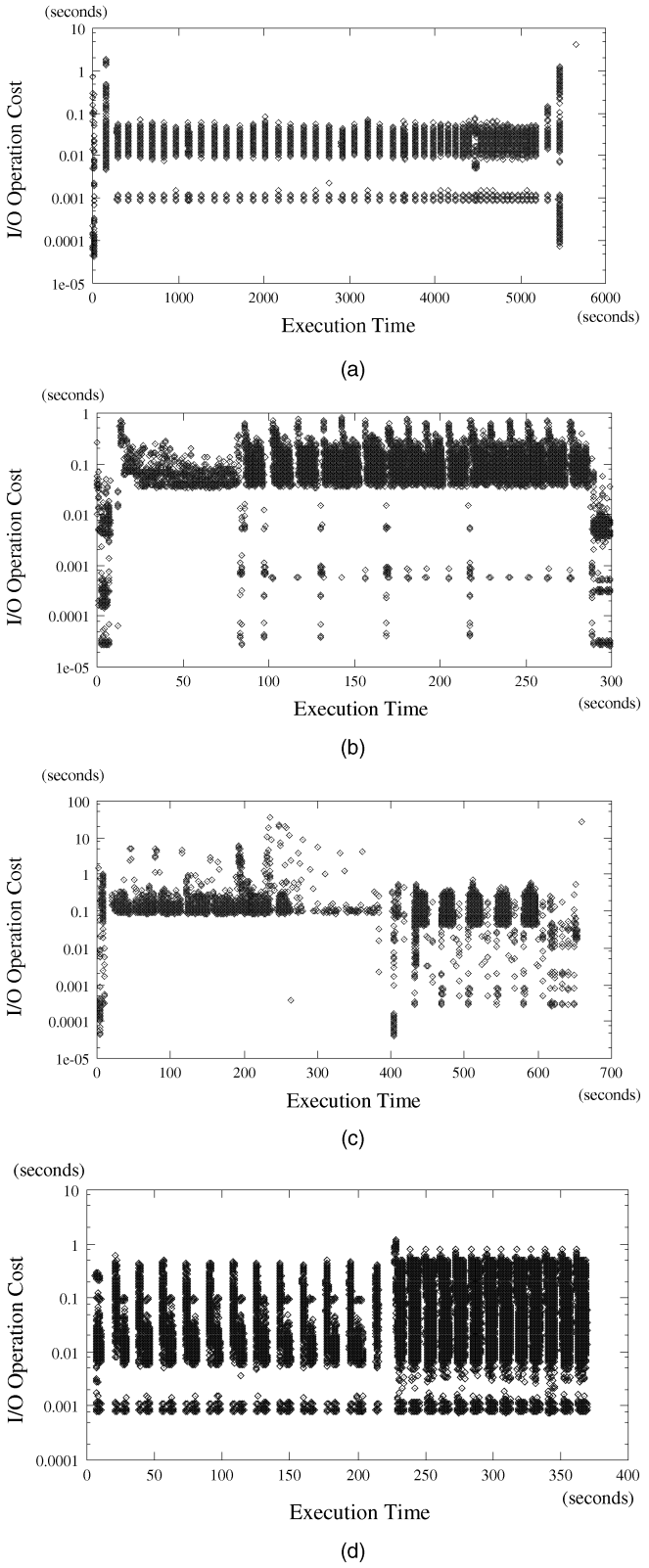

Fig. 1. Duration of I/O operations as a function of application execution time for a selection of scientific applications when executed on a 64 processor partition of a 512-node Intel Paragon XP/S accessing data striped across 64 4GB Seagate disks. (a) ESCAT, (b) NWCHEM, (c) MESSKIT, and (d) QCRD.

Note that the ESCAT and NWCHEM applications both consist of a single program, whereas MESSKIT and QCRD are comprised of multiple programs. MESSKIT consists of three programs, i.e., a short sequential program followed by two distinct parallel programs, which are easily identified in the figure by the clear shifts in behavior. The QCRD application consists of five programs, the first two of which are plotted in Fig. 1 and are easily identified by the shift in program behavior.

As the figure clearly suggests, a common characteristic of the execution behavior of many parallel programs is that each can be naturally partitioned into a series of disjoint intervals, each of which consists of a single burst of I/O activity (the densely plotted areas in the figure), possibly interleaved with a minimal amount of computation, followed by a single burst of computation (the white areas in the figure), possibly interleaved with a minimal amount of I/O. One common reason for such bursty behavior is that applications first fill a set of buffers in memory and then process the data stored in those buffers without having to access the disks [49]. The repetitive nature of these intervals is evident and it is a consequence of the iterative nature of the majority of scientific applications.

To precisely characterize this program behavior, we use the following terms throughout the remainder of the paper.

- *phase*: each disjoint interval composed of an I/O burst followed by a computation burst within a program,
- *I/O working set*: a sequence of consecutive phases that are statistically identical [27], [52].

The execution behavior of a program is therefore comprised of a sequence of I/O working sets and an application consists of a set of interdependent programs that execute in a coordinated manner. Note that each program of an application can exhibit different I/O behavior, e.g., the three programs of MESSKIT in Fig. 1, and that each phase of a program can exhibit different I/O behavior, e.g., the various phases of NWCHEM in Fig. 1, both representing changes in the application execution behavior over time.

3.2 Parallel Program Model

Based on the foregoing observations and definitions, we formulate a general model of program behavior that captures the characteristics and interactions of I/O and CPU activity in the representative class of parallel scientific applications found in the SIO suite. Our model can be hierarchically applied at the application level, the program level, and the phase level. Furthermore, by combining the characterization of individual applications with probability distribution functions that capture the details of the mixture of applications comprising the workload, a general workload model can be obtained with a similar formulation. Throughout the remainder of this paper, we use $\mathbb{Z}^+$ and $\mathbb{Z}_+$ to denote the set of positive and nonnegative integers, respectively, with $\mathbb{R}^+$ and $\mathbb{R}_+$ denoting the corresponding subsets of the real numbers.

Consider a program of a parallel application executed on one processor using one disk or on any minimum resource allocation that is required by the application. Let $N \in \mathbb{Z}^+$ denote the number of phases for the program of interest and define $T_n \in \mathbb{R}^+$ to be the duration or *length* of the nth phase, $1 \leq n \leq N$.[1] The relative length of the nth phase is then given by $L_n = T_n/\mathcal{T}$, where $\mathcal{T} \equiv \sum_{n=1}^{N} T_n$ is the total execution time of the program. Let T_n^{IO} and T_n^{CPU} be the length of the I/O and computation bursts of the nth phase, respectively, and, thus, $T_n = T_n^{\mathrm{IO}} + T_n^{\mathrm{CPU}}$, $\mathcal{T}^{\mathrm{IO}} \equiv \sum_{n=1}^{N} T_n^{\mathrm{IO}}$, $\mathcal{T}^{\mathrm{CPU}} \equiv \sum_{n=1}^{N} T_n^{\mathrm{CPU}}$, and $\mathcal{T} = \mathcal{T}^{\mathrm{IO}} + \mathcal{T}^{\mathrm{CPU}}$. We then define $F_n = T_n^{\mathrm{IO}}/T_n$ to be the fraction of the nth phase that represents the length of the phase's I/O burst; hence, $1 - F_n$ is the fraction of the phase that represents computation. Note that during program initialization and termination, either T_i^{IO} or T_i^{CPU} can be (essentially) zero, $i \in \{1, N\}$. The fraction of the program that represents I/O is given by $f^{\mathrm{IO}} = \sum_{n=1}^{N} F_n T_n/\mathcal{T}$, whereas the fraction of the program that represents computation is given by $f^{\mathrm{CPU}} = \sum_{n=1}^{N}(1 - F_n)T_n/\mathcal{T}$. For future reference, we summarize this notation in Table 1.

TABLE 1
Model Parameters for the Single-Processor, Single-Disk Program Execution Behavior

N	total number of phases
$\mathcal{T}$	total program execution time
$\mathcal{T}^{\mathrm{IO}}$	total I/O component of the execution time
$\mathcal{T}^{\mathrm{CPU}}$	total CPU component of the execution time
T_n	duration of the n-th phase
L_n	relative length of the n-th phase
T_n^{IO}	I/O component of the n-th phase
T_n^{CPU}	computation component of the n-th phase
F_n	fraction of I/O in the n-th phase
f^{IO}	total fraction of I/O in the program
f^{CPU}	total fraction of computation in the program

A simple example of these model parameter definitions is graphically illustrated in Fig. 2, where Figs. 2a and 2b show the phase behavior of a program with respect to absolute and relative execution time, respectively. We refer to the relative execution time diagram as the *phase signature* for the program. The program phase signature captures all the program behavior information, which can be described quantitatively for a program with M working sets by an M-dimensional vector $\vec{\psi} = [\psi_1, \psi_2, \cdots, \psi_M]$. The ith component of $\vec{\psi}$ is a vector $\psi_i = (\phi_i, \lambda_i, m_i)$ that characterizes the ith working set comprised of m_i phases each with I/O fraction ϕ_i and relative length λ_i. Letting $j = \sum_{k=1}^{i-1} m_k$, the ith working set consists of phases $j+1$ through $j+m_i$ and, thus, $\phi_i = F_{j+1} = \ldots = F_{j+m_i}$ and $\lambda_i = L_{j+1} = \ldots = L_{j+m_i}$. The program behavior depicted in Fig. 2 consists of five working sets, the initial single-phase working set being composed solely of a small amount of computation (i.e.,

1. All subsequent uses of the indices n, p, d, and x are intended to be under the constraints $1 \leq n \leq N$, $1 \leq p \leq P$, $1 \leq d \leq D$, and $x \in \{IO, CPU\}$, respectively, unless noted otherwise.

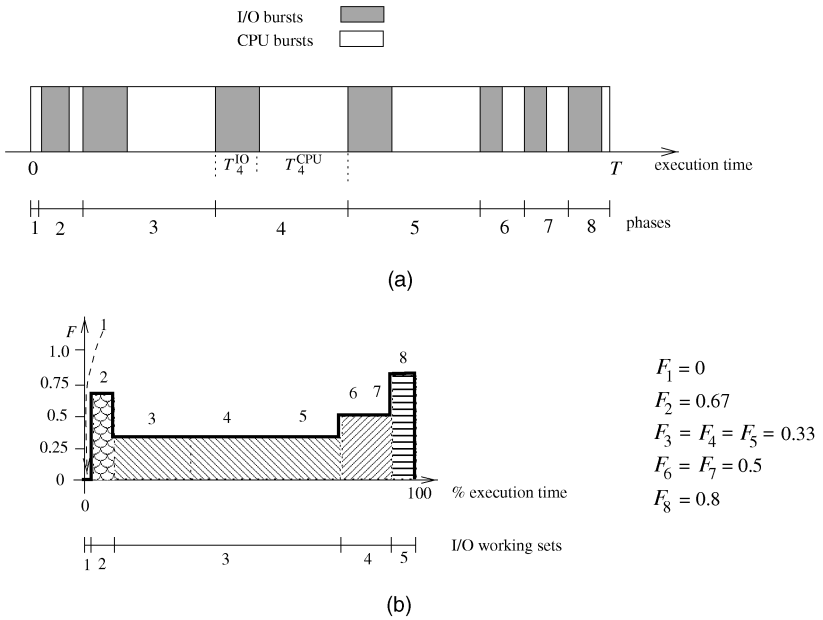

Fig. 2. Example of program behavior with $N = 8$ phases: (a) absolute execution time and (b) phase signature $\vec{\psi} = [(0, 0.013, 1), (0.67, 0.065, 1), (0.33, 0.23, 3), (0.5, 0.078, 2), (0.8, 0.076, 1)]$.

$(0, 0.013, 1)$), followed by another single-phase working set that primarily reads the problem data from disk (i.e., $(0.67, 0.065, 1)$). The program execution then has a working set consisting of three phases that are computationally intensive (i.e., $(0.33, 0.23, 3)$), followed by a working set composed of two phases with balanced I/O and CPU requirements (i.e., $(0.5, 0.078, 2)$). The program concludes with a single phase working set (i.e., $(0.8, 0.076, 1)$) during which the results are written to disk. In this example, the program has I/O-intensive single-phase working sets at the beginning and at the end of execution due to initial data acquisition and final data recording; in between these two working sets, the program cycles through a set of identical but not as highly I/O-intensive phases during which the acquired data is processed before saving it onto disks. This behavior is reflected in the phase signature where $\phi_2 > \phi_3$ and $\phi_3 < \phi_4 < \phi_5$.

We now consider the mathematical model for a parallel system consisting of P identical processors and D identical disks. When the program of interest is executed on p processors and d disks, $1 \leq p \leq P$, $1 \leq d \leq D$, the computation and I/O requirements of the program have both parallel and sequential components, as previously observed for the computational aspects of parallel programs [2]. Let $o^{\mathrm{CPU}}_{n,p,d}$ and $o^{\mathrm{IO}}_{n,p,d}$, respectively, denote the duration of the sequential components for the computation and I/O bursts of the nth phase when executed on p processors and d disks. These sequential components account for the nonparallelizable portions of the execution time, such as communication and synchronization delays at the processors and synchronization and data layout at the disks. Hence, the corresponding components that can be parallelized are given by $T^{\mathrm{CPU}}_n - o^{\mathrm{CPU}}_{n,1,1}$ and $T^{\mathrm{IO}}_n - o^{\mathrm{IO}}_{n,1,1}$. Define $s^{\mathrm{CPU}}_{n,p,d}$ and $s^{\mathrm{IO}}_{n,p,d}$ to be the respective scaling factors for the lengths of these parallel components of the computation and I/O bursts for the nth phase of the program when it is executed on p processors using d disks. These scaling factors account for the benefits and limitations of parallel execution, including forms of contention (which is considered in more detail in subsequent sections). The execution time for each component of the nth phase can then be expressed as

$$T^x_{n,p,d} = s^x_{n,p,d}\,(T^x_n - o^x_{n,1,1}) + o^x_{n,p,d}, \tag{1}$$

where $s^x_{n,1,1} \equiv 1$, $s^x_{n,p,d} \geq 0$, $x \in \{IO, CPU\}$.

