Integrated Performance Models for SPMD Applications and MIMD Architectures

Paolo Cremonesi and Claudio Gennaro

Abstract—This paper introduces queuing network models for the performance analysis of SPMD applications executed on general-purpose parallel architectures such as MIMD and clusters of workstations. The models are based on the pattern of computation, communication, and I/O operations of typical parallel applications. Analysis of the models leads to the definition of speedup surfaces which capture the relative influence of processors and I/O parallelism and show the effects of different hardware and software components on the performance. Since the parameters of the models correspond to measurable program and hardware characteristics, the models can be used to anticipate the performance behavior of a parallel application as a function of the target architecture (i.e., number of processors, number of disks, I/O topology, etc).

Index Terms—Single program multiple data (SPMD), multiple instruction multiple data (MIMD), performance model, queuing network model, fork-join queues, mean value analysis (MVA), parallel I/O, synchronization overhead, speedup surface.

1 Introduction

The objective of this paper is to study the performance of general-purpose parallel architectures when executing SPMD (Single Program Multiple Data) applications. A family of queuing network models is introduced that describes the behavior of SPMD applications under the assumption that the applications have a "regular" cyclic structure. The models require the knowledge of some hardware characteristics of the target architecture, which can be obtained by running simple benchmark programs available in literature. Furthermore, the knowledge of a few characteristics of the application being executed, like the amount of data transferred during the communication, and I/O phases, is needed. Nevertheless, the approach proposed is capable of modeling a wide range of parallel applications (from I/O bound to communication intensive applications).[1]

The parallel systems under consideration have a generic architecture (distributed or shared memory) with a finite number of homogeneous processors and a finite number of I/O devices (e.g., disks). The systems are assumed to be monoprogrammed and executing one task per processor. The performance analysis is limited to SPMD applications. Communication can be achieved via explicit message passing or by referring to shared variables, and can be synchronous or asynchronous. I/O operations can be reads and/or writes.

1. EDITOR'S NOTE: This paper originally appeared in *TPDS*, Vol. 13, No. 7. Unfortunately, due to extraordinary circumstances, errors were introduced into the figure captions of the paper. We are reprinting the paper in its entirety here.

- *P. Cremonesi is with Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. E-mail: cremones@elet.polimi.it.*
- *C. Gennaro is with Istituto di Elaborazione della Informazione, CNR, Pisa, Italy, E-mail: gennaro@iei.pi.cnr.it.*

Manuscript received 5 Jan. 2000; revised 17 Aug. 2001; accepted 13 Feb. 2002. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111195.

In a typical SPMD program, all the processes alternate computation, communication, and I/O in a cyclic fashion. Studies on the characterization of scientific applications [1], [2], [3], [4] show that most data-parallel applications present a behavior which is rather regular and cyclic along time. The properties of many scientific parallel programs result in an execution behavior that can be naturally partitioned into disjoint intervals, each of which consists of one computation burst followed by one I/O burst (see Fig. 1).

Examples of scientific parallel applications with regular and cyclic behavior are: H3expresso, a parallel solver for the full 3D Einstein equations used to construct dynamic black hole spacetimes [5]; PRISM, a parallel implementation of a 3D numerical simulation of the NavierStokes equations [6]; ESCAT, a parallel implementation of the Schwinger Multichannel method for calculating low-energy electronmolecule collisions [6]. More examples can be found in [7], [8].

Extending the model presented in [9] and [10], we consider a SPMD program to be composed of one or more *phases*. Each phase is in turn composed of a number of statistically identical CPU, communication and, eventually, I/O bursts that are executed in a cyclic fashion. Different phases correspond to different stages of a multistage application (e.g., preprocessing, core computation, postprocessing). The cyclic structure of each phase usually derives from the presence in the application of outer iteration loops (e.g., the time evolution of some physical model, or the convergence of some approximate algorithm). The paper shows that, for each phase, it is possible to define a queuing network model that describes the performance of the coupled application/architecture system.

The paper is organized as follows: Section 2 describes other works related to the performance modeling of parallel applications. Section 3 formally defines the class of parallel applications and systems to which our analysis can be applied. The section also describes the computation, communication, and I/O queuing network submodels. Section 4 integrates the submodels into a family of coupled application/architecture models. Section 5 discusses the

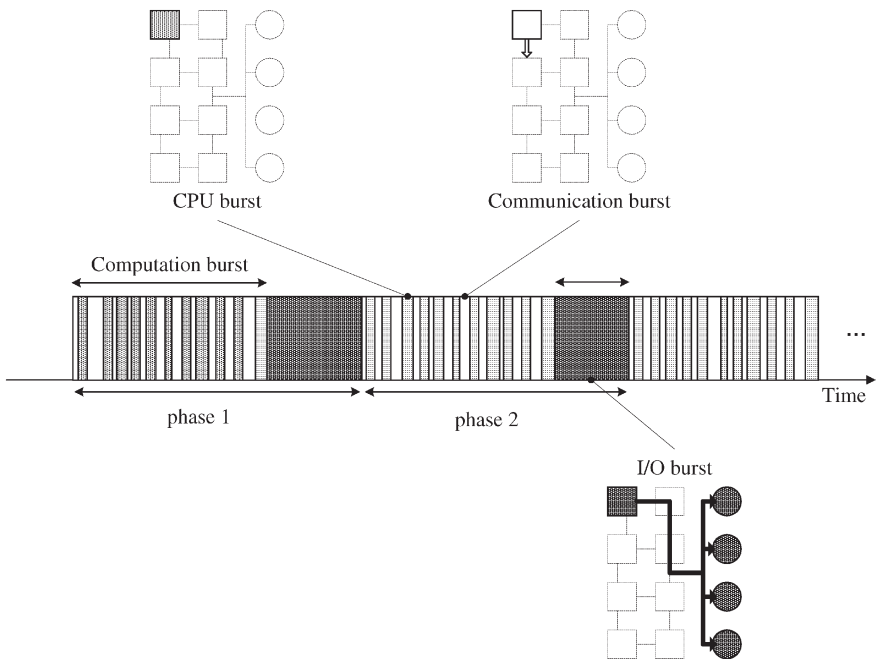

Fig. 1. An example of a program phase decomposed in cycles and bursts.

validation of the models by comparing the predicted performance results with those of real parallel applications. The applications considered are BTIO and QCRD. Finally, Section 6 summarizes the results and the plan for future research.

2 Related Works

The first effort to model the performance of parallel programs is due to Amdahl and the well-known homonymous law that relates the number of processors to the bound of the speedup which may be expected by parallel processing [11]. Several extensions to Amdahl's law have been proposed. Gustafson in [12] and [13] and Gustafson et al. in [14] introduce the concept of scaled speedup. Flat and Kennedy [15] investigate the impact of synchronization and communication overhead on the performance of parallel processors. Eager et al. [16] studied speedup versus efficiency. Wu and Li [17] propose a formal definition of scalability and discuss scalability of cluster systems. Rosti et al. [9] extends Amdahl's law to three-dimensional space to include processors and disks. All the above approaches have the advantage of expressing analytically the speedup in a closed form. However, these models are limited because they neglect the effects of important architecture characteristics.

Other approaches use task graph models to represent the intertask precedence relations within a program, the task execution times, and the amount of data transferred among tasks. In the case of program control structures that can be represented by "series-parallel" task graphs, such as the fork-join structure, methodologies have been derived to predict the best speedup. When the task execution times are deterministic, Coffman and Denning [18] use critical path analysis to find the program completion time. If the task execution times are stochastic and the number of processors is infinite, the probability distribution of execution time can be determined by a straightforward but in general very costly computation [19], [20]. General acyclic random graph models are presented in [21], [22], [23]. Gelenbe [24] generalizes the task graph model to take into account the effects of communications.

For more realistic cases, where the number of processors is smaller than the number of tasks in a program, queuing network techniques can be used, where processors are modeled as service centers and parallel tasks are modeled with requests circulating in the system. All the approaches describe multiprogrammed and multitasked parallel systems executing a sequence of programs of similar task structure. The models are based on fork-join, open queuing networks, where a stream of serial-parallel applications arrives in the system. Lui et al. [25] compute performance bounds for fork-join parallel program. Bacelli and Lui [26] propose a class of models for homogeneous parallel systems executing application with a general task structure. Balsamo et al. [27] describe some approximate models for the analysis of heterogeneous parallel systems. Apon and Dowdy [28] introduce a queuing network circulating processor model that differs from previous models in that the processors (the requests) circulate among the processes (the resources). Qin and Baer [29] adopt a similar technique to model the performance of cluster-based architectures where each cluster is a shared-based multiprocessor. All the above works focus mainly on finding analytical solutions to queuing network models with resource contention and synchronization barriers. However, the assumption behind the models are better suited for multiprocessor systems sharing a central memory than for distributed memory systems and do not take into consideration communication, and I/O contention. Moreover, only [29] addresses the modeling of monoprogrammed systems.

