ANALYTIC MODELS OF SUPERCOMPUTER PERFORMANCE IN MULTIPROGRAMMING ENVIRONMENTS

Daniel A. Menascé

DEPARTAMENTO DE INFORMÁTICA
PONTIFICIA UNIVERSIDADE CATÓLICA
22453 RIO DE JANEIRO, BRAZIL

Virgilio A. F. Almeida

DEPARTAMENTO DE CIENCIA DA COMPUTAÇÃO
UNIVERSIDADE FEDERAL DE MINAS GERAIS
30161 BELO HORIZONTE, BRAZIL

Summary

Supercomputers run multiprogrammed time-sharing operating systems, so their facilities can be shared by many local and remote users. Therefore, it is important to be able to assess the performance of supercomputers in multiprogrammed environments. Analytic models based on Queueing Networks (QNs) and Stochastic Petri Nets (SPNs) are used in this paper with two purposes: to evaluate the performance of supercomputers in multiprogrammed environments, and to compare, performance-wise, conventional supercomputer architectures with a novel architecture proposed here. It is shown, with the aid of the analytic models, that the proposed architecture is preferable performance-wise over the existing conventional supercomputer architectures. A three-level workload characterization model for supercomputers is presented. Input data for the numerical examples discussed here are extracted from the well-known Los Alamos benchmark, and the results are validated by simulation.

The International Journal of Supercomputer Applications, Volume 3, No. 2, Summer 1989, pp. 71–91.

Introduction

Vector computers are being widely used for applications that require high-speed computing, such as weather forecasting, spaceship and aircraft design and simulation, and analysis of geological and seismic data. These machines are also called supercomputers because they are the fastest machines of their times.

Supercomputers are very expensive machines, and they run multiprogrammed time-sharing operating systems (e.g., UNICOS) (Cray, 1988), so that their facilities can be shared by many local and remote users. Therefore, it is important to be able to assess the performance of supercomputers in multiprogrammed environments. Most studies of supercomputer performance are concerned with the evaluation of the effective speed of a program running in isolation on a particular supercomputer. The effective speed of the machine running a specific program results from the combination of different speeds, such as the sequential speed, the vector or synchronous speed, and the parallel or asynchronous speed. These three factors may be combined by a relation which is an extension of Amdahl's law (Amdahl, 1967). Several studies based on actual measurements of benchmarks have analyzed the effective speed of vector computers in uniprogramming environments (Bucher, 1983; Bucher and Simmons, 1985; Lubeck, Moore, and Mendez, 1985; Dongarra, Martin, and Worlton, 1987).

Analytic models based on Queueing Networks (QNs) and Stochastic Petri Nets (SPNs) are used in this paper with two purposes. The first is to evaluate the performance of supercomputers in multiprogrammed environments, and the second is to compare, performance-wise, conventional supercomputer architectures with a novel architecture proposed here. It is shown here, with the aid of the analytic models, that the proposed architecture is preferable performance-wise over the conventional architectures.

Queueing network models having product form solutions, which are amenable to efficient and general solution techniques, cannot represent directly the performance of vector and parallel computers (Almeida and Dowdy, 1986; Lazowska et al., 1984). The reason stems from the concurrency that exists between processors working on the same job. In order to model this concurrency, an SPN model (Molloy, 1981; Marsan, Balbo, and Conte, 1986) of a job executing in isolation is used at the lower level. An upper level model, that is, the QN

"Analytic models based on Queueing Networks and Stochastic Petri Nets are used in this paper to evaluate the performance of supercomputers in multiprogrammed environments and to compare conventional supercomputer architectures to a novel architecture proposed here."

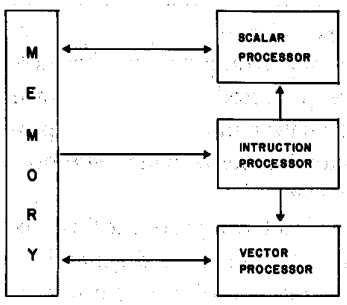


Fig. 1 Organization of a conventional supercomputer

model, is used to represent the multiprogramming environment. The combination of both modeling techniques leads to a new supercomputer performance model.

This paper is organized as follows. Section 1 presents a brief discussion on supercomputer architectures. Section 2 introduces a workload characterization model for supercomputers. Analytic models to analyze and compare supercomputer architectures are presented in section 3. Numerical results are then presented and discussed in section 4. Finally, section 5 presents some concluding remarks.