Each of the above model parameters are assumed to be random variables with general probability distribution functions and finite expectations (denoted by $\mathbb{E}[\,\cdot\,]$). For a given program with a particular data set executed on a specific system configuration, these variables may be deterministic. However, when the program is executed on different data sets, the probability distributions for these random variables capture the differences in execution times. Moreover, as previously noted, by combining the characterization of the individual programs comprising an application (or the individual applications comprising a workload) via the distributions for the random variables in (1) to capture the details of the mixture of the application programs (or the mixture of the workload applications), our stochastic model can be applied at the application (or workload) level.

Since the number of phases is modeled as an integer-valued random variable (e.g., the number of phases can vary, depending on the data set in many iterative programs), let $\widehat{N}$ denote the maximum number of phases in the program of interest. More formally, we define

$$\widehat{N} \equiv \sup\{n : \mathrm{P}[N \geq n] > 0,\ n \in \mathbb{Z}^+\}.$$

The expected execution time $\mathbb{E}[\mathcal{T}_{p,d}]$ of this program when executed on p processors and d disks can then be expressed as

$$\mathbb{E}[\mathcal{T}_{p,d}] = \sum_{n=1}^{\widehat{N}} \mathbb{E}[T^{\mathrm{IO}}_{n,p,d} \cdot I_{\{N \geq n\}}] + \mathbb{E}[T^{\mathrm{CPU}}_{n,p,d} \cdot I_{\{N \geq n\}}], \quad (2)$$

where I_A denotes the indicator function for event A having the value 1 if A occurs and the value 0 if event A does not occur [27], [52].

In most parallel programs, the event $\{N \geq n\}$ is independent of $T^{\mathrm{IO}}_{n,p,d}$ and $T^{\mathrm{CPU}}_{n,p,d}$ for each n (e.g., the termination of an iterative program after the nth phase may depend on the results generated during the nth phase, but often it does not depend on the execution time of the I/O and CPU components of the nth phase). Alternatively, assuming $T^{x}_{1,p,d}, T^{x}_{2,p,d}, \ldots$ to be a sequence of independent random variables since random variable N is a stopping time for this sequence [27], [52], then $T^{x}_{n,p,d}$ and $I_{\{N \geq n\}}$ are independent for each n. Therefore, in either of these cases, we have

$$\begin{aligned}\mathbb{E}[\,\mathcal{T}_{p,d}\,] &= \sum_{n=1}^{\widehat{N}} \mathbb{E}[\,T^{\mathrm{IO}}_{n,p,d}\,]\,\mathbb{E}[\,I_{\{N\geq n\}}\,] + \mathbb{E}[\,T^{\mathrm{CPU}}_{n,p,d}\,]\,\mathbb{E}[\,I_{\{N\geq n\}}\,] \\ &= \sum_{n=1}^{\widehat{N}} \mathbb{E}[\,T^{\mathrm{IO}}_{n,p,d}\,]\,\mathrm{P}[\,N \geq n\,] + \mathbb{E}[\,T^{\mathrm{CPU}}_{n,p,d}\,]\,\mathrm{P}[\,N \geq n\,].\end{aligned} \quad (3)$$

When N is a fixed constant, this can be equivalently written as

$$\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \sum_{n=1}^{N} \mathbb{E}[\,T^{\mathrm{IO}}_{n,p,d}\,] + \mathbb{E}[\,T^{\mathrm{CPU}}_{n,p,d}\,]. \quad (4)$$

On the other hand, if the $T^{x}_{n,p,d}$ are independent and identically distributed for given values of p, d, and x (and N is a stopping time for this sequence), it follows from Wald's equation [27], [52] that (3) reduces to

$$\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \mathbb{E}[\,T^{\mathrm{IO}}_{n,p,d}\,]\,\mathbb{E}[\,N\,] + \mathbb{E}[\,T^{\mathrm{CPU}}_{n,p,d}\,]\,\mathbb{E}[\,N\,]. \quad (5)$$

Equation (3) represents a general case of our stochastic parallel program model. This expression can be simplified considerably, with the exact form depending upon the properties of N, $T^{x}_{n,p,d}$ and the elements of $T^{x}_{n,p,d}$ for the program of interest. While (3) is the starting point for the remainder of our derivations, we note that the subsequent results can be simplified according to (4) and (5) whenever the corresponding assumptions hold.

Let $Z^{x}_{n} = T^{x}_{n} - o^{x}_{n,1,1}$ and define $\overline{\mathcal{F}}_n \equiv \mathrm{P}[N \geq n]$. Upon substituting (1) into (3), we then have

$$\begin{aligned}\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \sum_{n=1}^{\widehat{N}} &\mathbb{E}[\,s^{\mathrm{IO}}_{n,p,d}\,Z^{\mathrm{IO}}_{n} + o^{\mathrm{IO}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n \\ &+ \mathbb{E}[\,s^{\mathrm{CPU}}_{n,p,d}\,Z^{\mathrm{CPU}}_{n} + o^{\mathrm{CPU}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n.\end{aligned} \quad (6)$$

If the variables $s^{x}_{n,p,d}$ are constants for the program of interest, this yields

$$\begin{aligned}\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \sum_{n=1}^{\widehat{N}} &s^{\mathrm{IO}}_{n,p,d}\,\mathbb{E}[\,Z^{\mathrm{IO}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{IO}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n \\ &+ s^{\mathrm{CPU}}_{n,p,d}\,\mathbb{E}[\,Z^{\mathrm{CPU}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{CPU}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n.\end{aligned} \quad (7)$$

TABLE 2
Model Parameters for the p-Processor, d-Disk Program Execution Behavior

$\widehat{N}$	maximum number of phases in the program of interest
$\overline{\mathcal{F}}_{(\cdot)}$	tail probability distribution for the number of phases N
$o^{\mathrm{CPU}}_{n,p,d}$	sequential component for the CPU burst of the n-th phase
$o^{\mathrm{IO}}_{n,p,d}$	sequential component for the I/O burst of the n-th phase
Z^{CPU}_{n}	parallel component for the CPU burst of the n-th phase
Z^{IO}_{n}	parallel component for the I/O burst of the n-th phase
$s^{\mathrm{CPU}}_{n,p,d}$	scaling factor for the parallel component of the CPU burst of the n-th phase
$s^{\mathrm{IO}}_{n,p,d}$	scaling factor for the parallel component of the I/O burst of the n-th phase
$T^{\mathrm{CPU}}_{n,p,d}$	execution time of the CPU component of the n-th phase
$T^{\mathrm{IO}}_{n,p,d}$	execution time of the I/O component of the n-th phase
f^{x}_{par}	cumulative parallel fraction of the x-component
f^{x}_{seq}	cumulative sequential fraction of the x-component

Note that $s^{\mathrm{CPU}}_{n,p,d} \geq 0$ *and* $s^{\mathrm{CPU}}_{n,1,1} \equiv 1$. *The same holds for* $s^{\mathrm{IO}}_{n,p,d}$.

On the other hand, when $s^{x}_{n,p,d}$ and Z^{x}_{n} are independent random variables, (6) simplifies to

$$\begin{aligned}\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \sum_{n=1}^{\widehat{N}} &\mathbb{E}[\,s^{\mathrm{IO}}_{n,p,d}\,]\,\mathbb{E}[\,Z^{\mathrm{IO}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{IO}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n \\ &+ \mathbb{E}[\,s^{\mathrm{CPU}}_{n,p,d}\,]\,\mathbb{E}[\,Z^{\mathrm{CPU}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{CPU}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n.\end{aligned} \quad (8)$$

We note that the assumptions of this expression hold for and, thus, it is used to obtain the results presented in Section 5. Otherwise, our analysis yields the following expression in terms of conditional expectations

$$\begin{aligned}\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \sum_{n=1}^{\widehat{N}} &\mathbb{E}[\,s^{\mathrm{IO}}_{n,p,d}\,\mathcal{Z}^{\mathrm{IO}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{IO}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n \\ &+ \mathbb{E}[\,s^{\mathrm{CPU}}_{n,p,d}\,\mathcal{Z}^{\mathrm{CPU}}_{n}\,]\,\overline{\mathcal{F}}_n + \mathbb{E}[\,o^{\mathrm{CPU}}_{n,p,d}\,]\,\overline{\mathcal{F}}_n,\end{aligned} \quad (9)$$

where $\mathcal{Z}^{x}_{n} \equiv \mathbb{E}[Z^{x}_{n} \mid s^{x}_{n,p,d}]$.

For future reference, we summarize in Table 2 the additional model notation for the p-processor, d-disk case.

3.3 Performance Metrics

An important and standard measure to characterize the behavior of a parallel program is *speedup*, which in the present context is defined as the ratio of the execution time on a single processor and disk to the execution time on p processors and d disks. In the stochastic setting, speedup can be expressed in a number of different ways. One particular definition of interest here is given by

$$S_{p,d} = \frac{\mathbb{E}[\,\mathcal{T}_{1,1}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]}, \quad (10)$$

which describes the *speedup surface* in the space $(S_{p,d}, p, d)$. Note that this form of the expression for speedup is consistent with that used in [28], where speedup is defined

to be the ratio of the expected response time in a queueing system with one processor to the expected response time in this system with multiple processors.

The *efficiency* of a program is typically defined as the speedup per processor allocated to the program, where the number of allocated processors in this processor-only case represents the theoretical maximum achievable speedup for the program [11] (given that superlinear speedup is not considered in the context of this paper). Hence, efficiency can be viewed to be the percentage of the theoretical maximum speedup that is actually achieved by the program with a given resource allocation. This interpretation allows us to define efficiency in the present context so as to explicitly account for both processor and disk resources. Specifically, we express program efficiency as

$$\begin{aligned} E_{p,d} &= \frac{S_{p,d}}{S^*_{p,d}} = \frac{\mathbb{E}[\,\mathcal{T}_{1,1}\,]}{S^*_{p,d}\,\mathbb{E}[\,\mathcal{T}_{p,d}\,]} \\ &= \frac{\mathbb{E}[\,\mathcal{T}_{1,1}\,]}{\mathbb{E}[\,\mathcal{T}_{1,1}\,]/\mathbb{E}[\,\mathcal{T}^*_{p,d}\,]\,\mathbb{E}[\,\mathcal{T}_{p,d}\,]} \\ &= \frac{\mathbb{E}[\,\mathcal{T}^*_{p,d}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]}, \end{aligned} \tag{11}$$

where $S^*_{p,d}$ (respectively, $\mathbb{E}[\mathcal{T}^*_{p,d}]$) is a general function that can be used to represent any desired theoretical maximum speedup (respectively, theoretical minimum expected execution time) for the program when p processors and d disks have been allocated. Note that (11), which defines the *efficiency surface* in the space $(E_{p,d}, p, d)$, guarantees that the efficiency measure is in the range $[0, 1]$ for appropriately chosen functions $S^*_{p,d}$ or $\mathbb{E}[\mathcal{T}^*_{p,d}]$ and, thus, it is consistent with the processor-only case. Furthermore, efficiency can be thought of as a measure of the effective utilization of the system.

We now consider a particular set of functions $\mathbb{E}[\mathcal{T}^*_{p,d}]$, which yields a somewhat simpler form for the efficiency expression that is exploited in Section 5. Recall from Section 3.2 that

$$\mathbb{E}[\,\mathcal{T}_{p,d}\,] = \mathbb{E}[\,\mathcal{T}^{\mathrm{IO}}_{p,d}\,] + \mathbb{E}[\,\mathcal{T}^{\mathrm{CPU}}_{p,d}\,],$$

and, therefore, we can rewrite (11) as follows:

$$E_{p,d} = \frac{\mathbb{E}[\,\mathcal{T}^*_{p,d}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]} = \frac{\mathbb{E}[\,\mathcal{T}^{*,\mathrm{IO}}_{p,d}\,] + \mathbb{E}[\,\mathcal{T}^{*,\mathrm{CPU}}_{p,d}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]}. \tag{12}$$

The total computation component of the execution time is often independent of d, as we also demonstrate in Section 5. Hence, the number of processors p is a theoretical maximum speedup for the total computation component when the program is allocated p processors and d disks, i.e., $\mathbb{E}[\mathcal{T}^{*,\mathrm{CPU}}_{p,d}] = \mathbb{E}[\mathcal{T}^{\mathrm{CPU}}_{1,1}]/p$. On the other hand, the total I/O component of the execution time depends upon both the number of processors and disks allocated to the program, as shown in Section 5. Thus, the product of the number of processors p and the number of disks d is a specific reasonable choice for the theoretical maximum speedup for the total I/O component when the program is allocated p processors and d disks, i.e., $\mathbb{E}[\mathcal{T}^{*,\mathrm{IO}}_{p,d}] = \mathbb{E}[\mathcal{T}^{\mathrm{IO}}_{1,1}]/(pd)$. Since the degree of I/O parallelism depends not only on the number of allocated disks d, a more precise definition of the theoretical maximum speedup for the total I/O component could instead be based on the effective number of I/O operations that the I/O subsystem is able to perform in parallel. This theoretical maximum speedup, however, is rarely achieved in practice. Hence, upon substituting the above functions into (12), we obtain

$$E_{p,d} = \frac{\mathbb{E}[\,\mathcal{T}^{\mathrm{IO}}_{1,1}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]\cdot pd} + \frac{\mathbb{E}[\,\mathcal{T}^{\mathrm{CPU}}_{1,1}\,]}{\mathbb{E}[\,\mathcal{T}_{p,d}\,]\cdot p}. \tag{13}$$

We further note that each of the above performance measures can be defined on a per-phase basis. Specifically, the measures $S_{n,p,d}$ and $E_{n,p,d}$ denote the speedup and efficiency of the nth phase and are defined by appropriately modified versions of (10) and (11), respectively.