Very little has been done to model the performance of I/O operations in parallel applications. Most of the works

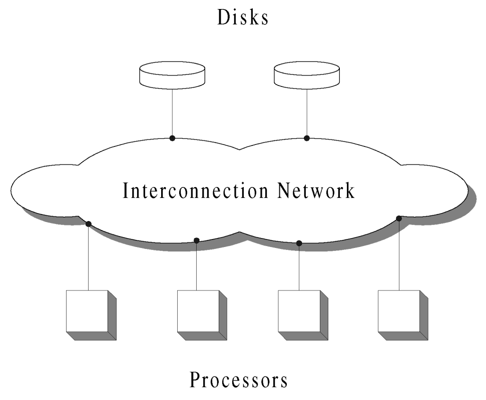

Fig. 2. General purpose parallel machine.

focus on the analysis of RAID (Redundant Arrays of Inexpensive Disks) systems [30] under different workload situations. Catania et al. [31] apply simulation techniques to study RAID architectures with dynamic declustering. They also provide some analytical lower bounds. Kotz [32] describes different implementations of collective I/O operations in MIMD architectures (simple parallel file system, two-phase I/O and disk-direct I/O). Simulation is used to compare the performance of the different implementations.

3 Architecture and Application Models

Throughout the discussion, we consider a monoprogrammed and monotasked parallel system with a generic structure as the one illustrated in Fig. 2. We assume the parallel system to be composed of p homogeneous computational processors and d homogeneous disks. Processors and disks are connected to each other via a generic communication network.

Improving the application model presented in [9], we consider an SPMD program as a sequence of M *phases*. Each phase m is composed of a sequence of $N(m)$ statistically identical *cycles*. Each cycle consists of a number of computation bursts followed by one I/O burst. The computation burst is, in turn, composed of one burst of CPU activity followed by one burst of communication, as show in Fig. 1. If the SPMD application is well designed (i.e., load-balanced), the computation, communication, and I/O loads have the same order of magnitude for all the processors. Within each phase, we therefore assume that CPU, communication, and I/O bursts are statistically identical, i.e., that their duration are sampled from an exponential distribution with average value S^{CPU}, S^{COM}, and S^{I}, respectively. This assumption allows us to model almost all the system by means of single class queuing networks.

3.1 Communication Subsystem

Among different processor interconnection networks, two extreme situations can be identified: a fully interconnected system and a single bus system. In the former case, no link contention arises in any communication. Messages exchanged between pairs of processors experience a simple delay regardless of the network usage. In the latter case, when two or more messages are exchanged simultaneously, there is contention for the use of the shared bus. Other interconnection networks, e.g., meshes or trees, represent trade offs between the two cases. The impact of contention on communications depends also on the parallel application. For instance, a pipeline implemented on a parallel machine with a mesh network can be realized in such a way that communications do not suffer from contention.

Different types of interconnection networks lead to different ways of modeling communications. A shared bus can be modeled as a queuing server, since it is capable of handling one message at a time, while a fully interconnected network can be modeled as a delay center since messages never queue for a link. More realistic network topologies (e.g., 2D meshes, hypercubes, toruses) are more complex to model (they require the exact knowledge of the network routing mechanism).

We define the *communication contention level* w, a real number $0 \leq w \leq 1$, that allows us to switch from the bus interconnection network architecture ($w = 1$) to the fully connected one ($w = 0$). Intermediate values represent different network architectures such as meshes and trees. The value of w depends on the communication hardware as well as on the algorithm implemented. However, it is difficult to predict the correct value of w, except for the boundary cases $w = 0$ and $w = 1$. When the value of w is unknown, it can be guessed by fitting model results with experimental performance measurements, as shown is Section 3. Alternatively, the two boundary values $w = 0$ and $w = 1$ can be used to obtain an optimistic and a pessimistic model.

3.2 I/O Subsystem

Real parallel applications exploit I/O for different purposes [33]: *checkpoints* to reduce the cost of system failures, saving of *simulation data* for subsequent visualization or analysis, *out-of-core computation* when program data structures are larger than available memory, and *data retrieval* when the application involves the analysis of large amounts of data.

I/O parallelism consists of using more disks and one or more controllers and distributing data across the disks. In RAID, systems files are interleaved across an array of disks sharing a common bus and controlled by a single controller. A more efficient and flexible parallel disk architecture is constituted by a set of independent disks, each disk with a separate controller and connected to a distinct processor of the parallel machine. The individual devices in a parallel independent disk system could themselves be arrays of disks.

Usually, processors with disks are not loaded with computational tasks, but they are dedicated to the handling of I/O activities. Such activities may include post-processing or visualization of output data. The I/O nodes often provide the user with the abstraction of a single, shared, file system. For example, the Panda parallel file system [34] is designed for an environment where processors are full-time compute nodes or full-time I/O nodes. Several commercial machines exhibit this type of high level architecture, including the Thinking Machines CM5, the Intel Paragon, the IBM-SP2, and the Cray T3E [35], [36], [37].

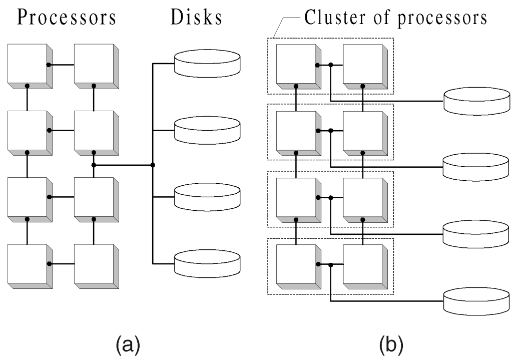

Fig. 3. I/O configurations considered: (a) BUS (b) CLU.

According to the architecture of the I/O subsystem, we consider two family of models (Fig. 3):

- *BUS models*: A single disk or a pool of disks (e.g., a RAID system) is connected via a single I/O node to the system.
- *CLU models*: Computational processors are logically or physically grouped (e.g., *clusterized*) and each cluster contains one I/O node.

3.3 Synchronization Model

Synchronization happens during communication and I/O activities. The performance cost of synchronization operations (e.g., synchronous communications, barriers, synchronous I/O, etc.) is a function of the number of processors involved. Within each phase, we assume that the number of processors participating to synchronization events (communications or I/O) does not change.

In order to capture the effects of synchronization during communications, we define the *synchronization level* c as the number of processors ($1 \leq c \leq p$) involved in a synchronous communication. According to the definition, processors can be thought as grouped into p/c *synchronization groups*. For instance, if $p = 8$ and $c = 2$, we have $p/c = 4$ groups of processors, each group composed of $c = 2$ processors. Processors belonging to the same group communicate with each other by means of synchronous send/receive operations. When $c = 1$, all the processors perform asynchronous communications.

Studies on the characterization of parallel scientific file accesses [38] show that most I/O activities are fully synchronous or fully asynchronous. According to the type of I/O, we therefore construct two family of models:

- *Synchronous I/O models (SIO):* Processors perform I/O bursts synchronously (e.g., checkpoints and simulation data).
- *Asynchronous I/O Models (AIO)*: Processors perform I/O bursts asynchronously (e.g., out-of-core computations).

3.4 Speedup Models

Throughout the rest of the paper, we study the performance of parallel programs composed of one phase (i.e., $M = 1$). The results can be extended to multiphase applications. We also assume that the duration of the phase (i.e., the number N of cycles) is long enough to allow the system to reach a steady-state condition.

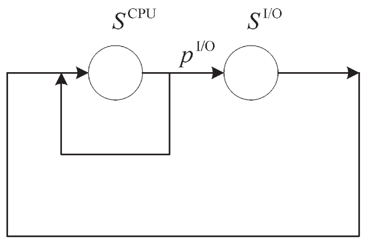

Fig. 4. Queuing network model of a program executed on a single processor and disk.

In this section, we develop a performance model for a program executed on a system composed of one processor and one disk. We use this model as a reference model. Let $p^{\rm I/O}$ be the probability of an I/O burst at the end of a computation burst ($n^{\rm I/O} = 1/p^{\rm I/O}$ is the average number of computation bursts between two successive I/O bursts). Let $S^{\rm CPU}$ and $S^{\rm I/O}$ be the average service time of CPU burst and I/O burst, respectively. Fig. 4 depicts the closed queuing network corresponding to the reference model. A single job, representing the program in execution, circulates in the queuing network. The average response time of the circulating job is

$$T_1 = S^{\rm CPU} n^{\rm I/O} + S^{\rm I/O}. \tag{1}$$

Note that T_1 is the average execution time of one cycle of the program. Throughout the paper, we use T_1 as a reference time in order to evaluate the speedup of different parallel programs and architectures.