1. SUPERCOMPUTER ARCHITECTURES

Vector computer architectures are characterized by CPUs composed of three different types of processors: (1) the instruction processor (IP) is the unit that fetches, decodes, prepares, and executes some special instructions; (2) the scalar processor (SP) is the unit that executes scalar instructions; and (3) the vector processor (VP) is the unit that executes vector instructions.

A vector computer may have several scalar functional units and several vector functional units capable of independent parallel operation. The CRAY X-MP computer is an example of this type of architecture (Cray, 1982). See Figure 1 for a schematic view of the architecture of a conventional supercomputer.

The operation of this type of architecture may be described as follows. The IP fetches and decodes an instruction. If it is a scalar instruction and if there is a free scalar functional unit, the scalar instruction is issued to the SP for execution. If the SP is busy, the IP stays idle until the SP becomes available. If the instruction is of the vector type and there is a vector functional unit available, the instruction is issued to the VP for execution. Otherwise, the IP stays idle until the VP becomes available.

There are basically two types of architectures of vector processing units: those which, like the ETA-10 (ETA, 1986), reference memory directly in their vector instructions, and those which, like the CRAY X-MP (Cray, 1982), require that the array be loaded piece-wise into vector registers before the execution of the operation can start. The first type of architecture will be referred to hereafter as M-M (for memory-to-memory)

computers and the second type as R-R (for register-to-register) computers. A more detailed description of the operation of supercomputers can be found elsewhere (Hwang and Briggs, 1987; Ercegovac and Lang, 1986; Weiss and Smith, 1984).

In a vector architecture, a vector instruction can initiate a very long vector operation. In the case of R-R architectures, when the vector length exceeds the number of elements that can be stored in a vector register, the vector operation may have to be broken up into a sequence of vector instructions. The execution time of a vector instruction is a function of the number of elements of the array (vector length) to be operated by the instruction, and of the time required to fill the pipe before starting the pipelined execution of the vector instruction.

A vector operation on a vector of length 1,000 may take roughly 12 microseconds on a CRAY X-MP computer (Bucher and Simmons, 1985). Since these times are orders of magnitude greater than those for scalar instruction execution, it may be advantageous to modify the architecture described above in the following manner:

1. If a vector instruction is decoded, prepared, and ready to be issued to the vector processor, and if there is at least one functional unit available in the VP, the instruction is executed while the issuing task continues its processing at the CPU.
2. If a vector instruction is decoded, prepared, and ready to be issued to the vector processor, and if there is no vector functional unit available on the VP, the following must occur: the vector instruction is placed on an execution queue of vector instructions for the VP; the current task execution is suspended and another task is dispatched by the operating system.
3. When the VP completes the execution of a vector instruction which had been started in an independent way (i.e., which does not belong to the task in execution), the VP generates an interrupt to the CPU so that the operating system may place the task whose vector instruction has just completed in the ready queue for the CPU.

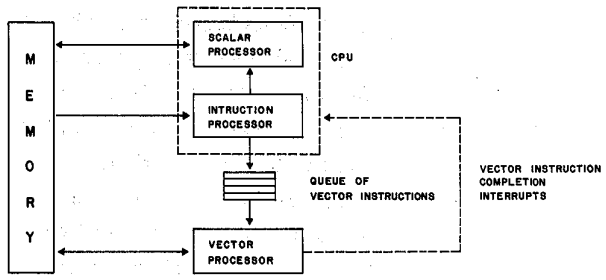


Fig. 2 Proposed architecture for supercomputers

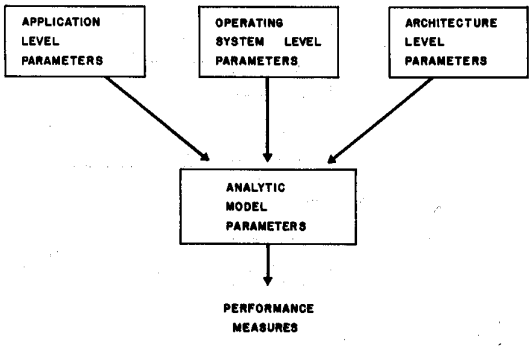


Fig. 3 Workload characterization approach

The architecture described above considers the VP very much like a peripheral unit of the CPU. The motivation for it stems from the potentially large execution times for vector instructions compared to those of scalar instructions, and from the fact that in a multiprogrammed environment more parallelism may be achieved if the VP is allowed to execute vector instructions for a task other than the one that is in hold of the CPU. Figure 2 depicts the proposed architecture.