3.4 Generalized Amdahl's Law

Our foregoing model and analysis can be used to derive several fundamental properties of the behavior of parallel programs. We now present one such derivation that extends Amdahl's law [2], [19] on three-dimensional space to include both processor and disk resources.

Upon substituting (6) into (10) and multiplying both the numerator and denominator by $\mathbb{E}[\mathcal{T}_{1,1}]^{-1}$, we obtain under the assumptions of (7), (8), and (9)

$$\begin{aligned} S_{p,d} = \Bigg\{ \mathbb{E}[\mathcal{T}_{1,1}]^{-1} \sum_{n=1}^{\widehat{N}} \Big(& s^{\mathrm{IO}}_{n,p,d}\,\mathbb{E}[\,Z^{\mathrm{IO}}_n\,] + \mathbb{E}[\,o^{\mathrm{IO}}_{n,p,d}\,] \\ & + s^{\mathrm{CPU}}_{n,p,d}\,\mathbb{E}[\,Z^{\mathrm{CPU}}_n\,] + \mathbb{E}[\,o^{\mathrm{CPU}}_{n,p,d}\,] \Big)\,\overline{\mathcal{F}}_n \Bigg\}^{-1}, \end{aligned} \tag{14}$$

$$\begin{aligned} S_{p,d} = \Bigg\{ \mathbb{E}[\mathcal{T}_{1,1}]^{-1} \sum_{n=1}^{\widehat{N}} \Big(& \mathbb{E}[s^{\mathrm{IO}}_{n,p,d}]\mathbb{E}[Z^{\mathrm{IO}}_n] + \mathbb{E}[o^{\mathrm{IO}}_{n,p,d}] \\ & + \mathbb{E}[s^{\mathrm{CPU}}_{n,p,d}]\mathbb{E}[Z^{\mathrm{CPU}}_n] + \mathbb{E}[o^{\mathrm{CPU}}_{n,p,d}] \Big)\overline{\mathcal{F}}_n \Bigg\}^{-1}, \end{aligned} \tag{15}$$

and

$$\begin{aligned} S_{p,d} = \Bigg\{ \mathbb{E}[\mathcal{T}_{1,1}]^{-1} \sum_{n=1}^{\widehat{N}} \Big(& \mathbb{E}[\,s^{\mathrm{IO}}_{n,p,d}\,\mathcal{Z}^{\mathrm{IO}}_n\,] + \mathbb{E}[\,o^{\mathrm{IO}}_{n,p,d}\,] \\ & + \mathbb{E}[\,s^{\mathrm{CPU}}_{n,p,d}\,\mathcal{Z}^{\mathrm{CPU}}_n\,] + \mathbb{E}[\,o^{\mathrm{CPU}}_{n,p,d}\,] \Big)\,\overline{\mathcal{F}}_n \Bigg\}^{-1}, \end{aligned} \tag{16}$$

respectively. These expressions can be viewed as general forms of Amdahl's law, with more simple forms obtained by making additional assumptions.

Define

$$f^x_{\mathrm{par}} \equiv \sum_{n=1}^{\widehat{N}} \frac{\mathbb{E}[\,Z^x_n\,]\,\overline{\mathcal{F}}_n}{\mathbb{E}[\,\mathcal{T}_{1,1}\,]} \quad \text{and} \quad f^x_{\mathrm{seq}} \equiv \sum_{n=1}^{\widehat{N}} \frac{\mathbb{E}[\,o^x_{n,p,d}\,]\,\overline{\mathcal{F}}_n}{\mathbb{E}[\,\mathcal{T}_{1,1}\,]}$$

to be the cumulative parallel and sequential fractions of the x-component in a parallel program, respectively. Under the assumption that the random variables $s^{\mathrm{IO}}_{n,p,d}$ and $s^{\mathrm{CPU}}_{n,p,d}$ are identical for all phases n, i.e., $s^x_{n,p,d} = h^{-1}_x(p,d)$, we can express (14) and (15) as

$$S_{p,d} = \left\{ f_{\text{par}}^{\text{IO}} \cdot h_{\text{IO}}^{-1}(p,d) + f_{\text{seq}}^{\text{IO}} + f_{\text{par}}^{\text{CPU}} \cdot h_{\text{CPU}}^{-1}(p,d) + f_{\text{seq}}^{\text{CPU}} \right\}^{-1} \tag{17}$$

and

$$S_{p,d} = \left\{ f_{\text{par}}^{\text{IO}} \cdot \mathbb{E}[\, h_{\text{IO}}^{-1}(p,d)\,] + f_{\text{seq}}^{\text{IO}} + f_{\text{par}}^{\text{CPU}} \cdot \mathbb{E}[\, h_{\text{CPU}}^{-1}(p,d)\,] + f_{\text{seq}}^{\text{CPU}} \right\}^{-1}, \tag{18}$$

respectively, where $h_x^{-1}(p,d)$ are general functions that can be used to scale the cumulative parallel fractions according to the two resource allocations. (A specific example of these functions is considered in Section 6.) An even simpler form can be obtained by setting $h_{\text{IO}}^{-1}(p,d) = d^{-1}$ and $h_{\text{CPU}}^{-1}(p,d) = p^{-1}$ in (17) and (18), which yields

$$S_{p,d} = \left\{ f_{\text{seq}}^{\text{IO}} + f_{\text{seq}}^{\text{CPU}} + f_{\text{par}}^{\text{IO}}/d + f_{\text{par}}^{\text{CPU}}/p \right\}^{-1}. \tag{19}$$

Note that (19) reduces to the original version of Amdahl's law upon ignoring the I/O component of the parallel program (i.e., $f_{\text{seq}}^{\text{IO}} = f_{\text{par}}^{\text{IO}} = 0$) and observing that $f_{\text{seq}}^{\text{CPU}} + f_{\text{par}}^{\text{CPU}} = 1$.

4 General Applications of the Parallel Program Model

Formal models can be used to explore a broad spectrum of performance issues and trade-offs in research studies having a wide range of objectives. Such performance models are most effective when they provide appropriate levels of flexibility so that users can select the proper balance between accuracy and cost to best address the objectives and needs of their performance study, including assisting with the collection of performance data. In this section, we briefly discuss how our stochastic model can be used to satisfy these requirements with respect to parallel application performance. The two sections that follow consider various specific examples of exploiting our parallel program model based on measurements performed as part of this study.

At one end of the spectrum, our model can be used together with rough estimates for the model parameters to investigate high-level performance issues. The use of our model in these cases is analogous to the ways that Amdahl's law has proven effective in practice when detailed measurement data are not directly available (e.g., see [15]). In many of these cases, the user may not have exact values for the scaling factors or sequential components of our model, but will instead have a reasonable range of possible values. This can then be used together with our model (and possibly other models) to investigate various performance issues of interest. At the other end of the spectrum, the user can perform detailed measurements of all instances of the parallel system to parameterize our model to obtain a complete characterization of the application (or workload) of interest. Our model can then be used to gain a better understanding of various performance issues in the current system and/or in instances of the system that do not exist, e.g., as part of a capacity planning study. Of course, a wide range of performance studies are possible in between these two extremes.

In the cases of more detailed and exact characterization, our model provides at least two benefits. First, the measurement process can be significantly simplified by allowing one to measure each unique phase (or working-set or program) in isolation and then using our model to estimate the execution behavior of the entire parallel application (or workload). In addition to reducing the amount of time required to perform measurement experiments, this approach can also lead to more accurate performance data by reducing the likelihood and impact of external events on the measured times for long-running applications. Moreover, these experiments can be repeated several times for additional accuracy, which could otherwise be prohibitively expensive for long-running applications. The key point is that our stochastic model can provide the user with greater flexibility in determining what to measure to best meet their needs and constraints. Note that it is not uncommon for measurements to be obtained at supercomputing centers, particularly for the class of applications of interest [1], [9], [20], [25], [30], [31], [32], [35], [40], [42], [43], [44] and various tools have been developed for this purpose [36]. Our model can therefore play an important role in simplifying this measurement process and enhancing these measurement-based tools.

Second, our model can be used to obtain better estimates of performance measures for instances of the parallel system where measurements are not available. For example, standard methods can be used to extrapolate the value of components of our model for larger numbers of processors and disks given the value of these components from our model parameterized by measurements for smaller numbers of processors and disks. In fact, we used extrapolation to predict the scaling factors of our model for the largest numbers of processors and disks for which we also had total execution time measurement data and then we compared the execution times from our model based on these predictions with the corresponding results obtained by applying the same extrapolation methods directly to the *total execution time* measurements for smaller numbers of processors and disks [38]. Our results demonstrate that the execution time estimates from our model based on extrapolation of the model scaling factors are essentially identical to the execution time measurement data for these cases and they are significantly more accurate than those obtained by extrapolation based solely on the total execution time measurements for smaller numbers of processors and disks. The estimates based on extrapolating (or interpolating) the scaling factors of our model can lead to considerably more accurate predictions than those obtained purely from total execution times because the model captures the computation and I/O characteristics of the parallel applications in detail and the extrapolation (or interpolation) of estimates benefit from this more detailed characterization.

Our parallel program model can also be used together with other models. As a specific example, recall from Section 3 that the scaling factors $s_{n,p,d}^x$ in our model account for various benefits and limitations of parallel execution when the program is executed on p processors and d disks. More than one processor will share a single disk whenever $p > d$ and, thus, contention among these processors to

satisfy the I/O requirements of the program at the disk can impact the execution time of the parallel program (or application or workload). The impact of such disk contention is captured in our model via the scaling factors $s^x_{n,p,d}$. When these variables are obtained from measurements, then the disk contention among multiple processors is included in these measurement data. However, if the user only obtains measurements for the cases where $p \leq d$, then another model must be used together with our program behavior model to capture the effects of disk contention in the estimates of $s^x_{n,p,d}$ for the cases $p > d$. As one example, the approach in [47] can be used to estimate these disk contention effects in applications of our model for certain system environments. In the remainder of this section, we present a mathematical analysis of the sojourn time through a disk resource shared by multiple processors in the execution of a parallel program based on a generalization of standard flow-equivalent service centers [23] to include higher moments. It is important to note that our experience demonstrates that this analysis can be used to accurately capture the shape and trends of the surfaces in Section 5 given measurements (or estimates) for the cases where $p \leq d$.

Consider the execution of a parallel program on p processors and d disks where $Y = \lceil p/d \rceil \geq 2$. The corresponding per-phase service times are modeled as independent sequences of independent and identically distributed $\mathbb{R}^+$-valued random variables following the PH-type distributions $\mathbf{B}^x_{n,p,d} = \mathrm{PH}(\underline{\beta}^x_{n,p,d}, \mathcal{S}^x_{n,p,d})$ of order $m^x_{n,p,d}$ with mean

$$(\mu^x_{n,p,d})^{-1} = -\underline{\beta}^x_{n,p,d}(\mathcal{S}^x_{n,p,d})^{-1}\mathbf{e} < \infty,$$

where e is the column vector of appropriate dimension containing all ones. Our use of PH-type distributions is of practical importance in that this class of distributions is dense within the set of probability distributions on $(0, \infty)$ (e.g., see [3]) and a number of algorithms have been developed for the fitting of PH-type distributions to empirical data (e.g., see [4], [22], [12] and the references cited therein). Furthermore, we note that the convolution of PH-type distributions is PH-type [29].

The execution of the parallel program is modeled as a closed two-center queueing network with a single delay center, representing the set of Y processors sharing one of the disks, and a single queueing center, representing the shared disk resource. The service times at the computation center follow an order m_P PH-type distribution $\mathcal{B}_P = PH(\underline{\gamma}, \mathcal{S}^P_{\mathcal{B}})$ with mean $Z_P = -\underline{\gamma}(\mathcal{S}^P_{\mathcal{B}})^{-1}\mathbf{e}$, which is constructed from the distributions $\mathbf{B}^{\mathrm{CPU}}_{\mathbf{n,p,d}}(\cdot)$. Similarly, the service times at the disk center follow an order m_D PH-type distribution $\mathcal{B}_D = PH(\underline{\delta}, \mathcal{S}^D_{\mathcal{B}})$ with mean $Z_D = -\underline{\delta}(\mathcal{S}^D_{\mathcal{B}})^{-1}\mathbf{e}$, which is constructed from the distributions $\mathbf{B}^{\mathrm{IO}}_{n,p,d}(\cdot)$.

The above formulation of the queueing network and the corresponding derivation of our approach focuses on the case where the program phases basically have the same probability distribution. This helps to elucidate the exposition of our approach and this assumption holds for the applications considered in the numerical experiments presented in [38]. Note, however, that our general approach can address the case where different phases have different probability distributions by using a multiple-class version of the queueing network representing the Y processors and the shared disk resource. Alternatively, the generator matrices $\mathcal{S}^P_{\mathcal{B}}$ and $\mathcal{S}^D_{\mathcal{B}}$ can be used to define a collection of PH-type distributions for the service times at the computation and disk centers, respectively. An analysis based on either of these approaches is viable in practice as the result of our efficient solution.

We represent the queueing network by the continuous-time stochastic process $\{X(t)\ ;\ t \in \mathbb{R}_+\}$ defined on the finite state space Ω, whose elements are given by $(i, \underline{j}^D, \underline{j}^P)$ where $\underline{j}^z \equiv (j^z_1, j^z_2, \ldots, j^z_{m_z})$, $z \in \{D, P\}$, $j^D_\ell \geq 0$, $\sum_{\ell=1}^{m_D} j^D_\ell = i$, $j^P_\ell \geq 0$, $\sum_{\ell=1}^{m_P} j^P_\ell = Y - i$. The state vector variable $i \in \{0, \ldots, Y\}$ denotes the number of parallel program threads being served at the disk center, $j^D_\ell \in \{0, \ldots, i\}$ denotes the number of parallel program threads at the disk center whose service time process is in stage ℓ, $1 \leq \ell \leq m_D$, $\sum_{\ell=1}^{m_D} j_{\ell^D} = i$, and $j^P_\ell \in \{0, \ldots, Y - i\}$ denotes the number of parallel program threads at the computation center whose service time process is in stage ℓ, $1 \leq \ell \leq m_P$, $\sum_{\ell=1}^{m_P} j_{\ell^P} = Y - i$. For convenience, we also partition the state space Ω into $Y + 1$ disjoint sets $\Omega_i \equiv \left\{(i, \underline{j}^D, \underline{j}^P) \mid (i, \underline{j}^D, \underline{j}^P) \in \Omega\right\}$, $0 \leq i \leq Y$.