Consider now a program executed on a parallel system with p processors and d disks. We can decompose the average execution time $T(p, d)$ into three terms

$$T(p,d) = N\Big[T^{\rm CPU}(p,d) + T^{\rm COM}(p,d) + T^{\rm I/O}(p,d)\Big], \tag{2}$$

where N is the number of cycles in the program (i.e., in the phase), $T^{\rm CPU}(p,d)$, $T^{\rm COM}(p,d)$, and $T^{\rm I/O}(p,d)$ represent the average time of the CPU, communication, and I/O bursts, respectively.

The speedup s can be expressed as

$$s(p,d) = \frac{T^{\rm CPU}(1,1) + T^{\rm I/O}(1,1)}{T^{\rm CPU}(p,d) + T^{\rm COM}(p,d) + T^{\rm I/O}(p,d)}. \tag{3}$$

Fig. 5 shows the generic queuing networks used for modeling the parallel program. The blocks represent "submodels" corresponding to computation and I/O bursts. In AIO applications, the jobs representing the computation burst perform I/O independently. In SIO applications, a pair of fork/join stations are added to the computation burst subsystem. When the jobs representing the computation burst activities are collected by the join station, they are replaced by a single job that corresponds to the I/O burst activity.

3.5 CPU Submodel

The relationship between computation load and number of processors is well approximated by Amdahl's law. The

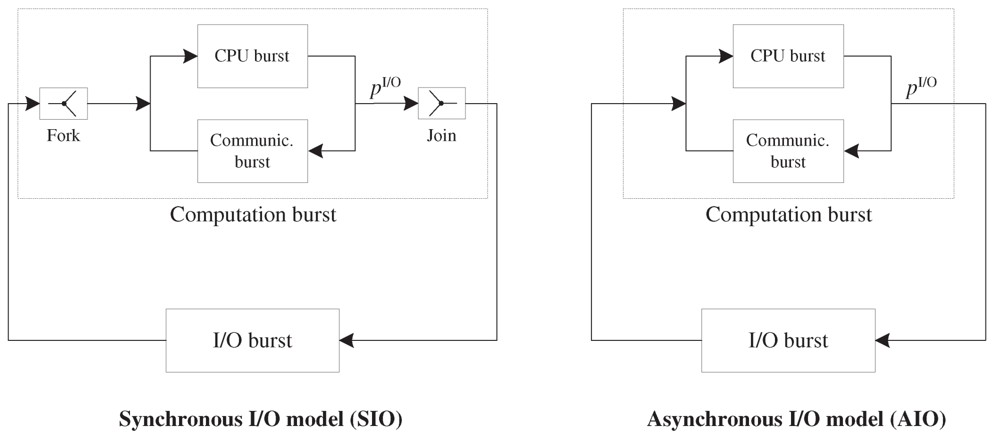

Fig. 5. Structure of the queuing network used for modeling parallel programs.

CPU load S^{CPU} can be divided into a parallel and a serial component, $S^{\mathrm{CPU}}_{\mathrm{par}}$ and $S^{\mathrm{CPU}}_{\mathrm{ser}}$, respectively. According to Amdahl's law, the CPU load on p processors is $S^{\mathrm{CPU}}_{\mathrm{par}}/p + S^{\mathrm{CPU}}_{\mathrm{ser}}$ (the parallel component scales perfectly with the number of processors, while the serial component is not affected).

Fig. 6 shows the queuing network that integrates CPU and communication submodels into a computational burst model. A job entering in the subnetwork represents a group of c processors communicating among themselves by means of synchronous communication operations. Each job proceeds to a set of delays that model the processor CPU time, then to the two service centers representing the communication network (the communication submodel). On average, a job performs $n^{\mathrm{I/O}}$ cycles before leaving the computation burst subnetwork. The maximum number of jobs circulating in the computational submodel is p/c.

The upper part of Fig. 6 refers to the CPU submodel. The processors are represented by p/c delays and as many pairs of fork/join stations. A job arriving in the fork station is split into c jobs, each of them representing a single processor computation. Each job is collected by a delay station (with service time $S^{\mathrm{CPU}}_{\mathrm{par}}/p + S^{\mathrm{CPU}}_{\mathrm{ser}}$) that represents pure CPU calculation. Finally, the jobs are recollected by the join station and, when all the jobs have been received, they are replaced with a single job representing the group of synchronized processors.

The time elapsed between the arrival of the job in the fork station and the departure from the join station is a random variable defined as $X = \max\{X_1, \ldots, X_c\}$, where

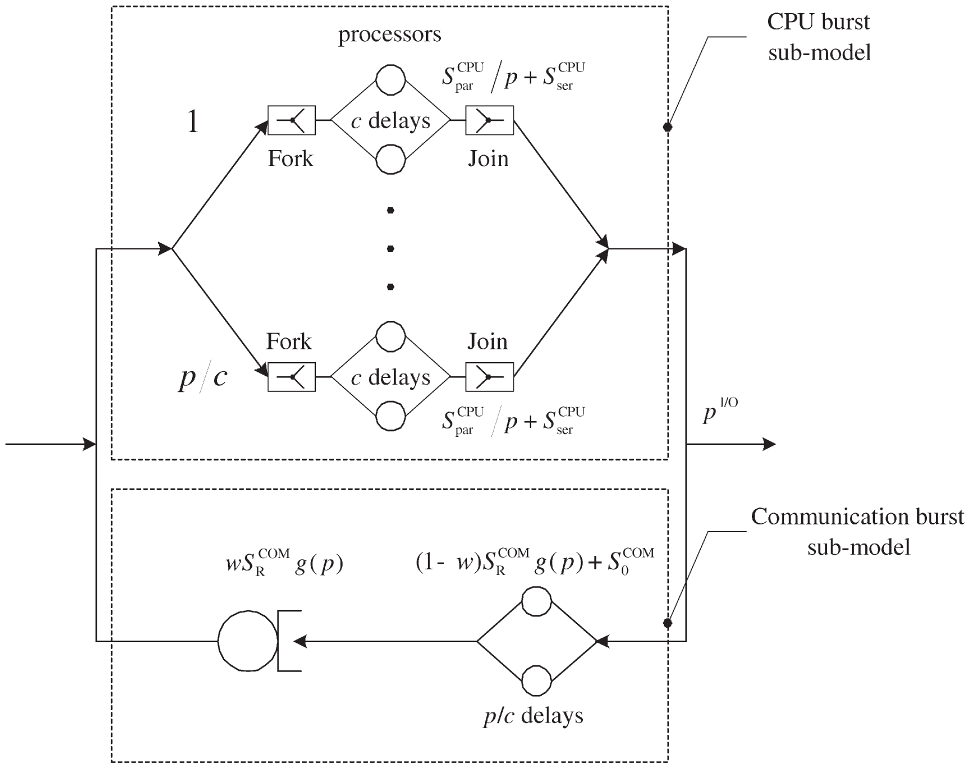
processors
$S^{\mathrm{CPU}}_{\mathrm{par}}/p + S^{\mathrm{CPU}}_{\mathrm{ser}}$
1
Fork
c delays
Join
CPU burst sub-model
Fork
Join
p/c
c delays
$S^{\mathrm{CPU}}_{\mathrm{par}}/p + S^{\mathrm{CPU}}_{\mathrm{ser}}$
$p^{\mathrm{I/O}}$
Communication burst sub-model
$wS^{\mathrm{COM}}_{\mathrm{R}}g(p)$
$(1-w)S^{\mathrm{COM}}_{\mathrm{R}}g(p) + S^{\mathrm{COM}}_{0}$
p/c delays

Fig. 6. Subnetwork that simulates the computational burst.

$X_1, \ldots, X_c$ are random variables exponentially distributed with the same mean $S_{\text{par}}^{\text{CPU}}/p + S_{\text{ser}}^{\text{CPU}}$, representing the c processor delays. X is then a hypoexponential distributed random variable with mean given by [39]

$$E[X] = \left(\frac{S_{\text{par}}^{\text{CPU}}}{p} + S_{\text{ser}}^{\text{CPU}} \right) \sum_{i=1}^{c} \frac{1}{i}. \tag{4}$$

Therefore, the average response time of the fork/join pair, can be written as

$$T^{\text{CPU}} = h(c) \left(\frac{S_{\text{par}}^{\text{CPU}}}{p} + S_{\text{ser}}^{\text{CPU}} \right), \tag{5}$$

where $h(c)$ is the synchronization cost function

$$h(c) = \sum_{i=1}^{c} \frac{1}{i}. \tag{6}$$

The assumption of exponentially distributed times in the fork-join model can lead to pessimistic results when the variability in work per processor of the real application is very low and the number of processors is small [40]. Different and less pessimistic distributions can be used, by changing the synchronization cost function $h(c)$. An example is the uniform distribution with average values $S_{\text{par}}^{\text{CPU}}$ and $S_{\text{ser}}^{\text{CPU}}$. In this case, the cost function becomes $h(c) = 2c/(c+1)$. However, if the distribution is different from exponential, the queuing network solution techniques presented in Section 4 cannot be used.