From this point on we will refer to the conventional architecture as C-architecture and to the proposed architecture as P-architecture.

2. WORKLOAD CHARACTERIZATION MODEL OF SUPERCOMPUTERS

An interesting report on workload characterization for vector computers was carried out by Martin, Bucher, and Warnock (1983) at the Los Alamos National Laboratory. This study used a benchmark for supercomputers, known as the Los Alamos benchmark, which is a set of scientific programs that run at that laboratory (Griffin and Simmons, 1983).

In this section we will take a slightly different approach toward workload characterization of supercomputers, since we are interested in using measured data as input for analytic models. Our approach for workload characterization considers three levels of parameters: application, operating system, and architecture level, as indicated by Figure 3. The parameters at these three levels will then be mapped into the analytic model parameters as discussed in section 3.

2.1 APPLICATION-LEVEL PARAMETERS

Consider the following parameters at this level:

p_S: percentage of scalar code, i.e., fraction of the total executed instructions which are executed at the SP.

p_V: percentage of vector code, i.e., fraction of the total executed instructions which are executed at the VP. (A vector instruction is counted here as one instruction, independently of the number of operations performed by it. In order to obtain the number of elements that have been operated in vector mode, one should multiply the vector instruction count by the average vector length.)

p_I: percentage of code which is executed exclusively by the IP (e.g., jumps, address computations, register transfers).

An obvious relationship between the above parameters is

$$p_S + p_V + p_I = 1. \quad (1)$$

The remaining parameters at this level are

v: average vector length
ic: instruction count, i.e., number of executed instructions.

It should be noted that parameters such as p_S and p_V are not exclusively application dependent. For instance, different compilers may generate different levels of vectorization for a given application (Tang and Davidson, 1988). A question that arises is whether these data may be easily obtained in practice. The answer is affirmative as can be deduced from the studies of Martin, Bucher, and Warnock (1983) and Bucher and Simmons (1985), which show tables containing the above parameters directly or other data from which the necessary parameters may be easily derived. For instance, the average vector length, v, for each code in the Los Alamos benchmark is given by Martin, Bucher, and Warnock (1983), Bucher (1983), and Lubeck, Moore, and Mendez (1985). Also, Table III of Martin, Bucher, and Warnock (1983) contains the instruction count, ic, for each benchmark code of the same benchmark. Besides the total instruction count, the same table displays the instruction count per instruction class. These figures allow us to easily derive the percentage of scalar and vector code, p_S and p_V. Finally, p_I may be derived from Eq. (1) above.

2.2 OPERATING SYSTEM PARAMETERS

The relevant parameters at this level are:

R: number of different types or classes of workloads. Different workload classes may differ in the type of demand they place on the several resources of the computer system.
N_r: maximum multiprogramming level for class r ($1 \leq r \leq R$).

sw: time necessary to switch the context between two tasks. This parameter is a function of the number of load and store instructions necessary to save the context of the suspended task and to install the context of a new task. Some computers, such as the ETA-10 Model G (ETA, 1987), are capable of saving the whole register file with a single instruction, the SWAP instruction, which takes 32 cycles (7 nanoseconds each) to execute. By making appropriate hardware modifications, the number of cycles could be significantly reduced, making this process even faster (Arvind and Iannucci, 1987).

2.3 ARCHITECTURE-LEVEL PARAMETERS

Consider the following parameters:

c_{ip}: instruction processor cycle time (considered the same for the scalar processor).

nc_{sp}: average number of cycles per scalar instruction. In a pipelined scalar processor, this value should reflect the average number of cycles required to produce a scalar result. Knowledge of the instruction repertoire and traces of the workload execution may be used to obtain this value.

nc_{ip}: average number of cycles per instruction executed at the IP.

ncp_{ip}: average number of cycles to prepare an instruction.

nf_s: average number of scalar functional units in use. This number should reflect the average scalar parallelism that can be exploited by a given workload on a specific hardware.

nf_v: average number of vector functional units in use. This number should reflect the average vector parallelism that can be exploited by a given workload on a specific architecture. Some characteristics of the architecture, such as chaining, may influence the value of this parameter. For instance, the CRAY X-MP series (Cray, 1982) introduces flexible chaining of vector instructions. According to Tang and Davidson (1988), this feature increases the level of functional unit concurrency, which may be directly represented by an appropriate setting of the value of nf_v.