Let $y_{i,w} \in \Omega_i$, $1 \leq w \leq D_i \equiv |\Omega_i|$, $0 \leq i \leq Y$, be a lexicographic ordering of the elements of Ω_i. We then define

$$\boldsymbol{\pi} \equiv (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_Y), \tag{20}$$

$$\boldsymbol{\pi}_i \equiv (\pi(y_{i,1}), \pi(y_{i,2}), \ldots, \pi(y_{i,D_i})), \tag{21}$$

$$\pi(y_{i,w}) \equiv \lim_{t \to \infty} \mathrm{P}[X(t) = y_{i,w}], \tag{22}$$

for $y_{i,w} \in \Omega_i$, $1 \leq w \leq D_i$, $0 \leq i \leq Y$. The limiting probability vector $\boldsymbol{\pi}$ is the stationary distribution for the stochastic process $\{X(t)\ ;\ t \in \mathbb{R}_+\}$. Assuming this process to be irreducible and positive recurrent, the invariant vector is uniquely determined by solving the global balance equations

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \tag{23}$$

together with the normalizing constraint

$$\boldsymbol{\pi}\mathbf{e} = 1, \tag{24}$$

where $\mathbf{Q}$ is the infinitesimal generator matrix for the process. The generator $\mathbf{Q}$, organized in the same order as the elements of the stationary vector $\boldsymbol{\pi}$, has a structure given by

$$\mathbf{Q} = \begin{bmatrix} A_{0,1} & A_{0,0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ A_{1,2} & A_{1,1} & A_{1,0} & \mathbf{0} & \cdots & & & \\ \mathbf{0} & A_{2,2} & A_{2,1} & A_{2,0} & \cdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & A_{Y-2,1} & A_{Y-2,0} & \mathbf{0} \\ & & & & & A_{Y-1,2} & A_{Y-1,1} & A_{Y-1,0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & A_{Y,2} & A_{Y,1} \end{bmatrix}, \tag{25}$$

where $A_{i,2}$, $A_{i,1}$, and $A_{i,0}$ are finite matrices of dimensions $D_i \times D_{i-1}$, $D_i \times D_i$, and $D_i \times D_{i+1}$, respectively, $0 \leq i \leq Y$.

We now derive a set of results that significantly reduces the time and space complexities of computing (23) and (24), following the approach devised independently in [14] and

[45] but applied to the complex queueing network considered herein. The stationary probability vector π satisfies the equations

$$\boldsymbol{\pi}_0 A_{0,1} + \boldsymbol{\pi}_1 A_{1,2} = \mathbf{0}, \tag{26}$$

$$\boldsymbol{\pi}_{k-1} A_{k-1,0} + \boldsymbol{\pi}_k A_{k,1} + \boldsymbol{\pi}_{k+1} A_{k+1,2} = \mathbf{0}, \quad \mathbf{1} \leq \mathbf{k} \leq \mathbf{Y-1}, \tag{27}$$

$$\boldsymbol{\pi}_{Y-1} A_{Y-1,0} + \boldsymbol{\pi}_Y A_{Y,1} = \mathbf{0}, \tag{28}$$

$$\boldsymbol{\pi}_0 \mathbf{e} + \boldsymbol{\pi}_1 \mathbf{e} + \ldots + \boldsymbol{\pi}_Y \mathbf{e} = 1. \tag{29}$$

Define

$$\begin{aligned} R_0 &\equiv A_{0,1}, \\ R_k &\equiv A_{k,1} - A_{k,2} R_{k-1}^{-1} A_{k-1,0}, \qquad 1 \leq k \leq Y-1, \\ R_Y &\equiv A_{Y,1}. \end{aligned}$$

Upon substituting these definitions into (26), (27), and (28), we obtain

$$\boldsymbol{\pi}_k = -\boldsymbol{\pi}_{k+1} A_{k+1,2} R_k^{-1}, \qquad 0 \leq k \leq Y-1, \tag{30}$$

$$\boldsymbol{\pi}_Y = -\boldsymbol{\pi}_{Y-1} A_{Y-1,0} R_Y^{-1}. \tag{31}$$

The nonsingularity of the $D_k \times D_k$ matrices R_k, $0 \leq k \leq Y$, follows from the properties of the irreducible generator matrix and the submatrices in (25). Substitution of (30) and (31) into (27) for $k = Y-1 > 0$ yields

$$\boldsymbol{\pi}_{Y-1} \left[A_{Y-1,1} - A_{Y-1,2} R_{Y-2}^{-1} A_{Y-2,0} - A_{Y-1,0} R_Y^{-1} A_{Y,2} \right] = \mathbf{0}. \tag{32}$$

This then can be used together with

$$\boldsymbol{\pi}_{Y-1} \mathbf{e} = \theta, \qquad \theta > 0, \tag{23}$$

to determine the vector $\boldsymbol{\pi}_{Y-1}$ up to a multiplicative constant when $Y > 1$. The remaining components of the vector π are obtained up to the same multiplicative constant via recurrence using (30) and (31). Given the properties of the generator matrix, the vector π is uniquely determined by the normalizing (29).

Using these results for calculating the invariant vector π significantly reduces the computational complexity over that of numerically solving the (23) and (24) directly. Specifically, this approach makes it possible to obtain the components of the stationary vector by solving $Y+1$ matrix equations of (time and space) complexity $O(D_i^2)$, $0 \leq i \leq Y$, as opposed to solving a single matrix equation of (time and space) complexity $O((\sum_{i=0}^{Y} D_i)^2)$. This makes it possible for us to compute solutions for large instances of the queueing network that are otherwise prohibitively expensive. Moreover, the algorithm (based on the above results) used to compute these solutions is numerically stable across a wide spectrum of model parameters. In fact, throughout all of the numerous experiments performed as part of our study, some of which are presented in [38], we encountered no numerical stability problems. This in turn has allowed us to exploit our matrix-analytic analysis to investigate and identify the key trends and fundamental properties of disk contention when $p > d$ across a wide range of parallel system and workload environments.

Given the invariant vector π, the kth moment of the number of parallel program threads at the two centers are given by

$$\mathbb{E}[Q_P^k] = \sum_{i=0}^{Y-1} (Y-i)^k \boldsymbol{\pi}_i \mathbf{e}, \qquad \mathbb{E}[Q_D^k] = \sum_{i=1}^{Y} i^k \boldsymbol{\pi}_i \mathbf{e}. \tag{34}$$

From Little's law [23], the throughput for the queueing network can be expressed as $\mathcal{X} = \mathbb{E}[Q_P]/Z_P$ and then we can obtain the mean sojourn time through the disk center as $\mathbb{E}[Z'_D] = \mathbb{E}[Q_D]/\mathcal{X}$. Higher moments of the sojourn time Z'_D can be obtained from (34), $\mathcal{B}_P$, π and generalizations of Little's law [7], [50], [6]. We then construct an order m'_D PH-type distribution $\mathcal{B}'_D = PH(\underline{\delta}', \mathcal{S}_{\mathcal{B}}^{D'})$ with kth moment $\mathbb{E}(Z'_D)^k = (-1)^k k! (\underline{\delta}' (\mathcal{S}_{\mathcal{B}}^{D'})^{-k} \mathbf{e})$, to match as many moments of Z'_D as desired. This PH-type distribution can then be used to characterize the per-phase sojourn time through the disk resource in our parallel program model when the number of allocated processors p is greater than the number of allocated disks d.

5 Experimental Results

We now turn to consider a specific example of exploiting our general stochastic model to characterize the behavior of real parallel applications and to validate our model against detailed measurements collected as part of this study. The characterization of application performance with our model can provide important information regarding program restructuring and/or performance optimizations. Space considerations preclude presenting an exhaustive characterization study of all applications from the SIO suite using our model and, thus, two scientific applications were selected which are representative of the class of applications considered in this paper and exhibit nonnegligible speedup.

5.1 The Experimental Platform

5.1.1 The System

Our experiments are conducted on the 512-node Intel Paragon XP/S with 64 4GB Seagate disks, each attached to a distinct computation node, at the Caltech Center for Advanced Computing Research. For the measurements obtained in this study, the subset of 64 nodes that have direct access to the 64 Seagate disks is used to execute the applications. Each application is executed in single user mode both on the processor partition and the disk partition, using the same data set for various system configurations. The number of processors and the number of disks are each varied between 1 and 64.

In every experiment, data is distributed on a set of selected disks. To minimize the problem of local versus remote disk contention, we ensure that only disks directly attached to the processor partition are in the selected set. We note, however, that there is evidence suggesting that the overhead of the parallel file system outweighs the cost of actual disk access [40]. Data is distributed on the selected set of disks according to the default Intel Parallel File System strategy using a striping unit of 64 KB, starting from a randomly chosen disk in the selected set, to keep the load balanced [18]. The performance measures are collected using Pablo, a performance analysis environment that provides trace data for all I/O and CPU requests of a parallel application [36].

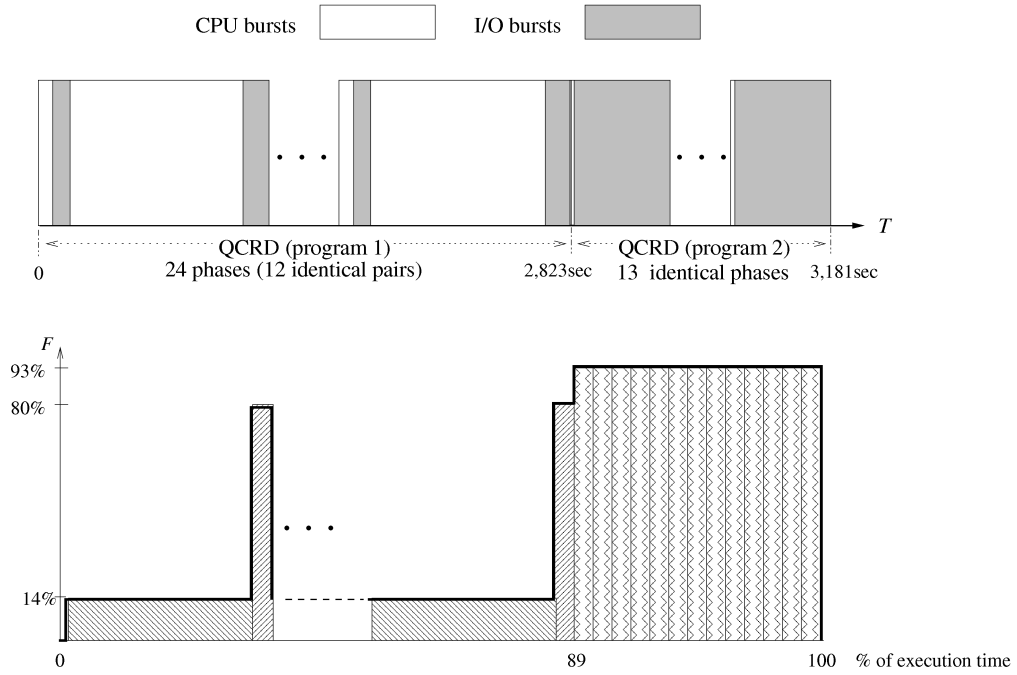

Fig. 3. Absolute execution time diagram and phase signature at application level with program identification for QCRD.

5.1.2 The Applications

We selected two chemistry applications for the purposes of this section, namely QCRD and MESSKIT.

As previously noted, QCRD is an application that solves the Schrödinger equation for the differential and integral cross section of the scattering of an atom by a diatomic molecule [53]. It was effectively used to investigate the reaction between a deuterium atom and a diatomic hydrogen molecule and led to the discovery of a geometric phase effect that had not been detected before. QCRD implements the method of symmetrical hyperspherical coordinates and local hyperspherical surface functions with a typical SPMD (Single Program Multiple Data) structure, i.e., all nodes execute the same code on different portions of the data set each of equal size so as to keep the load balanced. The complete execution of the application requires running five distinct programs in a pipelined fashion. During each program, every processor alternates computation-intensive operations, such as eigenvalue-eigenvector problem solution, integral computation, Hamiltonian matrix solution, with I/O operations to read input data and write computed results from and to the disks. At the end, all nodes read a file based on which scattering matrices are computed and written out to the disks. In this study, of the five programs comprising QCRD, we model only the two longest-executing ones which we indicate as the first and second program. As previously noted, MESSKIT is a FORTRAN application that performs electronic structure calculations using the Hartree-Fock algorithm [34]. The electronic density around a molecule is computed by considering each of the molecule electrons in the collective field of the others, iterating the computation until the field felt by each electron is consistent with that of the other electrons. Input to the algorithm is the basis functions derived from the atoms and the relative geometries of the atom centers. The Coulomb interactions between electrons are computed by solving the atomic integrals over the basis functions, thus producing an approximate molecular density. The density and the atomic integrals are then used to derive a Fock matrix. A self-consistent field (SCF) method is finally applied until the molecular density converges to within an acceptable threshold. The application is comprised of three logical programs executed in a pipelined fashion. During the first program, a single node (node 0) is active, reading initialization data from disk, calculating the basis sets, and writing the results out to disk. Since this program is not parallel, it is not modeled here. We model the other two programs, i.e., ARGOS and SCF. Both programs exhibit different I/O and CPU characteristics.