3.6 Communication Submodel

We introduce a communication scale function $g(p)$, which measures how the average amount of data transmitted by each processor scales (during a communication burst). By definition, $g(1) = 0$. The choice of the scaling function $g(p)$ depends on the algorithm implemented but not on the target architecture. Through the scaling function $g(p)$, we are able to capture a broad class of parallel applications.

SPMD applications typically work with r-dimensional arrays that are distributed in a block fashion among processors. The amount of interprocessor communication depends on the hyperperimeter of the distributed data structure, while the computational load depends on the hypervolume. For instance, in low-level image processing, the working data structure is a two-dimensional matrix of pixels, partitioned into square windows among the processors. The size of the messages exchanged between pair of processors is proportional to the length of the internal border between adjacent windows ($1/\sqrt{p}$), while the size of the overall communications is proportional to $\sqrt{p}$. We can generalize the idea for a generic number r of dimensions in the application data space, obtaining

$$g(p) = \frac{1}{p^{\frac{r-1}{r}}}. \tag{7}$$

If the communication network has no contention, the communication service time T^{COM} can be well approximated by a linear relation

$$T^{\text{COM}} = S_0^{\text{COM}} + g(p) S_{\text{R}}^{\text{COM}}, \tag{8}$$

where S_0^{COM} is the communication startup time and $S_{\text{R}}^{\text{COM}}$ is the reciprocal of the communication transfer rate.

The lower part of Fig. 6 shows the communication submodel, containing one delay station and one queuing station. The service time of the queuing station $S_{\text{queue}}^{\text{COM}}$ takes into account the fraction of the transfer time affected by the network contention

$$S_{\text{queue}}^{\text{COM}} = w g(p) S_{\text{R}}^{\text{COM}}. \tag{9}$$

The service time $S_{\text{delay}}^{\text{COM}}$ of the delay station is the sum of two components

$$S_{\text{delay}}^{\text{COM}} = S_0^{\text{COM}} + (1-w) g(p) S_{\text{R}}^{\text{COM}}. \tag{10}$$

The first component is the startup time. The second component is the fraction of the transfer time that is not affected by the network contention.

3.7 I/O Submodel

It is difficult to provide a general discussion of parallel I/O because different parallel computers have radically different I/O architectures and hence parallel I/O mechanisms.

In SIO models, the I/O load depends on the number of disks but not on the number of processors. Amdahl's law can be applied to I/O parallel file systems as a function of the number d of disks

$$T^{\text{I/O}} = S_0^{\text{I/O}} + \frac{S_{\text{R}}^{\text{I/O}}}{d}, \tag{11}$$

where $S_0^{\text{I/O}}$ and $S_{\text{R}}^{\text{I/O}}$ are the serial and parallel part of the I/O service time with respect to the number d of disks.

In AIO models, I/O operations are divided into p/c chunks. In BUS-AIO models, each chunk of I/O data is striped among the d disks. The I/O service time of one chunk of I/O is given by

$$T^{\text{I/O}} = S_0^{\text{I/O}} + \frac{S_{\text{R}}^{\text{I/O}}/d}{p/c}. \tag{12}$$

In CLU-AIO models, processors are logically grouped into clusters and each cluster shares one I/O node. I/O clustering occurs mainly because of application design issues but can be also motivated by the parallel architecture configuration. Clustering occurs among the p/c synchronization groups. Therefore, p/c must be an integer multiple of d (i.e., $p/c = kd$, where k is the number of synchronization groups sharing one disk). The I/O service time for one chunk of I/O is given by

$$T^{\text{I/O}} = S_0^{\text{I/O}} + \frac{S_{\text{R}}^{\text{I/O}}}{p/c}. \tag{13}$$

We can summarize the I/O service time for different I/O architectures

$$T^{\text{I/O}} = \begin{cases} S_0^{\text{I/O}} + S_{\text{R}}^{\text{I/O}}/d & \text{SIO} \\ S_0^{\text{I/O}} + \frac{S_{\text{R}}^{\text{I/O}}/d}{p/c} & \text{BUS}-\text{AIO} \\ S_0^{\text{I/O}} + \frac{S_{\text{R}}^{\text{I/O}}}{p/c} & \text{CLU}-\text{AIO}. \end{cases} \tag{14}$$

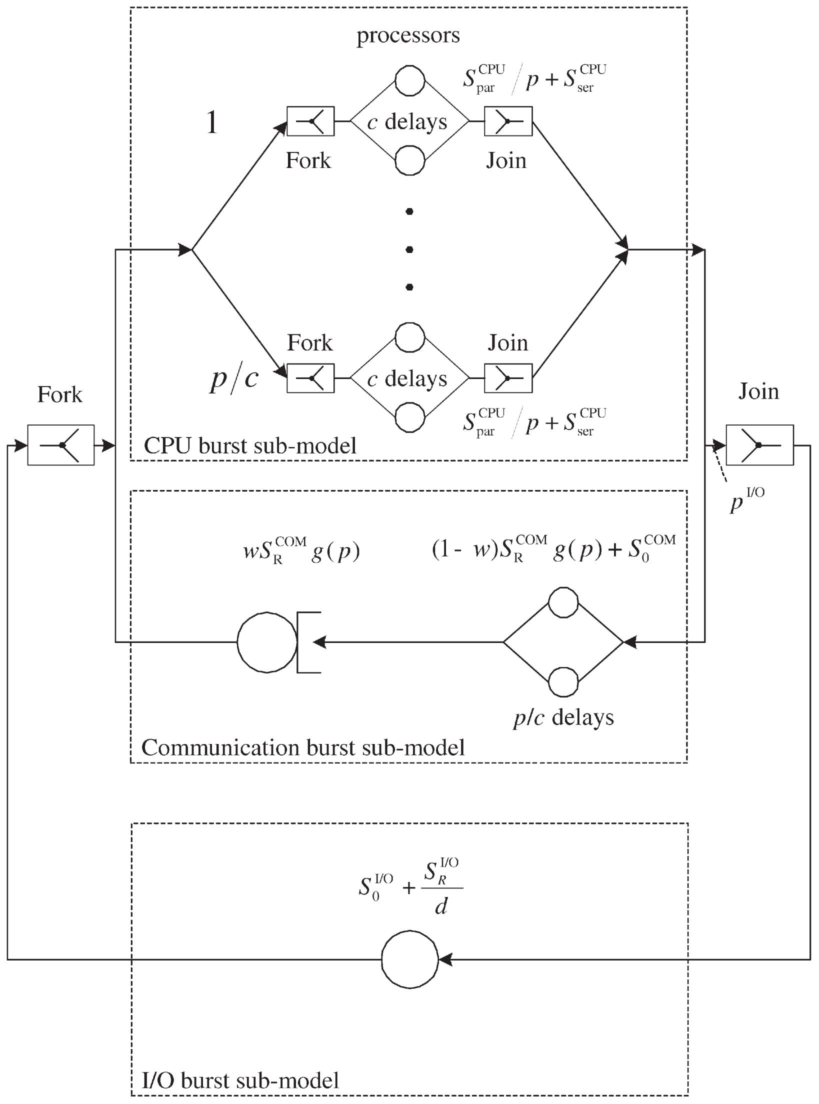
processors
$S_{par}^{CPU}/p + S_{ser}^{CPU}$
1
Fork
c delays
Join
Fork
p/c
c delays
Join
$S_{par}^{CPU}/p + S_{ser}^{CPU}$
Fork
Join
CPU burst sub-model
$p^{I/O}$
$wS_R^{COM}g(p)$
$(1-w)S_R^{COM}g(p)+S_0^{COM}$
p/c delays
Communication burst sub-model
$S_0^{I/O}+\frac{S_R^{I/O}}{d}$
I/O burst sub-model

Fig. 7. SIO queuing network model.

4 Model Analysis

In this section, we describe how to integrate the submodels described in the previous sections and we develop and illustrate the algorithms required to evaluate the performance results.