$\phi_T(v)$: function that determines the average execution

time of a vector operation on vectors of average length v for architectures of type T (T = M-M or R-R).

So, according to Bucher (1983),

$$\phi_{M\text{-}M}(v) = T_{start} + v * T_{elem}, \quad (2)$$

where

T_{start} = start-up time for the vector operation, and
T_{elem} = time per result element in a pipelined execution mode.

and

$$\phi_{R\text{-}R}(v) = T_{start} + v * (T_{startstrip}/V_{max} + T_{elem}), \quad (3)$$

where T_{start} and T_{elem} are as defined above and V_{max} = number of elements of the vector register, $T_{startstrip}$ = time to initiate a new suboperation, which is smaller than T_{start} because part of it can be overlapped with the previous suboperation.

For instance, Bucher (1983) shows that for the Cyber 205 supercomputer (which is of the M-M type), the following relationship holds

$$\phi_{Cyber\text{-}205}(v) = 1{,}000 + 10 * v, \quad (4)$$

where the constants in the above equation are given in nanoseconds.

3. ANALYTIC MODEL OF SUPERCOMPUTERS

In order to evaluate and compare the C-architecture and the P-architecture we are going to use a two-level modeling approach (Menascé and Nakanishi, 1981) indicated by Figure 4.

A Queueing Network (QN) model is used to obtain the desired performance measures—namely, average response time and throughput in a multiprogrammed environment. QN models require as input parameters the set of average service demands for each server and each class (see Appendix A). So, let

$D_{i,r}$ = average service demand of class r tasks at server i.

In other words, $D_{i,r}$ is the average total time spent by a class r task at device i while being served at the device.

Notice that the queueing time is not considered in $D_{i,r}$ but is computed when the QN model is solved, using

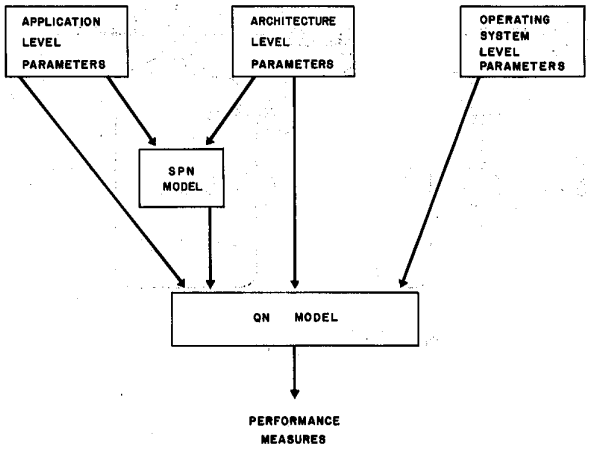


Fig. 4 Two-level modeling approach

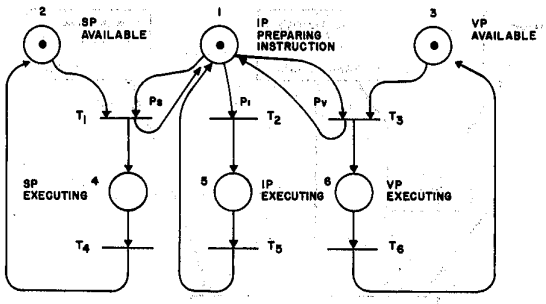


Fig. 5 SPN for the C-architecture

the standard mean value analysis technique (see Appendix A).

A continuous-time Stochastic Petri Net (SPN) model is used to derive the service demand at the CPU. An SPN model is necessary here in order to reflect the parallelism between the various processors (IP, SP, and VP) at the CPU. The following sections discuss the analytic model used to evaluate both architectures.

3.1 ANALYTIC MODEL FOR THE C-ARCHITECTURE

Consider the SPN shown in Figure 5, which represents the CPU composed of the IP, SP, and VP. The following meanings are associated with the various places of the above SPN when there is a token in the place:

Place 1: IP is preparing an instruction.
Place 2: SP is available.
Place 3: VP is available.
Place 4: SP is busy executing instructions in all its functional units.
Place 5: IP is executing an instruction.
Place 6: VP is busy executing instructions in all its functional units.