5.2 Phase Signatures

To characterize the behavior of the two applications, we start by examining their phase signatures. Due to memory requirements, the base-case execution of QCRD is for a configuration of two processors and one disk. Fig. 3 depicts the absolute execution time diagram and the phase signature for each of the two representative programs of QCRD.

In the first program, the application is characterized by a sequence of alternating CPU- and I/O-intensive phases that repeat 12 times, thus leading to the phase signature $\vec{\psi}_1 = [\psi_{1,1}, \cdots, \psi_{1,24}]$, where $\psi_{1,i} = (0.14, 0.066, 1)$ for $i \in \{1, 3, \ldots, 23\}$ and $\psi_{1,i} = (0.97, 0.0082, 1)$ for $i \in \{2, 4, \ldots, 24\}$. During the second program, the I/O activity is more intense and the signature is characterized by a single working set comprised of 13 identical phases, which yields the phase signature vector $\vec{\psi}_2 = [(0.92, 0.03, 13)]$. The overall application signature is therefore obtained by concatenating the two vectors, i.e., $\vec{\psi} = [\vec{\psi}_1, \vec{\psi}_2]$.

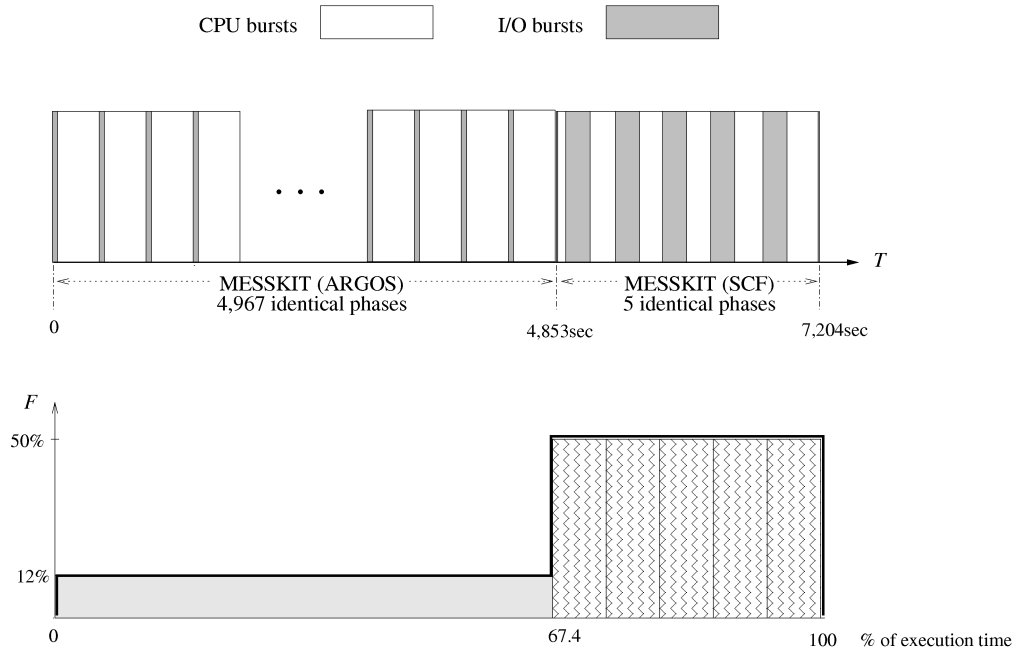

Fig. 4. Absolute execution time diagram and phase signature at application level with program identification for MESSKIT.

The phase signatures for the two significant programs of MESSKIT are reported in Fig. 4. The behavior of the two programs is clearly different. The first program, ARGOS, is CPU bound, while the second, SCF, is balanced between CPU and I/O. The application is comprised of two multiphase working sets given by the phase signature vector $\vec{\psi} = [\psi_1, \psi_2]$, where $\psi_1 = (0.12, 0.00013, 4967)$ and $\psi_2 = (0.5, 0.065, 5)$. The first one consists of a large number of phases where single complex operations, such as matrix multiplication, alternate with nonnegligible I/O operations. During SCF, both computation and I/O are concentrated in fewer longer periods where only either activity is performed.

Since both applications are written using the SPMD programming paradigm, all nodes execute essentially the same code on different data. However, one particular node plays the role of the execution coordinator, sending and receiving data to and from the rest of the nodes, possibly performing some additional computation in order to combine the results of all nodes, even reading and/or writing data to the file system. Given that this extra work performed by the coordinator node is often not parallelized, it is identified as sequential. Since Pablo allows us to detect the exact number and cost of I/O operations performed by each node, as well as the total time spent in pure computation along with the number of CPU operations, we can easily determine the sequential components in the cumulative fractions of I/O and CPU for each application.

For QCRD and MESSKIT the amount of sequential components identified was negligible, so we consider that the total fractions of I/O and CPU identified in both applications are fully parallelizeable. The application level and program level values of f_{par}^{IO} and f_{par}^{CPU} for both QCRD and MESSKIT are reported in Table 3. We note that, for both applications, the total percentage of the aggregate time spent on I/O, given the smallest configuration of I/O and CPU resources, is about 25 percent. This percentage changes across programs, i.e., we observe that the second program of each application is more I/O intensive than the first. As the I/O and CPU resources assigned to the application increase, the total percentage of time spent on I/O versus CPU will change according to the scalability of the CPU and I/O components.

5.3 Speedup

We now present the speedup surfaces obtained from measurement data at the application and program levels. Fig. 5 reports the speedup surfaces for both applications and their respective programs as a function of p and d.

As these results show, speedups at the application level are smoother than those at the program level because measurement instabilities tend to compensate when data from a set of programs is combined. While our experiments were run in single-user mode, there were some unavoidable interferences because it was still possible for other users to copy data to/from the disks, even though no other user was allowed to run on the processors. Additionally, the operating system would perform its usual periodic bookkeeping. Note that our trace data indicate the exact times where such interferences occur. These interferences, although sporadic, do indeed affect the smoothness of the speedup surface. As previously discussed, the use of our model in the measurement process can help to address these issues.

A characteristic common to all the surfaces plotted in Fig. 5 is the poorer scalability exhibited for fixed numbers of processors when varying the number of disks, than for fixed numbers of disks when varying the number of processors. Such behavior suggests that simply increasing the amount of I/O resources allocated to an application does not necessarily guarantee a corresponding performance improvement for I/O-bound applications. Specifically, being

TABLE 3
Fractions of Computation and I/O for the Applications of Interest at Application and Program Levels that are Fully Parallelizeable

	QCRD			MESSKIT		
	application	1^{st} program	2^{nd} program	application	1^{st} program	2^{nd} program
f_{par}^{CPU}	0.738	0.822	0.073	0.755	0.879	0.499
f_{par}^{IO}	0.262	0.178	0.927	0.245	0.121	0.501

Note that there are negligible sequential components for both I/O and CPU, both at the application and program levels. Also, the cumulative I/O fractions in the table correspond to the total I/O area in each of the phase signatures in Figs. 3 and 4.

more I/O intensive as shown in Table 3, QCRD achieves lower global speedup than MESSKIT. In fact, the I/O-intensive programs of both applications (the second program in both cases) achieve very low speedups, in the range of 12-13 or less. The first program in both cases is CPU intensive. The larger fraction of computation for the case of MESSKIT is reflected in a better speedup.

For the specific implementation of QCRD, the limited scalability observed is primarily caused by a poor matching between the I/O request size and the disk stripe size, i.e., 2,400 bytes and 64KB, respectively [44]. Since file accesses both for reads and writes are strided, processes compete to access the same disk, thus increasing contention without exploiting the file system parallelism as reflected in the scaling factors obtained from our measurements. Although we could have improved QCRD's performance by rewriting the application so as to match the I/O request size with the disk stripe size, we decided against this option first because application rewriting was beyond the scope of this study and, second, because our purpose was to characterize the exact applications of the SIO suite. Our analysis, however, can be used to improve this aspect of the SIO suite.

On the other hand, the implementation of MESSKIT exploited the file system parallelism in a more effective way, due to a better match of request size and striping unit size. The second program of QCRD is so I/O-intensive that speedup decreases after leveling off at medium size configurations. Such behavior is due to a considerable increase of contention and synchronization delays, which is reflected in the scaling factors that are greater than 1 (see Table 4), i.e., the ratio between the execution time of the parallel component with p processors and d disks to its counterpart with one processor and one disk (see the definition in Section 3) is greater than 1.

5.4 Efficiency

We now consider the corresponding results from our model for application efficiency. As defined in (11), efficiency depends upon both p and d. Fig. 6 illustrates the efficiency surfaces for both MESSKIT and QCRD at the application and program levels. Note that efficiency is less than or

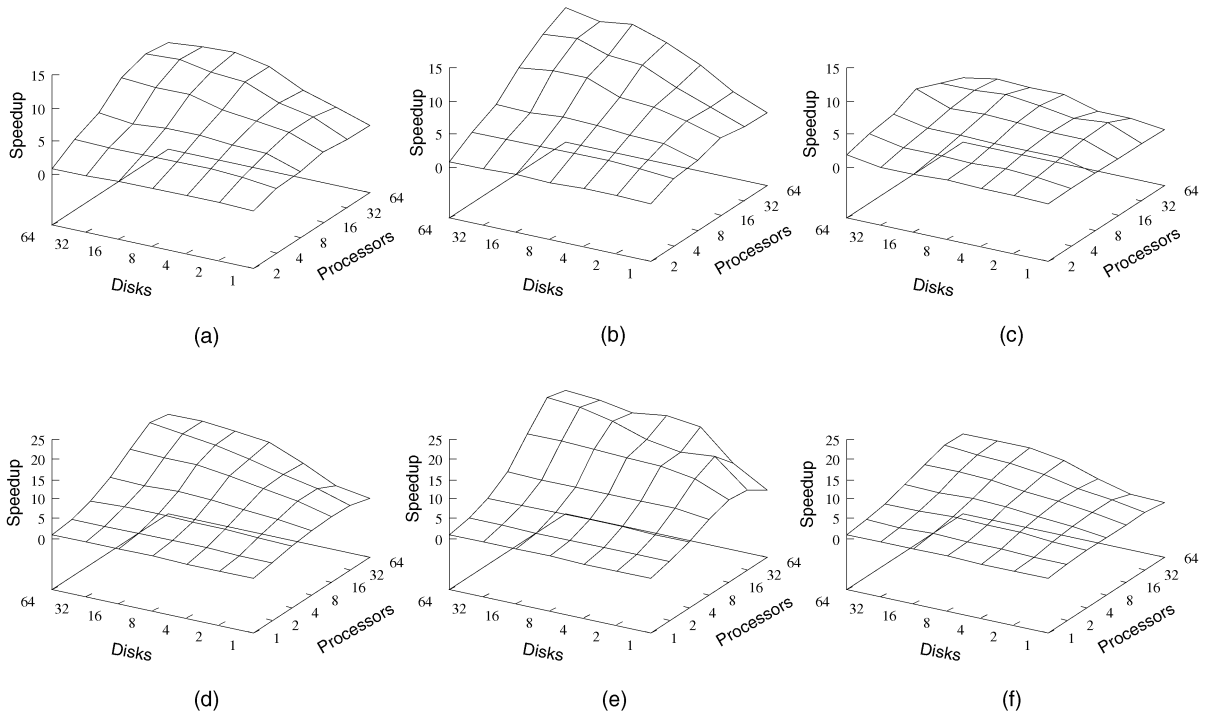

Fig. 5. Measured speedup surfaces as a function of p, d in log scale at application level (leftmost column) and program level (center and rightmost columns) for QCRD ((a)-(c)) and MESSKIT ((d)-(f)). Note that, to clarify the presentation, we use a different scale on the z-axis for each application and its programs.

TABLE 4
Computation Scaling Factors $s_{p,d}^{\mathrm{CPU}}$ at Application and Program Level for the Applications of Interest

	QCRD			MESSKIT		
p	application	1^{st} program	2^{nd} program	application	1^{st} program	2^{nd} program
1	Not Available	Not Available	Not Available	1.002(0.003)	1.002(0.004)	1.001(0.003)
2	1.094(0.127)	1.100(0.130)	1.002(0.155)	0.527(0.002)	0.511(0.002)	0.586(0.006)
4	0.338(0.002)	0.331(0.001)	2.216(0.445)	0.285(0.002)	0.258(0.0004)	0.382(0.008)
8	0.197(0.010)	0.187(0.009)	2.506(0.555)	0.163(0.0008)	0.133(0.001)	0.273(0.002)
16	0.124(0.001)	0.113(0.001)	2.443(0.409)	0.104(0.001)	0.071(0.001)	0.223(0.004)
32	0.098(0.001)	0.084(0.001)	2.835(0.291)	0.087(0.008)	0.055(0.011)	0.203(0.007)
64	0.097(0.007)	0.078(0.003)	3.978(0.902)	0.082(0.010)	0.049(0.014)	0.202(0.012)

Values are averaged across d and standard deviation is reported in parenthesis.

equal to 1. The two example applications reach an efficiency equal to 1 only for the minimum (p, d) configuration.

As in the case of speedup, we note that the overall efficiency of the QCRD application is dominated by that of its first program because the first program is significantly longer than the second one. We note that the first QCRD program is CPU bound (see Table 4) but still does not scale very well when more than 16 processors are assigned to the application (see its CPU scaling factors in Table 5). Additionally, as we will also see in the next section, contention for the disks adversely affects scalability when the number of processors exceeds the number of disks (see the first QCRD program's scaling factors in Table 5). The second QCRD program is heavily I/O bound and, because of heavy contention on the disks (as shown by its I/O scaling factors reported in Table 5), it exhibits even lower speedup than the first QCRD program and its efficiency is very low.