4.1 SIO Model

The SIO model describes parallel algorithms performing synchronous I/O operations (see Fig. 7). In order to estimate the speedup of the SIO model, we first evaluate the time $T^{\mathrm{CPU}} + T^{\mathrm{COM}}$ spent by the program during the computation burst by computing the response time of the computation burst subnetwork.

Let z be the sum of all the delays belonging to the computation subnetwork

$$z = h(c)\left(\frac{S_{\mathrm{par}}^{\mathrm{CPU}}}{p} + S_{\mathrm{seq}}^{\mathrm{CPU}}\right) + S_0^{\mathrm{COM}} + (1-w)g(p)S_{\mathrm{R}}^{\mathrm{COM}}. \quad (15)$$

The approximate response time for the computation fork-join subnetwork is

$$T^{\mathrm{CPU}} + T^{\mathrm{COM}} = n^{\mathrm{I/O}} \sum_{i=1}^{p/c} \frac{z + R_1(i, z, S_{\mathrm{queue}}^{\mathrm{COM}})}{i}, \quad (16)$$

where $R_1(i, z, x)$ is the response time of a closed queuing network with population i composed of a delay z and a queuing station with service time x. The response time $R_1(i, z, x)$ is given by

$$R_1(i, z, x) = x \frac{\sum_{(e_1,e_2)\in L_{i-1}^2} \frac{e_1+1}{e_2!} x^{e_1} z^{e_2}}{\sum_{(e_1,e_2)\in L_{i-1}^2} \frac{1}{e_2!} x^{e_1} z^{e_2}}, \quad (17)$$

where L_i^2 is the set of pairs (e_1, e_2) of nonnegative integers such that $e_1 + e_2 = i$. A proof of (16) and (17) can be found in [41] and [42], respectively.

The response time for the I/O burst is given by (14)

$$T^{\mathrm{I/O}} = S_0^{\mathrm{I/O}} + S_{\mathrm{R}}^{\mathrm{I/O}}/d. \quad (18)$$

Upon substituting (16) and (18) into (3), we can evaluate the speedup $s(p, d)$.

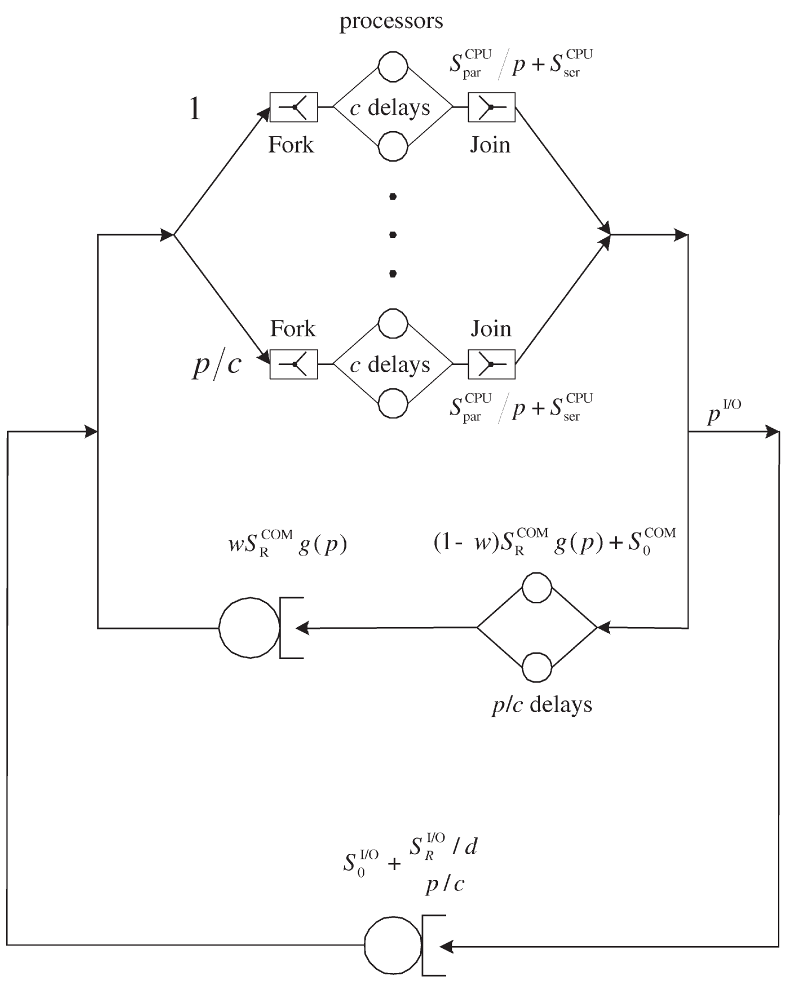
processors
$S_{\text{par}}^{\text{CPU}}/p+S_{\text{ser}}^{\text{CPU}}$
1
Fork
c delays
Join
Fork
Join
p/c
c delays
$S_{\text{par}}^{\text{CPU}}/p+S_{\text{ser}}^{\text{CPU}}$
$p^{\text{I/O}}$
$wS_{\text{R}}^{\text{COM}}g(p)$
$(1-w)S_{\text{R}}^{\text{COM}}g(p)+S_{0}^{\text{COM}}$
p/c delays
$S_{0}^{\text{I/O}}+\frac{S_{R}^{\text{I/O}}/d}{p/c}$

Fig. 8. BUS-AIO queuing network model.

4.2 BUS-AIO Mode

The BUS-AIO model represents centralized I/O architectures and parallel applications characterized by asynchronous I/O. Since in this case we do not have any I/O synchronization, we remove from the model the fork/join pair (Fig. 8). The BUS-AIO queuing network can be solved by means of exact MVA algorithm. The queuing network is composed of a delay station and two queuing stations. The service time z of the delay station is given by (15). The service time $S_{\text{queue}}^{\text{COM}}$ of the first queuing station is given by (9). The the second queuing station represents the I/O burst and its service time is given by (14)

$$T^{\text{I/O}}=S_{0}^{\text{I/O}}+\frac{S_{\text{R}}^{\text{I/O}}/d}{p/c}. \tag{19}$$

By using MVA, we have

$$T^{\text{CPU}}+T^{\text{COM}}=zn^{\text{I/O}}+R_2\Big(p,zn^{\text{I/O}},S_{\text{queue}}^{\text{COM}}n^{\text{I/O}},T^{\text{I/O}}\Big) \tag{20}$$

and

$$T^{\text{I/O}}=R_2\Big(p,zn^{\text{I/O}},T^{\text{I/O}},S_{\text{queue}}^{\text{COM}}n^{\text{I/O}}\Big), \tag{21}$$

where $R_2(i,z,x,y)$ is the response time of a closed queuing network with population i composed of a delay z and two queuing station with service time x and y

$$R_2(i,z,x,y)=x\frac{\sum_{(e_1,e_2,e_3)\in L^3_{p/c-1}}\frac{e_1+1}{e_3!}x^{e_1}y^{e_2}z^{e_3}}{\sum_{(e_1,e_2,e_3)\in L^3_{p/c-1}}\frac{1}{e_3!}x^{e_1}y^{e_2}z^{e_3}} \tag{22}$$

and L_p^3 is the set of triples (e_1,e_2,e_3) of nonnegative integers such that $e_1+e_2+e_3=p$. From (20) and (21), we can evaluate the speedup of the BUS-AIO model.

4.3 CLU-AIO Model

The model for the CLU-AIO is a multiclass closed queuing network. The queuing network contains d classes, each class representing the portion of a parallel application executed on one of the d clusters of processors that shares a common disk (see Fig. 9). As mentioned, we consider only the case $p/c=kd$. Because of the load-balance hypothesis stated in Section 3, the d components of the population vector $\vec{n}$ are identical