The firing time of transitions T_1, T_2, and T_3 represent the time needed to fetch, decode, and prepare an instruction, regardless of its type. The expected firing time of T_1, T_2, and T_3 is equal to

$$F_p = c_{ip} * ncp_{ip} \tag{5}$$

The firing time of transitions T_4, T_5, and T_6 represent the execution time of a scalar instruction, of an IP instruction and of a vector instruction, respectively. Their expected firing times are given by

$$F_{T4} = c_{ip} * nc_{sp} / nf_S \tag{6}$$

$$F_{T5} = c_{ip} * nc_{ip} \tag{7}$$

$$F_{T6} = \phi(v) / nf_V \tag{8}$$

The factors nf_s and nf_v which appear in Eqs. (6) and (8) account for the speedup that can be obtained due to the existence of multiple functional units within the scalar and vector processor, respectively.

The solution of an SPN is the set of steady state probabilities of all possible markings of its reachability

set (Peterson, 1981). These probabilities may be obtained by solving the Markov chain equivalent to the SPN. Appendix B presents the Markov chain for the SPN of Figure 5. The solution to it for each set of parameters may be obtained numerically using the Gauss elimination method.

Given the solution of the SPN, one is able to compute the service demand of a task at the CPU. This procedure will now be explained with the aid of Figure 6, which illustrates three time axes, one for each of the three processors (IP, SP, and VP). Consider the following sequence of instructions:

$$S_1, S_2, V_1, V_2, S_3, I_1, V_3, S_4, I_2, I_3,$$

where S_i denotes the ith scalar instruction of a task, V_i the ith vector instruction of a task, and I_i its ith IP instruction. As it can be seen, the IP time axis shows sequences of intervals of the following types:

i. preparation of scalar instructions
ii. preparation of vector instructions
iii. preparation of IP instructions
iv. execution of IP instructions
v. idle periods of type A
vi. idle periods of type B

An IP idle period of type A occurs when a scalar instruction is ready to be issued but the SP is busy. Similarly, a type B idle period occurs when a vector instruction is ready to be issued but the VP is busy.

Therefore, the service demand at the CPU, D_{CPU}, is given by the sum of the lengths of the following intervals: total time to prepare all scalar instructions, total time to prepare all vector instructions, total time to prepare and execute all IP instructions, total duration of all type A intervals, and total duration of all type B intervals.

Thus,

$$\begin{aligned} D_{CPU} &= ic * p_S * c_{ip} * ncp_{ip} + ic * p_V * c_{ip} * ncp_{ip} + ic * p_I * c_{ip} \\ &\quad * ncp_{ip} + ic * p_I * c_{ip} * nc_{ip} + ic * p_S * p_A * F_{T4} + ic \\ &\quad * p_V * p_B * F_{T6} \\ &= ic * (c_{ip} * ncp_{ip} + p_I * c_{ip} * nc_{ip} + p_S * p_A * F_{T4} \\ &\quad + p_V * p_B * F_{T6}), \end{aligned} \tag{9}$$

where P_A is the probability that a type A idle period occurs when a scalar instruction is to be issued. This is

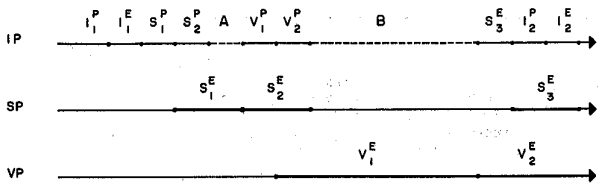


Fig. 6 Execution sequence at the IP, SP, and VP

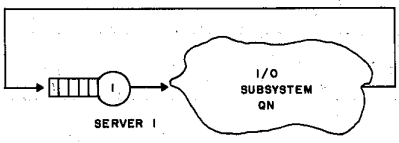


Fig. 7 QN model for the C-architecture

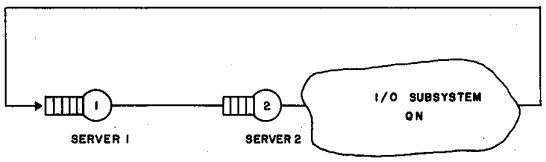


Fig. 8 QN model for the P-architecture

simply the sum of the probabilities (Pr) of two markings (see Appendix B) in the SPN as indicated below

$$p_A = Pr([1, 0, 1, 1, 0, 0]) + Pr([1, 0, 0, 1, 0, 1]). \quad (10)$$

Similarly, p_B is the probability that a type B idle period occurs when a vector instruction is to be issued. Thus,

$$p_B = Pr([1, 1, 0, 0, 0, 1]) + Pr([1, 0, 0, 1, 0, 1]). \quad (11)$$

The queueing network that represents the C-architecture is shown in Figure 7. Server 1 represents the CPU whose service demand for class r, $D_{CPU,r}$, is obtained from expression (9). The set of application-level parameters may be different for each workload class r, while the architecture-level parameters are the same for all classes. Numerical results obtained with this model are given in section 4.