MESSKIT's overall efficiency is dominated by the efficiency of its first program because ARGOS is almost twice as long as SCF. We note that ARGOS is heavily CPU bound with good CPU scaling factors (see Table 4) and this is reflected in the high efficiency levels up to 16 processors. The I/O requirements of ARGOS scale moderately and require a minimum of eight disks to maintain low disk contention. The SCF efficiency surface is flatter than that of ARGOS and this is attributed to the fact that SCF balances CPU and I/O. However, SCF achieves half the scalability of ARGOS and this is also reflected in its CPU and I/O scaling factors.

5.5 Model Validation

In this section, we validate one of the closed-form expressions from our stochastic model, i.e., (18) which is based on (8) and (10), using the corresponding measurement data for both applications. The critical parameters to be estimated in (18) are the computation and I/O scaling factors. Table 4 reports the computation scaling factors, averaged across all disks together with their standard deviation, at both the application and program levels. We

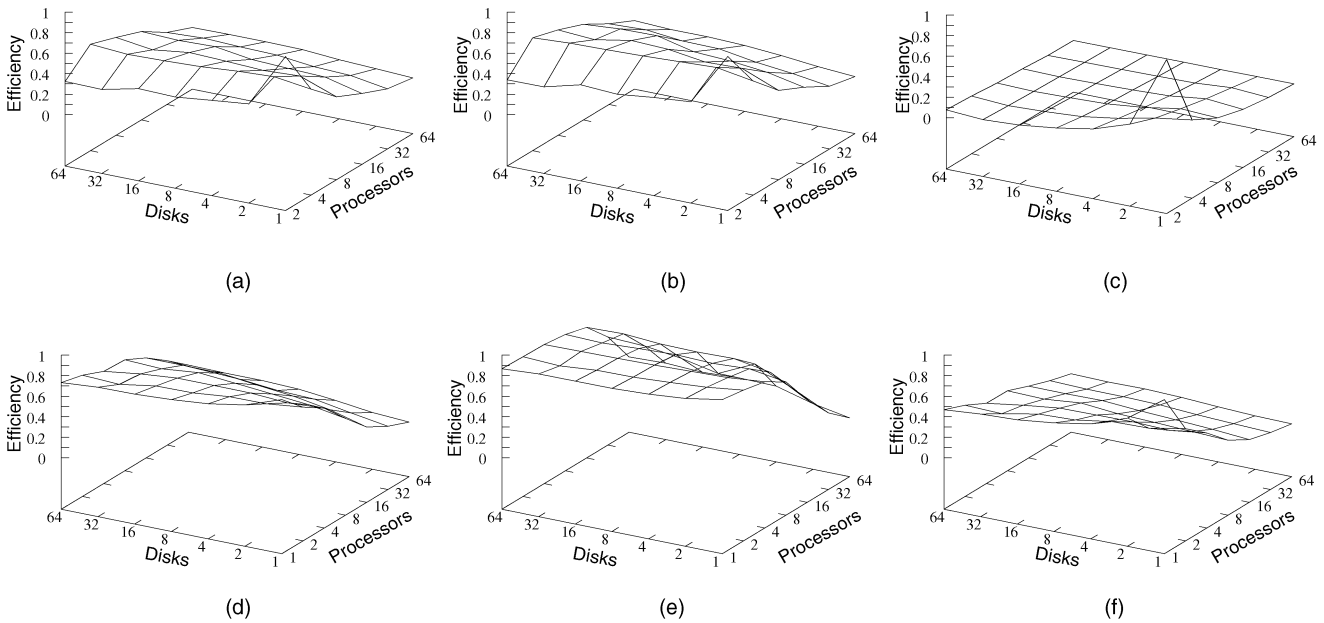

Fig. 6. Measured efficiency surfaces as a function of p, d in log scale at application level (leftmost column) and stage level (center and rightmost columns) for QCRD ((a)-(c)) and MESSKIT ((d)-(f)).

TABLE 5
I/O Scaling Factors $s_{p,d}^{\mathrm{IO}}$ for QCRD and MESSKIT at the Application Level

	QCRD, application						
p,d	1	2	4	8	16	32	64
2	1.000	0.773	0.747	0.788	0.755	0.827	0.722
4	0.857	0.468	0.415	0.416	0.422	0.437	0.369
8	1.027	0.393	0.350	0.287	0.246	0.257	0.243
16	0.912	0.389	0.312	0.250	0.150	0.158	0.149
32	1.101	0.461	0.326	0.196	0.164	0.130	0.143
64	1.123	0.591	0.389	0.206	0.156	0.151	0.178
	MESSKIT, application						
p,d	1	2	4	8	16	32	64
1	1.000	0.995	0.988	1.003	1.024	1.014	1.119
2	0.926	0.732	0.576	0.586	0.563	0.702	0.643
4	0.655	0.490	0.370	0.315	0.296	0.342	0.422
8	0.651	0.386	0.269	0.205	0.154	0.180	0.161
16	0.777	0.360	0.206	0.143	0.097	0.093	0.085
32	0.880	0.403	0.200	0.105	0.073	0.067	0.045
64	1.828	0.858	0.294	0.094	0.051	0.050	0.029

note that, for the case of both applications, the low standard deviations in the average values of $s_{p,d}^{\mathrm{CPU}}$ across all disks illustrate that the number of assigned disks affect only marginally the scalability of the computational component of the application. All the scaling factors are less than 1, indicating good scalability, with 1 referring to the base case. Notice that the scaling factors for the second program of QCRD are greater than 1, confirming the poor scalability shown by the decreasing speedup in Fig. 5. In our sample of applications, CPU-intensive programs have smaller scaling factors, thus showing better scalability, than I/O-intensive programs.

The I/O scaling factors at the application level for both applications are reported in Table 5. The program level scaling factors are shown in Table 6. Unlike the computation factors, the I/O scaling factors are a function of both processors and disks. Therefore, we opt to illustrate all of the data, i.e., not just averages and standard deviations as we did in Table 4. Since the applications we focus on are structured in the typical SPMD fashion, the load on the parallel file system depends on the number of processing elements that simultaneously make requests to the disks. Consequently, the I/O scaling factor should reflect a dependency on *both* processors and disks.

When the scaling factors are known a priori, e.g., from measurements, they can be substituted in (18) and the application speedup is computed. The results of the comparisons of the measured speedup surfaces with those computed using (18), upon substituting the scaling factors of Tables 4, 5, and 6, are summarized in Table 7, where the relative error is averaged across all combinations of processors and disks. As these results show, the computed values are in very good agreement with the measured values.

TABLE 6
I/O Scaling Factors $s_{p,d}^{\mathrm{IO}}$ for QCRD and MESSKIT at the Program Level

	QCRD, 1^{st} program						
p,d	1	2	4	8	16	32	64
2	1.000	0.677	0.661	0.767	0.644	0.746	0.818
4	0.781	0.383	0.333	0.326	0.330	0.340	0.396
8	1.073	0.403	0.328	0.226	0.183	0.190	0.252
16	0.894	0.475	0.311	0.252	0.109	0.109	0.149
32	1.202	0.615	0.350	0.187	0.163	0.095	0.116
64	1.216	0.632	0.317	0.177	0.106	0.105	0.086
	QCRD, 2^{nd} program						
p,d	1	2	4	8	16	32	64
2	1.000	0.917	0.877	0.819	0.923	0.950	0.578
4	0.971	0.595	0.540	0.552	0.562	0.584	0.328
8	0.956	0.377	0.383	0.379	0.341	0.359	0.229
16	0.941	0.261	0.312	0.246	0.213	0.233	0.150
32	0.948	0.228	0.290	0.211	0.167	0.184	0.184
64	0.982	0.530	0.498	0.250	0.231	0.221	0.318
	MESSKIT, 1^{st} program						
p,d	1	2	4	8	16	32	64
1	1.000	0.985	0.985	0.993	0.964	0.949	1.053
2	0.570	0.509	0.483	0.493	0.491	0.589	0.502
4	0.359	0.288	0.256	0.249	0.247	0.246	0.256
8	0.332	0.179	0.150	0.128	0.127	0.135	0.130
16	0.533	0.188	0.095	0.078	0.068	0.076	0.068
32	0.764	0.280	0.093	0.052	0.039	0.052	0.038
64	1.580	0.652	0.168	0.031	0.023	0.036	0.023
	MESSKIT, 2^{nd} program						
p,d	1	2	4	8	16	32	64
1	1.000	1.000	0.989	1.008	1.053	1.046	1.152
2	1.103	0.843	0.622	0.632	0.599	0.758	0.713
4	0.802	0.590	0.426	0.348	0.321	0.390	0.505
8	0.809	0.489	0.328	0.244	0.167	0.203	0.176
16	0.897	0.446	0.261	0.176	0.111	0.101	0.093
32	0.938	0.465	0.253	0.132	0.090	0.075	0.049
64	1.951	0.961	0.357	0.125	0.064	0.056	0.032

It is important to point out, however, that the differences between the measurement data and the model estimates are primarily due to system instabilities. Our trace data indicate that during some runs, in particular for those runs that contribute to the "jagged" speedup points of Fig. 5, there was some I/O activity on the disks caused by a source external to the application. Recall that, while all experiments were conducted in single-user mode with respect to job execution, routine operating system activity was unavoidable. Additionally, although other users could not execute on the machine, they were able to copy files to/from the parallel file system via the operating system. In particular, the trace data indicate that, in some experiments, within a program, the I/O times of only very few of the phases were much longer in comparison to the remaining

TABLE 7
Average Relative Error Over All the Available System Configurations for the Measured and Computed Speedup ($p, d \in \{1, 64\}$) at Application and Program Levels for the Applications of Interest

	QCRD			MESSKIT		
	application	1^{st} program	2^{nd} program	application	1^{st} program	2^{nd} program
avg. rel. error	2.14%	2.65%	2.92%	1.79%	4.58%	1.05%

phases. Since this effect vanishes from run to run of the exact same experiment, we conclude that this is a side-effect of other read/write activity to the file system. Specifically, our trace data indicate that the higher errors observed for MESSKIT's first program are due to a few outliers corresponding to a not fully isolated file system as the partially jagged speedup surface also shows (see Fig. 5e). These outliers can have a profound effect on obtaining the exact scaling functions of the application. In fact, if scaling factors were computed for each phase instead of for each program, the results from our model would be in perfect agreement with the measurements. On the other hand, this may not be practical, as a program may consist of a large number of phases, e.g., ARGOS consists of 4,967 phases. As discussed in Section 4, we again observe that our model makes it possible for users to choose the proper balance between accuracy and cost to best address the objectives and needs of their performance study.

A graphical illustration for one case of model validation is reported in Fig. 7, where the measured (solid line) and computed (dashed line) speedup surfaces for QCRD at the application level are plotted. As the figure shows, the measured speedup and the speedup obtained using (18) are in excellent agreement. The scaling factors reported in Tables 4, 5, and 6 and used in (18) are the key for accurate speedup estimates. Our derivation in Section 4 of a general analysis to capture the contention for disk resources when multiple processors share a single disk can be used to accurately estimate these scaling factors when measurements are not available, as we have demonstrated by comparing this approach with measurement data [38].

Finally, we note that the analysis in this section, based on our parallel program model, provides values and trends of

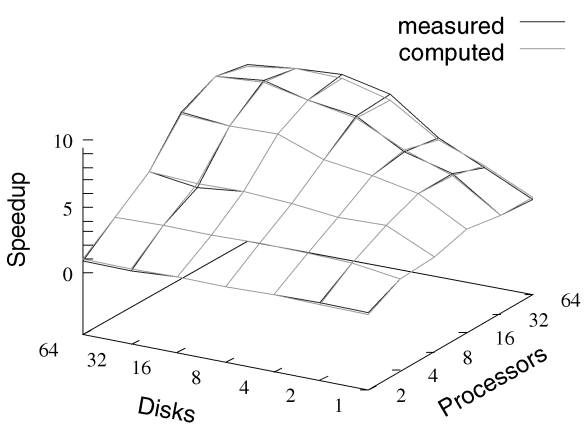

Fig. 7. An example of validation of the computed speedup surfaces using Amdahl's law given by (18) at application level for QCRD against the measured speedup surface as a function of p, d in log scale.

aspects of the model components for the class of parallel applications considered in our study. In particular, the model parameter values and trends provided in Tables 3, 4, 5, and 6 can be used together with our model to investigate various high-level performance issues for parallel applications with characteristics similar to those of QCRD and MESSKIT when little or no measurements of these other applications are available, possibly together with other mathematical models and methods [38].

6 Case Study

In this section, we illustrate some additional possible uses of our stochastic model of parallel program behavior. We focus on the characterization of application scalability by means of functional forms and on the analysis of the impact of hardware upgrades on application scalability as part of a capacity planning study.

6.1 Functional Models of Application Scalability

As demonstrated in the previous section, our model can be used to effectively characterize parallel application execution behavior. While a complete and exact characterization of parallel applications is possible only when the applications are available and can be executed on an actual system, a functional form that captures the application scalability can contribute to the identification of the typical behavior of parallel applications in a 3-dimensional space, providing information on the program behavior across both processor and disk resources. When no measurement data are available, the behavior of parallel applications can be characterized by means of appropriate functional forms. Furthermore, if only a partial set of measurements is available for a limited number of system configurations, estimates of missing points can be derived using such functional forms (possibly together with other models, e.g., see Section 4).

In a manner similar to what has been done in the processor-only case [2], [10], [39], we consider a functional form for the application speedup surfaces in the (p, d) space. By an appropriate parameterization of such a form, families of speedup surfaces can be derived that capture the likely behaviors of parallel applications. The number and nature of the parameters (i.e., the physical meaning, if it exists, such as average parallelism, parallelism variability, degree of uneven spread of the work among the processors, or overhead due to parallel processing) that appear in the functional form characterize the variety of possible behaviors the functional form may describe. Specific values of the parameters characterize individual applications. For example, in the processor-only case, the single-parameter

Amdahl's law [2] yields a single family of speedup curves, while Dowdy's two-parameter [10] and Sevcik's four-parameter [39] functions allow one to describe a wider range of speedup behaviors.