$$\vec{n}=\Big(\frac{p}{cd},\frac{p}{cd},\dots,\frac{p}{cd}\Big).$$

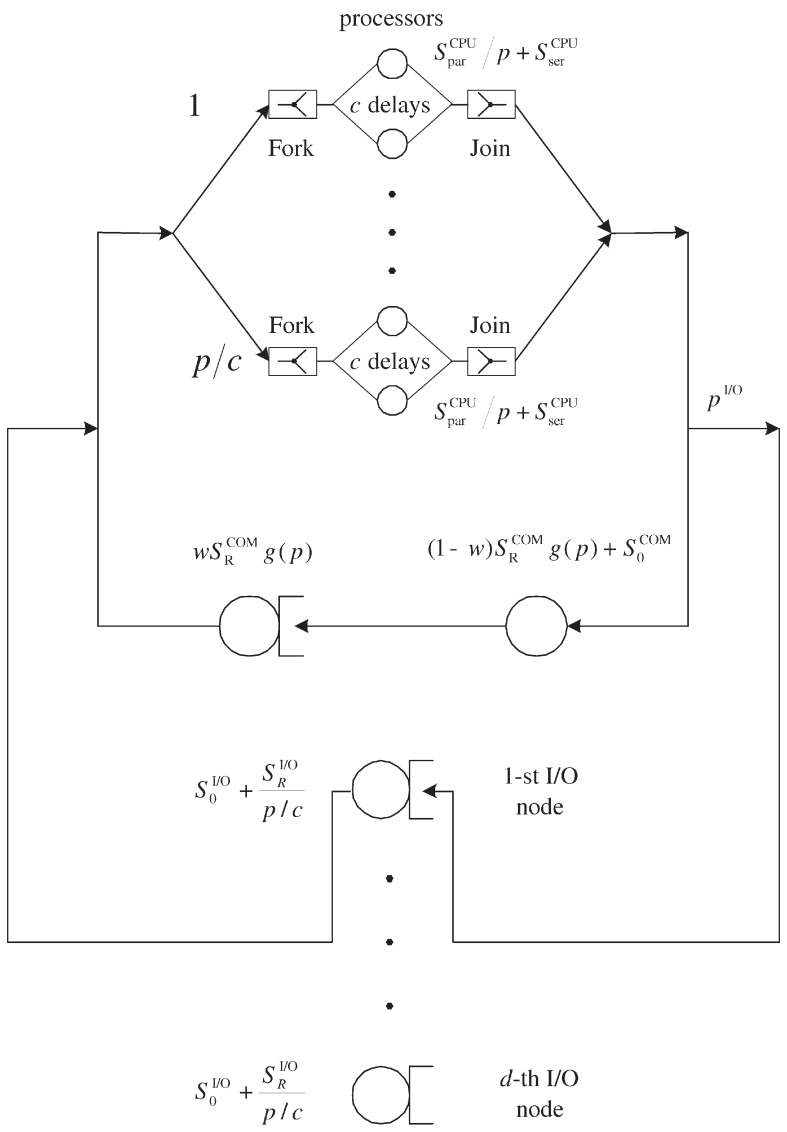
processors
$S_{par}^{CPU}/p+S_{ser}^{CPU}$
1
Fork
c delays
Join
p/c
Fork
c delays
Join
$S_{par}^{CPU}/p+S_{ser}^{CPU}$
$p^{I/O}$
$wS_R^{COM}g(p)$
$(1-w)S_R^{COM}g(p)+S_0^{COM}$
$S_0^{I/O}+\frac{S_R^{I/O}}{p/c}$
1-st I/O node
$S_0^{I/O}+\frac{S_R^{I/O}}{p/c}$
d-th I/O node

Fig. 9. CLU-AIO queuing network model.

The I/O subsystem is composed of d queuing stations operating in parallel, each station representing a disk. The service time of each disk station is given by (14)

$$T^{\mathrm{I/O}} = S_0^{\mathrm{I/O}} + \frac{S_{\mathrm{R}}^{\mathrm{I/O}}}{p/c}.$$

The service times of the queuing network are represented by a $(d+2)\times d$ matrix G

$$G = \begin{pmatrix} T^{\mathrm{I/O}} & 0 & \dots & 0 \\ 0 & T^{\mathrm{I/O}} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & T^{\mathrm{I/O}} \\ S_{\mathrm{queue}}^{\mathrm{COM}} n^{\mathrm{I/O}} & S_{\mathrm{queue}}^{\mathrm{COM}} n^{\mathrm{I/O}} & \dots & S_{\mathrm{queue}}^{\mathrm{COM}} n^{\mathrm{I/O}} \\ zn^{\mathrm{I/O}} & zn^{\mathrm{I/O}} & \dots & zn^{\mathrm{I/O}} \end{pmatrix},$$

where $S_{\mathrm{queue}}^{\mathrm{COM}}$ and z are defined in (9) and in (15), respectively. In order to solve the CLU-AIO queuing network, the MVA algorithm can be used.

5 Analysis of Real Applications

We now apply our model to analyze the behavior of real parallel applications, demonstrating that the methodology is a powerful abstraction to evaluate application scalability as a function of its resource requirements.

In Section 5.1, we apply the model to the performance analysis of BTIO, a parallel I/O intensive application from the NAS Parallel Benchmark suite (NPB). The target architecture is the IBM SP-2. We investigate the structure of the algorithm and the architecture of the parallel machine in order to obtain the model parameters. A comparison between the results of the analytic model and the measures obtained from the execution of the benchmark is presented.

In Section 5.2, we infer the inner characteristics of a parallel applications by fitting the analytic model to observed speedup surfaces. The application considered (QCRD) is selected among those of the Scalable I/O Initiative.

TABLE 1
Models Parameters Summary

Parameter	Meaning
p	number of processors
d	number of disks
r	number of dimensions of the parallel data structure
$S_{\text{par}}^{\text{CPU}}$	parallel computation time
$S_{\text{ser}}^{\text{CPU}}$	serial computation time
S_0^{COM}	communication startup time
$S_{\text{R}}^{\text{COM}}$	reciprocal of the communication transfer rate
$S_0^{\text{I/O}}$	I/O startup time
$S_{\text{R}}^{\text{I/O}}$	I/O transfer time
w	communication contention level (0 = delay, 1 = queue)
c	synchronization level
$g(p)$	communication scale function

5.1 The BTIO Application

In order to evaluate usefulness and accuracy of the model, we compare the model results with the performance measures of BTIO, a kernel benchmark from the NPB suite, as reported by Fineberg et al. in [43]. The NPB suite [44] consists of a set of seven kernels (BT, LU, FT, IS, EP, MG, and SP) extracted from typical computational fluid dynamics codes. Working together, IBM T.J. Watson Research Center and the Parallel Systems Group at the NAS facility (NASA Ames Research Center) have drafted MPI-IO, a proposal to address the portable parallel I/O problem [45]. BTIO is the NAS kernel benchmark for MPI-IO. The BTIO kernel extends the original BT kernel by using MPI-IO to write data to a file at regular time intervals [46].

The BTIO (Block Tridiagonal) pseudoapplication simulates high-speed compressible airflow by solving a system of partial differential equations using the ADI method. BTIO simulates airflow in time steps: Air is modeled in a 3D grid of (velocity, temperature, pressure) values. For each time step, airflow is approximated by traversing the grid in the X, Y, then Z dimension. For each traversal, all lines can be done in parallel. The parallel implementation adopts a multipartition scheme that requires the number of processors p to be a perfect square. The kernel executes 200 time steps and writes the solution matrix every five time steps.

The BTIO kernel comes in three possible sizes (class A, B, and C), each class with a different grid size. The analysis described in the following subsection refers to a problem size of class A ($64 \times 64 \times 64$ grid points).

5.1.1 Parameters Estimation

In order to calculate the parameters of the model (listed in listed in Table 1), we need to measure some performance features that characterize the IBM SP-2. We use three simple low-level kernel applications *POLY1*, *COMMS1*, and *COMMS3* from the Parkbench benchmark suite [47].

The *POLY1* kernel executes a polynomial evaluation and can be used to calculate the CPU floating point instructions rate. The CPU rate measured for the IBM SP-2 is $r^{\text{CPU}} = 120$ MFlop/s. The analysis of the BTIO algorithm shows that each time step requires approximatively 840 MFlop for a class A problem size (830 MFlop in the parallel fraction, 10 MFlop in the serial fraction). The S^{CPU} parameters can be computed as the ratio between the number of MFlop required by a CPU burst and the CPU performance rate r^{CPU}. It follows that the serial and parallel CPU time are $S_{\text{ser}}^{\text{CPU}} = 0.08$ s and $S_{\text{par}}^{\text{CPU}} = 6.9$ s, respectively.

The communication kernel *COMMS1* measures the basic communication properties of a parallel computer by ping-ponging a message of given length between two processors. The values obtained by COMMS1 are the startup time t_0 required to send a message and the communication transfer rate r_∞ without contention. The measured values for the IBM SP-2 are $t_0 = 150\mu s$ and $r_\infty = 27$ MByte/s [48]. The $S_{\text{R}}^{\text{COM}}$ parameter is the reciprocal of the communication transfer rate $S_{\text{R}}^{\text{COM}} = 1/r_\infty = 0,037\text{s/MByte}$. The S_0^{COM} parameter can be calculated as the product between the startup time t_0 and the average number of messages sent by one processor during a communication burst. The analysis of the BTIO algorithm shows that the number of messages transmitted at each communication burst by one processor is proportional to $p^{1/2}$, while the average message size is proportional to $p^{-2/3}$. The scale function $g(p)$ is equal to $p^{-1/6}$. The number of messages transmitted by each processor and their average message size are reported in

TABLE 2
BTIO Characteristics—The Values Refer to One Processor

Number of processors	MFlop 1 time step	Number of messages 1 time step	Message size 1 time step	I/O write size 5 time steps
9	95	18	64 KByte	10 MByte
64	13	48	18 KByte	10 MByte

TABLE 3
BTIO Execution Time, Five Time Steps

Number of processors	9	64
Real execution (s)	11.5	6.0
Model estimate (s)	13.1	6.2

Table 2. Since the BTIO algorithm uses asynchronous communications, we choose $c = 1$.