3.2 ANALYTIC MODEL FOR THE P-ARCHITECTURE

The SPN model for this architecture is identical to that of Figure 5. The equivalence of the SPN model for both architectures stems from the fact that the C-architecture and the P-architecture have the same behavior when the multiprogramming level is equal to one. The difference between the architectures is represented by the queueing network model, which models the multiprogramming effects.

Before we indicate how to obtain the service demand at the CPU it is important to explain how the CPU is going to be modeled in this type of architecture. The QN model for the P-architecture is shown in Figure 8. Server 1 accounts for the time spent by a task at the CPU while using the IP or SP, or using the VP in an overlapped fashion with the other two processors. Also, the service demand of server 1 includes the additional time (CS) spent by a task in context switching due to VP unavailability. The service demand of server 1 is given by the expression (12) below.

$$D_1^p = D_1 + CS, \quad (12)$$

where D_1 and CS are given by expressions (13) and (14), respectively.

$$D_1 = ic * (c_{ip} * ncp_{ip} + p_I * c_{ip} * nc_{ip} + p_S * p_A * F_{T4}), \quad (13)$$

where p_A is defined in (10).

$$CS = ic * p_V * p_B * sw. \quad (14)$$

Expression (13) is derived using an argument similar to the one used for the previous case, taking into account the fact that in this case the IP never becomes idle due to the unavailability of the VP, since task execution is interrupted in that event. Recall that if the VP is available when a vector instruction has to be issued, the task in execution is not interrupted. Notice that in this case, type B intervals will not occur; therefore expression (13) correctly represents the time spent at the CPU, since this expression is a particular case of expression (9) with p_B set to zero.

Finally, the total time spent by a task executing vector instructions in a nonoverlapped manner with the execution of other (scalar or IP) instructions of the same task is represented by server 2. The service demand D_2 at this server is

$$D_2 = ic * p_V * p_B * F_{T_6} \quad (15)$$

Notice that the service demand at the CPU for the C-architecture given by expression (9) is the sum of D_1 and D_2, given by expressions (13) and (15).

4. NUMERICAL RESULTS

The input/output portion of the computer system was disregarded in the numerical studies conducted for this paper, since the architectures differ only in their CPU organization. However, the inclusion of I/O devices, if desired, may be easily considered in the manner usually done in QN models of conventional computer architectures.

In order to render our conclusions more realistic we used, at the application level, parameters derived from published measurements of well-known benchmarks, such as the Los Alamos benchmark, Livermore loops, and Linpack benchmark (Martin, Bucher, and Warnock, 1983; Bucher, 1983; Lubeck, Moore, and Mendez, 1985). Without loss of generality, we chose the Los Alamos benchmark data in order to obtain numerical results shown in this section. For our numerical example, we use parameters chosen from a "generic" vector supercomputer, roughly a composite of a number of current supercomputers.

Table 1
Architecture-Level Parameters (R-R)

Parameter	Parameter Value
c_{ip}	9.5 nsec
nc_{sp}	9
ncp_{ip}	1
nc_{ip}	1
T_{start}	798 nsec
$T_{startstrip}$	358 nsec
T_{elem}	9.5 nsec
V_{max}	64

Table 2
Application-Level Parameters

Workload	p_v	p_s	ic ($\times 10^6$)	v	R_v
BMK1	0.013	0.177	1235.39	61	.81
BMK4A	0.1905	0.189	143.89	7	.88
BMK11A	0.011	0.702	292.28	64	.50
BMK11B	0.021	0.774	199.12	64	.63
BMK11C	0.108	0.343	100.42	64	.95
BMK14	0.052	0.291	52.46	49	.90
BMK21A	0.0092	0.576	136.04	35	.36
BMK24A	0.0459	0.349	66.53	31	.75
BMK24B	0.033	0.362	246.24	63	.84
BMK24C	0.039	0.357	555.84	47	.84

Table 1 indicates the values considered for the architecture-level parameters in the case of R-R-type architectures.