One of the functional forms we consider in this paper, i.e., (18), is characterized by the scaling functions $h^x_{p,d}$ and the parallel and sequential fractions of IO and CPU, resulting in a total of six parameters (which we further simplify below). We note that versions of Amdahl's one-parameter (see (19), Dowdy's two-parameter, and Sevcik's four-parameter functions can be used together with our model. Hence, in order to consider the more detailed six-parameter function, we provide a functional form for the scaling functions $h^x_{p,d}$ so that (18) can be used in the absence of measured data to derive suitable speedup surfaces for given sets of f^x_{par} values. Recall that all of these parameters were derived from measurements in the previous section, in order to validate the functional form of equation (18).

Simple functions such as $h^{\mathrm{CPU}}_{p,d} = p$, although appealing because they lead to a form of (18) similar to the original version of Amdahl's law (see (19)), fail to capture the actual scalability of some applications. As Table 4 shows, such linear scalability is not suitable to characterize the application behavior. Likewise, the corresponding function for the I/O component, i.e., $h^{\mathrm{IO}}_{p,d} = d$, fails to capture the dependence upon both p and d that the I/O scaling factors exhibit (see Tables 5 and 6). Since the key to effective speedup modeling is a functional form that is able to capture and describe the actual scalability of the application, we propose the use of logarithmic functions. Note that logarithmic scalability has been observed in the past for the processor-only case [26], as it models the diminishing return when the amount of processors assigned to the application increases.

Building on these considerations and on the analysis of empirical scaling factors such as those reported in Tables 4, 5, and 6, we consider the following scaling functions:

$$h^{\mathrm{CPU}}_{p,d} = \alpha_1 \log p, \qquad h^{\mathrm{IO}}_{p,d} = \alpha_2 \log d \log p, \tag{35}$$

where α_1 and α_2 are coefficients in $\mathbb{R}^+$ that allow one to tune the model to the specific application of interest. Note that the degree to which I/O resources can be effectively used, i.e., I/O scalability, depends upon the number of processors assigned to the application. By substituting (35) into (18) and letting $f_{seq} = f^{\mathrm{CPU}}_{seq} + f^{\mathrm{IO}}_{seq}$, we obtain

$$S_{p,d} = \left\{ \frac{f^{\mathrm{CPU}}_{par}}{\alpha_1 \log p} + \frac{f^{\mathrm{IO}}_{par}}{\alpha_2 \log d \log p} + f_{seq} \right\}^{-1}. \tag{36}$$

Note that, as p increases to infinity, this expression asymptotically approaches $1/f_{seq}$, just like Amdahl's law. On the other hand, as the number of disks increases to infinity, $S_{p,d}$ tends toward $\alpha_1 \log p/(f^{\mathrm{CPU}}_{par} + f_{seq}\alpha_1 \log p)$.

As an example of the effectiveness of this functional form for speedup in characterizing the behavior of real applications, we plot in Fig. 8 the measured speedup surface (solid line) for QCRD at the application level (from Section 5) and the corresponding speedup surface obtained from (36) (dashed line) using $\alpha_1 = 2.5$, $\alpha_2 = 0.5$, $f^{\mathrm{IO}}_{par} = 0.262$, and $f^{\mathrm{CPU}}_{par} = 0.738$. The values of α_1 and α_2 are obtained by applying a best-fit method to (36) and the scaling factors are

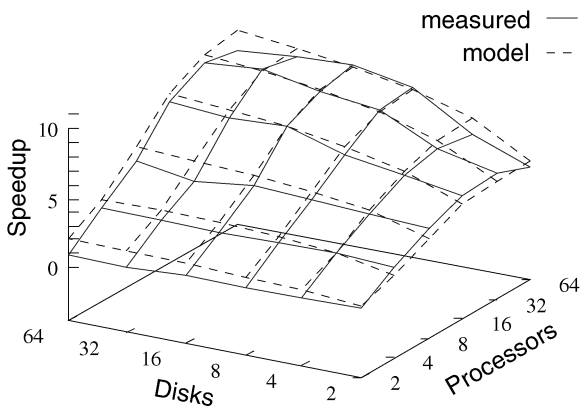

Fig. 8. An example of use of the characterization model with the logarithmic functional forms for the scaling factors (dashed line) compared with a measured speedup surface (solid line) for QCRD program 1.

from the available set of values reported in Tables 4 and 5. The average relative error between the measured speedup values and those computed using (36) is in the range of 12 percent, which is a reasonably good approximation in workload forecasting studies. Note that, if only a subset of the measured speedup values of Fig. 8 were available, (36) could be parameterized so as to match these speedup values and then used (possibly together with other models) to derive estimates for the complete grid. The quality of such estimates depends upon the functional form used, in this case (36), the available measurement points used for the parameterization, and the relative position of the required estimates with respect to the available values. However, the value of functional forms such as (18) and (36) is that they represent smooth approximations of actual applications behavior that can be used in the design and analysis of system algorithms, as well as system tuning and procurement studies. By varying the functional form parameters, an entire range of possible behaviors can be explored and analyzed. Finally, we again note that the results presented in Section 5 can be used to examine high-level performance issues for applications with characteristics that are similar to those considered in this paper.

6.2 Application Scalability and Capacity Planning

We now illustrate how the above functional form for application speedup can be exploited to forecast the impact of system changes on application scalability, such as upgrades of hardware components aimed at improving application-level and/or system-level performance in the most cost effective manner. While throughput may be a more important performance metric for system administrators, application execution time can be more important to the users in high-performance supercomputing centers who often need to factor the expected execution time into their decision of which problem instance to submit for execution (especially when the scheduling policy makes it possible for users to estimate the time between job submission and the start of job execution [41]). Since the user might also want to consider how effectively the allocated resources are used (especially if the user is charged based on both the number of resources allocated and the execution time) and, in order

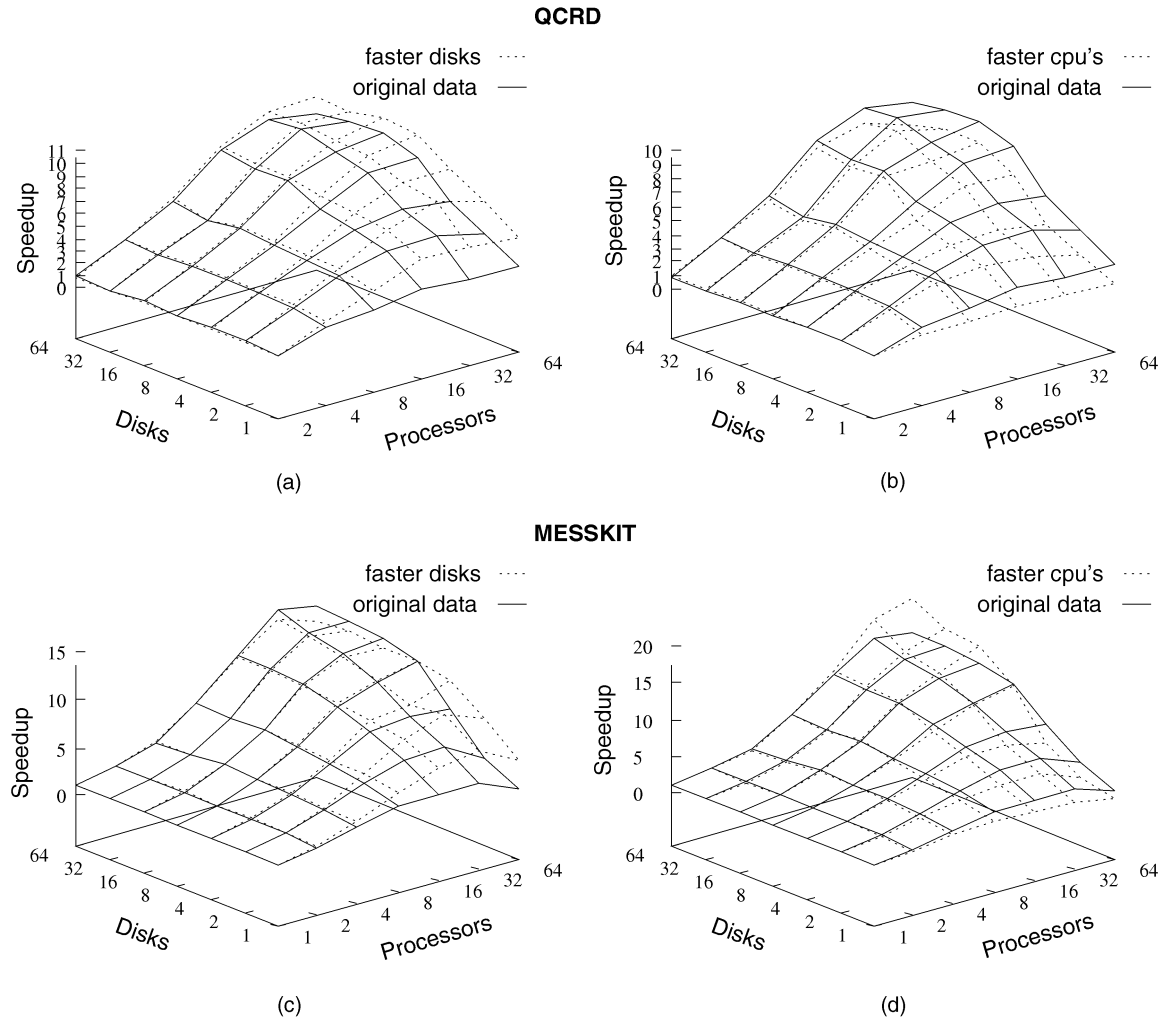

Fig. 9. Comparison of speedup surfaces for QCRD (a) and (b) and MESSKIT (c) and (d) at application level for the measured data (solid line in all graphs) and data derived assuming four-fold faster CPU's and disks.

to maintain consistency with the previous sections, we consider application scalability measures such as speedup or efficiency to analyze the impact of system upgrades. Moreover, given the definition of speedup in (10), our analysis herein can be used in a similar manner to consider other application measures, such as execution time. Finally, by combining our model with other mathematical models and methods (e.g., [38]), capacity planning issues can be investigated at the system level with more typical metrics, such as response time or throughput.

We first assume a baseline value for the speedup surface and then forecast the effects of the upgrading action using our model. A reference speedup surface is assumed to be available either from measurements or from previous modeling studies. We then examine the impact of system upgrades based on faster processors, i.e., processors that execute a given piece of code in a shorter time as is the case for processors with the same instruction set architecture but different clock speeds, or on faster disks, i.e., disks whose average access time is lower than that of the base case, or both.

We use the measurements presented throughout this paper as the baseline case, both at the application and program levels. We focus on the QCRD and MESSKIT applications. The speedup surfaces for both applications, at the application level, are depicted in Fig. 9. The original surfaces (solid lines) are compared with the surfaces (dashed lines) obtained for the same parallel system with faster processors or faster disks.

Figs. 9a and 9b compare the speedup surfaces for QCRD when four-fold faster disks and four-fold faster processors are assumed to be used, respectively. Significantly faster components are considered in order to stress the effects of the upgrading actions. Because the application is CPU-bound ($f^{\mathrm{CPU}} = 0.738$ and $f^{\mathrm{IO}} = 0.262$), faster processors are expected to improve performance. Although this is the case for the total computation time, the opposite effect is observed on speedup. As the figures show, with faster processors, the application does not scale as much as it does in the base case, while it improves with faster disks. To help explain these results, consider the relative values of the I/O and CPU scaling factors. The upgrading action should apply to the worst scaling resource, i.e., the resource that scales the least or, equivalently, has the largest scaling factors. Since the I/O scaling factors for each number of disks are greater than the CPU scaling

factors as shown in Tables 4 and 5, the upgrading action should concentrate on the disks. Thus, it is more beneficial to use faster disks rather than faster processors as, in this case, faster processors increase disk contention, which exceeds the reduction in CPU execution time, thus slowing down the overall execution. The maximum speedup gain or loss with respect to the baseline measure depends upon the improvement factor (e.g., four in our case) of the upgraded resource. It is interesting to note that, if the two programs are considered separately, they exhibit opposite behaviors with respect to scalability. While the first program, whose I/O scaling factors are greater than the CPU scaling factors, takes advantage of faster disks, the speedup of the second program improves with faster processors. In this case, the CPU scaling factors are greater than the I/O factors, thus suggesting that the upgrading action should concentrate on the processors.

The same relationship between relative values of the scaling factors and the variations in the application speedup is observed with MESSKIT, both at the program and application level. Whenever $s_{p,d}^{\mathrm{CPU}} > s_{p,d}^{\mathrm{IO}}$, faster processors yield better speedup and vice-versa. Figs. 9c and 9d show the corresponding speedup surfaces at the application level. Although the effects of faster resources are not consistent across the entire (p, d) space, these effects consistently reflect the relative values of the scaling factors for CPU and I/O. In particular, faster disks improve speedup only if their number is not greater than 4, while they are not effective when eight disks are used as they yield the same speedup as the original case. More than eight disks decrease speedup. Note that both speedup gains and losses are evident for large numbers of processors. When faster processors are used, we observe the opposite effect, i.e., speedup improves for large system configurations and worsens when the number of disks is small and the number of processors is large. The observed behavior reflects the scaling factors.

In the above analysis, disks and processors were considered separately. The impact of combined resource upgrades can also be analyzed. If the resource improvement factor is the same, no impact on speedup is observed as it is measured by a relative figure of merit. However, the overall execution time improves. If the improvement factors of the two resources are different, then the overall effect is a combination of the two individual ones, possibly compensating opposite effects.

A similar approach can be used when code restructuring and optimization are performed instead of resource upgrading. In this case, the application characteristics change, both in terms of scalability and relative fractions of I/O and CPU. By appropriately changing the parameters of (18), comparisons, such as those presented above, can be performed for these and other cases.