The communication contention level w is the most difficult parameter to estimate, because its value depends on both the interconnection network and the communications pattern. However, it is possible to compute an upper bound for w by means of the COMMS3 kernel. In the *COMMS3* kernel, each processor sends a message to all the other processors, then waits to receive all the messages directed at it. The value obtained by COMMS3 is the total saturation bandwidth r_{sat} and corresponds to the maximum throughput of the communication submodel $1/(wS_{\text{R}}^{\text{COM}})$ [49]. The saturation bandwidth measured for the IBM SP-2 is $r_{\text{sat}} = 120$ MByte/s [48], yielding to $w \simeq r_\infty / r_{\text{sat}} = 0.23$.

The BTIO implementation described in [43] was run on an IBM SP-2 with $d = 3$ I/O nodes, each node capable of 10 MByte/s throughput during writing operations. The amount of data written by the kernel every five time steps ($p_{I/O} = 1/5$) is 10 MByte, yielding to $S_{\text{R}}^{\text{I/O}} = 1$ s. During I/O bursts, BTIO uses a small number of large write operations, therefore we neglect the I/O startup time $S_0^{\text{I/O}} = 0$. Since the MPI-IO implementation described in [43] used synchronized I/O, we choose to adopt the SIO model.

The results of the model are presented in Table 3. The Table shows a comparison between the experimental execution time and the execution time obtained from the SIO model. Note that the model yields to an overestimation of the execution time with 9 processors. This happens because the assumption of exponentially distributed CPU time leads to pessimistic results.

TABLE 4
QCRD Stage 2: Model Parameters

Parameter	Value
w	0.19
$S_{\text{ser}}^{\text{CPU}} n^{\text{I/O}}/T_1$	~ 0
$S_{\text{par}}^{\text{CPU}} n^{\text{I/O}}/T_1$	0.71
$S_0^{\text{COM}} n^{\text{I/O}}/T_1$	0.049
$S_{\text{R}}^{\text{COM}} n^{\text{I/O}}/T_1$	0.41
$S_{\text{R}}^{\text{I/O}}/T_1$	0.001
$S_0^{\text{I/O}}/T_1$	~ 0

5.2 Speedup Surfaces of Applications with Intensive I/O

In this section, we illustrate how the modeling technique can be used to study the inner characteristics of real parallel applications. The main goal of the section is not to predict the performance of a parallel application, but rather to analyze its scalability in order to find its major bottlenecks.

The application considered is selected among those of the Scalable I/O Initiative and is described in [9]. The Scalable I/O Initiative [6] is an effort to collect a suite of I/O intensive challenge scientific applications in order to design and evaluate policies for the management of parallel file systems. The experimental platform used is the 512-processor Intel Paragon XP/S with 64 4GB Seagate disks attached to an I/O processor, at the Caltech Center of Advanced Computing Research. Performance measures are collected using Pablo [50], a performance analysis environment that provides trace data for the I/O and CPU requests of the parallel applications.

The application considered is QCRD (Quantum Chemical Reaction Dynamics) that solves the Schroedinger equation for the differential and integral cross section of the scattering of an atom by a diatomic molecule. QCRD implements the method of symmetrical hyperspherical coordinates and local hyperspherical surface functions with a typical SPMD structure. All processors execute the same

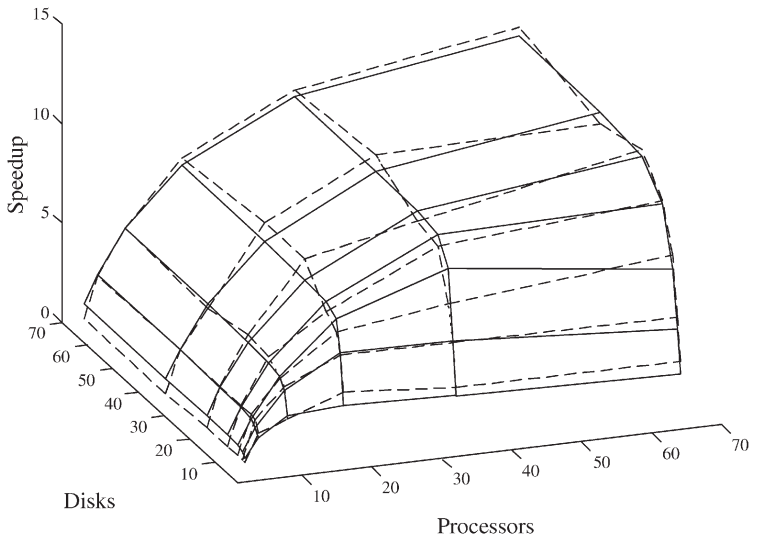

Fig. 10. QCRD stage 2—Experimental speedup surface (dashed line) versus analytical (solid line).

code on different portions of the data set each of equal size so as to keep the load balanced. The execution is divided into five consecutive stages that proceed in a pipeline fashion. The analysis focus on stage 2 because it achieves reasonable speedups. During stage 2, each processor independently computes a subset of the integrals that are needed to evaluate the two-dimensional quadratures involving the primitive basis functions. Both read and write operations are performed by all processors.

Since in the QCRD stage 2 the processors uses asynchronous communications and perform I/O operations independently, we use the BUS-AIO model with $c = 1$. The analysis of the data partition scheme adopted in the QCRD stage 2 yields to define the communication scale function as $g(p) = 1/p$.

Given a set of 42 observed speedup $\bar{s}(p, d)$, we estimate the values of the remaining seven model parameters by means of a least square fitting (the number of observations required must be greater or at least equal to the number of unknown parameters). The parameters are shown in Table 4. All the parameters but w are normalized with respect to the application execution time on one processor and one disk T_1.

The analysis of Table 4 shows that the application shall not be considered I/O intensive but rather CPU intensive because $S_{\mathrm{R}}^{\mathrm{I/O}}$ and $S_0^{\mathrm{I/O}}$ are negligible with respect to $S^{\mathrm{CPU}} n^{\mathrm{I/O}}$. Moreover, the main performance bottleneck lays in the communication phase. In fact $S_{\mathrm{R}}^{\mathrm{COM}} n^{\mathrm{I/O}}$ and w are relevant and their overhead effects increase with the number of processors.

Fig. 10 shows the speedup measured (dashed line) versus the speedup obtained from the model (solid line). We define the average error as

$$\frac{1}{N_P}\sqrt{\sum_{p,d}\frac{[s(p,d)-\bar{s}(p,d)]^2}{\bar{s}(p,d)^2}},$$

where N_P is the number of experimental points in the speedup surface. We observe that the fitted model is a good match for the observed data (the average error being 0.2 percent).

6 Conclusion

In this paper, we have described a family of queuing network models for the performance analysis of parallel programs when executed on different type of systems. The purpose of the models is to estimate the speedup of a parallel application through some high level parameters characterizing the application and the target architecture.

The underlying idea is that the performance of a parallel application strictly depends on the coupling between the application and the architecture characteristics.

From these models we obtain the qualitative and quantitative behavior of programs that alternate computations and I/O in a cyclic fashion. The models allow to study the impact of both communication and I/O contention, showing the dependence among speedup, number of processors and number of disks in the parallel machine. Various aspects of the communication and I/O have been analyzed and different hardware architectures have been taken into consideration.

Moreover, we have shown that we can infer the programming characteristics of an application by fitting the model to observed speedup. This can help programmers to analyze the scaling properties of an application with respect to: number of processors, number of disks, CPU performance, disk performance and problem size. The fitting of experimental speedup surfaces produced very small errors; thus it would be appropriate to use these estimated parameters for allocation and scheduling.

Future works are planned to extend the computation subsystem model in order to take into account the effects of deep load unbalance (by using multiclass queuing networks) and the effects of processor multiprogramming (by substituting the CPU delay centers with queue centers).

References

[1] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Ellis, and M. Best, "File-Access Characteristics of Parallel Scientific Workloads," Technical Report PCS-TR95-263, Mar. 1995.