7 Conclusions

In this paper, we have presented a formal model of parallel program behavior in scientific applications that have large computational requirements and manipulate huge amounts of data. Our model captures the distinct phase behavior that characterizes these parallel programs. Speedup surfaces are presented as functions of the number of processors and disks allocated to the application. A form of Amdahl's law that explicitly considers the I/O component is presented and validated against measurement data collected as part of our study. The model effectively captures the variability and complexity observed in scientific parallel applications. It can be applied at different levels of granularity, thus providing various degrees of accuracy, cost, and flexibility in terms of parameter specification in order to meet the needs of the user.

Our model and results can provide benefits to both application programmers and operating system designers. For application developers, the model may provide suggestions for code optimization by restructuring the implementation and coalescing the existing operations. By running small benchmarks that represent different granularity of the I/O and CPU bursts within a phase, the application programmer may be able to quickly predict the application scalability using our model.

For operating system designers, our model can suggest ways to better manage the resources in a parallel system. The bursty nature of I/O and CPU activity observed in parallel programs is strikingly similar to the behavior observed in uniprocessor systems in the 1960s that motivated the development of uniprocessor time-sharing algorithms. This characterization suggests the development of time-sharing policies in high-performance parallel computing environments [38]. The availability of an application performance prediction tool can greatly facilitate the analysis and comparison of various scheduling policies.

Acknowledgments

The authors would like to thank the Caltech Center for Advanced Computing Research for providing access to the Intel Paragon XP/S where all of our experiments were run and data collected. Mark Wu and Aaron Kupperman at Caltech are thanked for providing the QCRD code. Rick Kendall and David Bernholdt of the Pacific Northwest Laboratory are thanked for providing the MESSKIT code. We also thank the anonymous referees for comments that helped to improve an earlier version of the paper. This work was partially supported by CESTIA-CNR, Milano, Italy, by the US National Science Foundation under grants EIA-9974992 and EIA-9977030, and by a William & Mary Summer Research Grant.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J.K. Hollingsworth, J. Saltz, and A. Sussman, "Tuning the Performance of I/O Intensive Parallel Applications," *Proc. Fourth Workshop Input/Output in Parallel and Distributed Systems,* pp. 15–27, May 1996.

[2] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proc. AFIPS Spring Joint Computing Conf.,* vol. 30, pp. 483–485, 1967.

[3] S. Asmussen, *Applied Probability and Queues.* John Wiley and Sons, 1987.

[4] S. Asmussen, "Phase-Type Distributions and Related Point Processes: Fitting and Recent Advances," *Matrix-Analytic Methods in Stochastic Models,* S.R. Chakravarthy and A.S. Alfa, ed., pp. 137–149, 1997.

[5] S.J. Baylor and C.E. Wu, "Parallel I/O Workload Characteristics Using Vesta," *Proc. IPPS '95 Workshop Input/Output in Parallel and Distributed Systems,* pp. 16–29, Apr. 1995.

[6] D. Bertsimas and D. Nakazato, "The Distributional Little's Law and its Applications," *Operations Research,* vol. 43, no. 2, pp. 298–310, 1995.

[7] S.L. Brumelle, "A Generalization of $L = \lambda W$ to Moments of Queue Length and Waiting Times," *Operations Research,* vol. 20, pp. 1127–1136, 1972.

[8] W. C, P. H, and M. V, "Parallel Computation of Electron-Molecule Collisions," *IEEE Computational Science and Eng.,* pp. 34–42, 1995.

[9] P. Crandall, R.A. Aydt, A.A. Chien, and D.A. Reed, "Input/Output Characterization of Scalable Parallel Applications," *Proc. Supercomputing 1995,* 1996.

[10] L. Dowdy and M. Leuze, "On Modeling Partitioned Multiprocessor Systems," *J. High Performance Computing,* vol. 1, no. 6, pp. 31–53, 1993.

[11] D.L. Eager, J. Zahorjan, and E.D. Lazowska, "Speedup versus Efficiency in Parallel Systems," *IEEE Trans. Computers,* vol. 38, no. 3, pp. 408–423, Mar. 1989.

[12] A. Feldmann and W. Whitt, "Fitting Mixtures of Exponentials to Long-Tail Distributions to Analyze Network Performance Models," *Performance Evaluation,* vol. 31, pp. 245–279, 1998.

[13] G. Ganger, B. Worthington, and Y. Patt, "The Disksim Simulation Environment, version 2.0," technical report, Carnegie Mellon Univ., 1998.

[14] D. Gaver, P. Jacobs, and G. Latouche, "Finite Birth-and-Death Models in Randomly Changing Environments," *Advances in Applied Probability,* vol. 16, pp. 715–731, 1984.

[15] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1990.

[16] High Performance Computational Chemistry Group, *NWChem, A Computational Chemistry Package for Parallel Computers, Version 1.1.* Richland, Wash.: Pacific Northwest National Laboratory, 1995.

[17] J. Huber, C.L. Elford, D.A. Reed, A.A. Chien, and D.S. Blumenthal, "PPFS: A High Performance Portable Parallel File System," *Proc. Nineth ACM Int'l Conf. Supercomputing,* pp. 385–394, July 1995.

[18] Intel Corporation, *Paragon System User's Guide.* Beaverton, Ore: Intel SSD, 1995.

[19] A.H. Karp and H.P. Flatt, "Measuring Parallel Processor Performance," *Comm. ACM,* vol. 33, no. 5, pp. 539–543, May 1990.

[20] D. Kotz and N. Nieuwejaar, "Dynamic File-Access Characteristics of a Production Parallel Scientific Workload," *Proc. Supercomputing '94,* pp. 640–649, Nov. 1994.

[21] D. Kotz, S.B. Toh, and S. Radhakrishnan, "A Detailed Simulation Model of the hp 97560 Disk Drive," Technical Report PCS-TR94-220, Darthmouth College, 1994.

[22] A. Lang and J.L. Arthur, "Parameter Approximation for Phase-Type Distributions," *Matrix-Analytic Methods in Stochastic Models,* S.R. Chakravarthy and A.S. Alfa, eds., pp. 151–206, 1997.

[23] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice Hall, 1984.

[24] T. Madhyastha and D.A. Reed, "Intelligent, Adaptive File System Policy Selection," *Proc. Frontiers '96,* 1996.

[25] E.L. Miller and R.H. Katz, "Input/Output Behavior of Supercomputer Applications," *Proc. Supercomputing '91,* pp. 567–576, Nov. 1991.

[26] M. Minsky and S. Papert, "On Some Associative, Parallel and Analog Computations," *Associative Information Technologies,* 1971.

[27] R. Nelson, *Probability, Stochastic Processes, and Queueing Theory–The Mathematics of Computer Performance Modeling.* Springer-Verlag, 1995.

[28] R.D. Nelson, "A Performance Evaluation of a General Parallel Processing Model," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems,* pp. 13–26, May 1990.

[29] M.F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins Univ. Press, 1981.

[30] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis, and M. Best, "File-Access Characteristics of Parallel Scientific Workloads," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 10, pp. 1075–1089, Oct. 1996.

[31] B.K. Pasquale and G.C. Polyzos, "A Static Analysis of I/O Characteristics of Scientific Applications in a Production Workload," *Proc. Supercomputing '93,* pp. 388–397, 1993.

[32] B.K. Pasquale and G.C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications," *Proc. Supercomputing '94,* pp. 660–669, Nov. 1994.

[33] J.T. Poole, "Preliminary Survey of I/O Intensive Applications," Technical Report CCSF-38, Scalable I/O Initiative, Caltech Concurrent Supercomputing Facilities, California Institute of Technology, Altadena, 1994.

[34] J.T. Poole, "Scalable I/O Initiative," California Institute of Technology, Available at http://www.ccsf.caltech.edu/SIO/, 1996.

[35] A. Purakayastha, C.S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor," *Proc. Ninth Int'l Parallel Processing Symp.,* pp. 165–172, Apr. 1995.

[36] D.A. Reed, R.A. Aydt, R.J. Noe, P.C. Roth, K.A. Shields, B.W. Schwartz, and L.F. Tavera, "Scalable Performance Analysis: The Pablo Performance Analysis Environment," *Proc. Scalable Parallel Libraries Conf.,* A. Skjellum, ed., pp. 104–113, 1993.

[37] D.A. Reed, C.L. Elford, T. Madhyastha, E. Smirni, and S.L. Lamm, "The Next Frontier: Interactive and Closed Loop Performance Steering," *Proc. 1996 Int'l Conf. Parallel Processing Workshop,* pp. 20–31, Aug. 1996.

[38] E. Rosti, G. Serazzi, E. Smirni, and M. Squillante, "The Impact of I/O on Program Behavior and Parallel Scheduling," *Proc. ACM SIGMETRICS '98 Conf. Measurement and Modeling of Computer Systems and Performance '98,* pp. 57–65, June 1998.

[39] K.C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems," *Performance Evaluation–An Int'l J.,* vol. 19, nos. 2-3, pp. 107–140, Mar. 1994.

[40] H. Simitci and D. Reed, "A Comparison of Logical and Physical Parallel I/O Patterns," *Int'l J. High Performance Computing Applications,* vol. 12, no. 3, pp. 364–380, Fall 1998.

[41] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY-LoadLeveler API Project," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 41–47, 1996.

[42] E. Smirni, R.A. Aydt, A.A. Chien, and D.A. Reed, "I/O Requirements of Scientific Applications: An Evolutionary View," *Proc. High Performance Distributed Computing,* pp. 49–59, 1996.

[43] E. Smirni, C. Elford, A. Lavery, and A. Chien, "Algorithmic Influences on I/O Access Patterns and Parallel File System Performance," *Proc. 1997 Int'l Conf. Parallel and Distributed Systems,* pp. 794–801, Dec. 1997.

[44] E. Smirni and D.A. Reed, "Lesson from Characterizing the Input/Output Behavior of Parallel Scientific Applications," *Performance Evaluation,* vol. 33, pp. 27–44, 1998.

[45] M.S. Squillante, "A Matrix-Analytic Approach to a General Class of G/G/c Queues," Technical report, IBM Research Division, May 1996.

[46] The MPI-IO Committee, *MPI-IO: A Parallel File I/O Interface for MPI,* Version 0.5. Apr. 1996.

[47] A. Thomasian and P. Bay, "Analysis Techniques for Queueing Network Models of Multi-Computer Systems with Shared Resources," *Computer Performance,* vol. 4, no. 3, pp. 151–166, Sept. 1983.

[48] S. Toledo and F.G. Gustavson, "The Design and Implementation of SOLAR, a Portable Library for Scalable Out-of-Core Linear Algebra Computations," *Proc. Fourth Workshop Input/Output in Parallel and Distributed Systems,* pp. 28–40, May 1996.

[49] M. Uysal, A. Acharya, and J. Saltz, "Requirements of I/O Systems for Parallel Machines: An Application-Driven Study," Technical Report CS-TR-3802, Univ. of Maryland, College Park, May 1997.

[50] W. Whitt, "A Review of $L = \lambda W$ and Extensions," *Queueing Systems,* vol. 9, pp. 235–268, 1991.

[51] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *ACM Trans. Computer Systems,* vol. 14, no. 1, pp. 108–136, Feb. 1996.

[52] R.W. Wolff, *Stochastic Modeling and the Theory of Queues.* Prentice Hall, 1989.

[53] Y.-S.M. Wu, S.A. Cuccaro, P.G. Hipes, and A. Kuppermann, "Quantum Chemical Reaction Dynamics on a Highly Parallel Supercomputer," *Theoretica Chimica Acta,* vol. 79, pp. 225–239, 1991.

Emilia Rosti received the PhD degree in computer science in 1993 and is an associate professor in the Computer Science Department of the University of Milan. She has spent several periods of time abroad collaborating with foreign academic institutions. Her scientific interests are in computer and network security, computer system performance evaluation, parallel and distributed systems, and allocation policies for multiprocessor systems. She has served on the steering committee of FIRST, the forum of incident response, and security teams. She coauthored various papers that have appeared in international journals and conferences. She is a member of the IEEE.

Giuseppe Serazzi received a Laurea Degree in mathematics from University of Pavia, Italy, in 1969. He is professor in the Computer Science Department of Politecnico di Milano, Milano, Italy. He spent four years at the University of Milano before joining the Politecnico di Milano in 1991. From 1978 to 1987, he was an associate professor of the Department of Mathematics at the University of Pavia, Italy. His current research interests include modeling and other topics related to performance evaluation of computer systems, Web servers, Intranets, Internet, and high-performance computing. He is a member of the IEEE Computer Society.

Evgenia Smirni received a Diploma in computer engineering and informatics from the University of Patras, Patras, Greece, in 1987, and the MS and PhD degrees in computer science from Vanderbilt University in 1993 and 1995, respectively. From August 1995 to June 1997, she had a postdoctoral research associate position at the University of Illinois at Urbana-Champaign. She is currently an assistant professor in the Department of Computer Science at the College of William and Mary, Williasburg, Virginia. Her research interests include analytic modeling, resource allocation policies, parallel input/output, workload characterization, and modeling of distributed systems and applications. She is a member of the IEEE Computer Society.

Mark S. Squillante received the PhD degree in computer science from the University of Washington, Seattle, WA, in 1990. He has been a research staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, since 1990, and an adjunct faculty member of the Department of Computer Science at Columbia University, New York, from 1991 through 1996. He currently manages the Systems Analysis and Optimization group. From 1982 to 1985, he was a member of the technical staff at Bell Telephone Laboratories, Murray Hill, New Jersey. His research interests concern fundamental problems in the mathematical analysis, modeling, and optimization of computer systems, applications, and services. Dr. Squillante is a senior member of the IEEE and a member of the ACM, IFIP W.G. 7.3, INFORMS and SIAM. He currently is an associate editor of *Operations Research* and an editor of *Performance Evaluation*.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.