[2] S. Baylor and C. Wu, *I/O, in Parallel and Distributed Computer Systems.* chapter 7, Kluwer Academic, 1996.

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

[4] B.K. Pasquale and G. Plyzos, "A Static Analysis of I/O Characterization of Scientific Applications in a Production Workload," *Proc. Conf. Supercomputing '93*, pp. 388-397, Nov. 1993.

[5] S. Kuo, M. Winslett, Y. Chen, Y. Cho, M. Subramaniam, and K. Seamons, "Application Experience with Parallel Input/Output: Panda and the H3expresso Black Hole Simulation on the SP2," *Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing,* 1997.

[6] J.T. Poole, "Scalable I/O Initiative," Available at http://www.ccsf.caltech.edu/SIO/. 1996.

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

[8] E. Rosti, G. Serazzi, E. Smirni, and M.S. Squillante, "Models of Parallel Applications with Large Computation and I/O Requirements," *IEEE Trans. Software Eng.*, vol. 28, no. 3, Mar. 2002.

[9] E. Rosti, G. Serazzi, E. Smirni, and M.S. Squillante, "The Impact of I/O on Program Behavior and Parallel Scheduling," *ACM Sigmetrics Conf.*, pp. 56-65, June 1998.

[10] C. Gennaro, "Performance Models for I/O Bound SPMD Applications on Clusters of Workstations," *Proc. Seventh Euromicro Workshop Parallel and Distributed Processing,* 1999.

[11] G.M. Amdhal, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proc. AFIPS 1967 Spring Joint Computer Conf.*, vol. 30, pp. 483-485, Apr. 1967.

[12] J.L. Gustafson, "Reevaluating Amdahl's Law," *Comm. ACM,* vol. 31, no. 5, pp. 532-533, 1988.

[13] J.L. Gustafson, "The Scaled-Sized Model: A Revision of Amdhal's Law," *ICS Supercomputing,* vol. II, pp. 130-133, 1988.

[14] J.L. Gustafson, G.R. Montry, and R.E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Scientific and Statisical Computing,* vol. 9, no. 4, pp. 609-638 1988

[15] H.P. Flatt and K. Kennedy, "Performance of Parallel Processors," *Parallel Computing,* vol. 12, pp. 1-20, 1989.

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

[17] X. Wu and W. Li, "Performance Models for Scalable Cluster Computing," *J. System Architecture* vol. 44, pp. 189-205, 1998.

[18] E.G. Coffman and P.J. Denning, *Operating System Theory.* Inglewood Cliffs, N.J.: Prentice-Hall, 1973.

[19] U. Herzog and W. Hoffmann, "Syncrhonization Problems in Hierachically Organized Multiprocessor Computer Systems," *Performance of Computer System, Proc. Fourth Int'l Symp. Modeling Performance Evaluation Computer Systems,* pp. 29-48, 1979.

[20] K.R. Backer, *Introduction to Sequencing and Software.* John Wiley & Sons, 1974.

[21] G. Fayolle, P.J.B. King, and I. Mitrani, "On the Execution of Programs by Many Processors," *Proc. Conf. Performance '88,* pp. 217-228, 1983.

[22] P. Mussi and J. T. Nain, "Evaluation of Parallel Execution of Program Tree Structures," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 78-87, 1984.

[23] T. Philips, E. Gelenbe, R. Nelson, and A. Tantawi, "The Asymptotic Processing Time for a Model of Parallel Computation," *Proc. Nat'l Computer Conf.,* 1986.

[24] E. Gelenbe, *Multiprocessor Performance.* pp. 83-90, John Wiley & Sons, 1989.

[25] J.C.S. Lui, R.R. Muntz, and D. Towsley, "Computing Performance Bounds of Fork-Join Parallel Programs under a Multiprogrammed Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, pp. 295-311, Mar. 1998.

[26] F. Baccelli and Z. Liu, "On the Execution of Parallel Programs on Multiprocessor System—A Queuing Theory Approach," *J. ACM* vol. 37, no. 2, pp. 373-414, 1990.

[27] S. Balsamo, Z. Liu, and N.M. Van Dijk, "Bound Performance Models of Heterogeneous Parallel Processing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 10, pp. 1041-1056, Oct. 1998.

[28] A.W. Apon and L.W. Dowdy, "The Circulating Processor Model of Parallel Systems," *IEEE Trans. Computers,* vol. 46, no. 5, pp. 572-587, May 1997.

[29] X. Qin and J.-L. Baer, "A Performance Evaluation of Cluster Architectures," *Proc. ACM SIGMETRICS '97,* 1997.

[30] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Survey,* vol. 26, no. 2, pp. 145-185, 1994.

[31] V. Catania, A. Puliafito, S. Riccobene, and L. Vita, "Design and Performance Anlaysis of a Disk Array System," *IEEE Trans. Computers,* vol. 44, no. 10, pp. 1236-1247, Oct. 1995.

[32] D. Kotz, "Disk-Directed I/O for MIMD Multiprocessors," *ACM Trans. Computer Systems,* vol. 15, no. 1, pp. 41-74, Feb. 1997.

[33] I. Foster, "Design and Building Parallel Programs," Available at www.mcs.anl.gov/dbpp/text/. 1995.

[34] Y. Chen, M. Winslett, K. Seamons, S. Kuo, Y. Cho, and M. Subramaniam, "Scalable Message Passing in Panda," *Proc. Fourth Ann. Workshop I/O Parallel and Distributed Systems,* May 1996.

[35] P. Messina, "The Concurrent Supercomputing Consortium: Year One," *IEEE Parallel and Distributed Technology,* vol. 1, no. 1, pp. 9-16, 1993.

[36] C.E. Leiserson, "The Network Architecture of the Connection Machine CM-5," *Proc. Fourth Symp. Parallel Algorithms and Architectures,* pp. 272-285, June 1992.

[37] "Scalable Powerparallel Systems High-Performance Technical Computing Solutions," Technical Report GH23-2485-00, IBM, Mar. 1994.

[38] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. 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.

[39] K.S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications.* Durham, North Carolina: Prentice-Hall, 1982.

[40] B.A. Mahafzah and W.E. Cohen, "Verification on the Burst Send Queuing System Model for Parallel Programs," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications '99,* 1999.

[41] E. Varki, "Mean Value Technique for Closed Fork-Join Networks," *Proc. ACM SIGMETRICS '99,* 1999.

[42] P.J. Schweitzer, "Exact Solution of the MVA Equations," *SIAM Rev.,* vol. 23, pp. 528-532, 1981.

[43] S. Fineberg, P. Wong, B. Nitzberg, and C. Kuszmaul, "PMPIO—A Portable Implementation of MPI-IO," *Proc. Sixth Symp. Frontiers of Massively Parallel Computation,* pp. 188-195, Oct. 1996.

[44] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," Technical Report NAS-95-020, NAS, Available at http://www.nas.nasa.gov/Research/Reports/Techreports/1995/. 1995.

[45] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J. Prost, M. Snir, B. Traversat, and P. Wong, "Overview of the MPI-IO Parallel I/O Interface," *Proc. Third Workshop I/O in Parallel and Distributed Systems (IPPS'95),* Apr. 1995.

[46] R. Bagrodia, S. Docy, and A. Kahn, "Parallel Simulation of Parallel File Systems and I/O Programs," *Proc. Conf. Supercomputing '97,* pp. 15-21, Nov. 1997.

[47] R. Hockney and M. Berry, "Public International Benchmarks for Parallel Computers: PARKBENCH Committee," Technical Report Report-1, PARKBENCH Committee, Available at http://www.netlib.org/parkbench/. Feb. 1994.

[48] G.R. Luecke, B. Raffin, and J.J. Coyle, "Comparing the Communication Performance and Scalability of a Linux and a NT Cluster of PCs, a Cray Oorigin 2000, an IBM SP, and a Cray T3E-600," *Proc. First IEEE CS Int'l Workshop Cluster Computing,* pp. 26-35, 1999.

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

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

Paolo Cremonesi received the MSc degree in aerospace engineering in 1992 and the PhD degree in computer science in 1996, both from the Politecnico di Milano, Milan, Italy. He is currently an assistant professor of computer science with the Politecnico di Milano. His research interests include high-performance computing modeling and other topics related to the performance evaluation of computer systems and networks.

Claudio Gennaro received the MSc degree in electronic engineering from University of Pisa in 1994 and the PhD degree in computer science from Politecnico di Milano in 1999. He is now a researcher with the National Research Council (CNR), Pisa, Italy. His current main research interests are performance evaluation of computer systems and parallel applications, similarity retrieval, and storage structures for multimedia information retrieval and multimedia document modeling.