The application-level parameters are indicated in Table 2. The identification of the workloads is the one used in the Los Alamos benchmark. The meaning of the rightmost column will be discussed shortly.

Note that the average vector length is equal to or less than V_{max} ($V_{max} = 64$). So for those specific workloads, a vector operation coincides with a vector instruction. The different codes of the Los Alamos benchmark can be classified according to the ratio, R_V, of arithmetic operations executed in vector mode to the total number of arithmetic operations executed by the program. An estimate for this ratio is given by the expression below:

$$R_V = (p_V * v) / (p_S + p_V * v) \qquad (16)$$

Codes for which this ratio is close to one are called vector-bound applications, those for which this ratio is close to zero are called scalar-bound applications, and those for which this ratio is close to 0.5 are called balanced applications. From Table 2 one can see that codes BMK11C, BMK14, BMK4A, BMK24B, BMK24C, BMK1, and BMK24A are vector-bound applications; codes BMK11A and BMK11B are balanced; and code BMK21A is scalar-bound. The context switching time, *sw*, used in all examples is equal to 50 nsec, except for Figure 10, where the influence of *sw* is studied. It should be noted that this value (50 nsec) is considerably lower than those of commercially available current supercomputers (ETA, 1987; Cray, 1982). Forthcoming architectures may be expected to exhibit reduced context switching times (Arvind and Iannucci, 1987).

From the input parameters given in Tables 1 and 2 one may solve the SPN and calculate the service demands D_1 and D_2 for the P-architecture according to expressions (12) through (15). Recall that the service demand for the C-architecture is equal to D_1 plus D_2, as indicated in expression (9). Table 3 shows the values of D_1, D_2, and CS in seconds, obtained by solving the SPN.

Experimentation with an event-driven simulation program has provided validation of our analytic models in multiprogramming environments. Table 4 presents a small sample of the results of our simulations and the corresponding results obtained with the analytic models.

Several independent runs of the simulator were made to produce 93% confidence intervals. The close correspondence between the analytic and simulation results validates our model.

Let us first consider the case of a single class model and carry out an asymptotic analysis of the response time. From the mean value analysis formulae (see Appendix A) it is a trivial matter to verify that for a single class single server QN, the average response time $R(N)$ is simply $N*D$ where N is the multiprogramming level and D the average service demand of the single server. Thus, since the C-architecture is modeled at the QN level as a single server (if we disregard the I/O subsystem), the average response time for this architecture, $R_C(N)$, is given by the following expression:

$$R_C(N) = N * (D_1 + D_2). \quad (17)$$

From Lazowska et al. (1984), we know that the average response time $R(N)$ of a closed queueing network with multiprogramming level equal to N has upper and lower bounds given by

$$\max(\Sigma D, N * D_{max}) \leq R(N) \leq N * D, \quad (18)$$

where ΣD is the sum of the service demands of all servers and D_{max} is the largest service demand at any single server. So, for sufficiently large values of N, $N * D_{max} > \Sigma D$. Hence, for the P-architecture we have the following bounds for the average response time $R_P(N)$

$$N * D_{max} \leq R_P(N) \leq N * (D_1 + D_2 + CS). \quad (19)$$

If the term CS (the context switching service demand) is small when compared with $(D_1 + D_2)$, it can be seen from expressions (17) and (19) that the response time for the C-architecture is approximately equal to the upper bound on the average response time for the P-architecture. In other words, the proposed architecture is always preferable, as far as response time is concerned, over the conventional architecture, provided that appropriate care is taken to minimize the impact of context switching. Let us now try to assess, with the help of the bounds defined above, the actual improvement of the P-architecture over the C-architecture. Consider then, the following definition for the percentage of asymptotic response time improvement (PARTI):

Table 3
Resource Demands for the Los Alamos Benchmark

Workload	D_1 (sec)	CS (sec)	D_2 (sec)
BMK1	24.1	0.40	4.472
BMK4A	2.633	0.784	5.465
BMK11A	8.61	0.122	1.368
BMK11B	6.372	0.17	1.9
BMK11C	1.477	0.336	3.797
BMK14	1.078	0.078	0.789
BMK21A	3.538	0.0425	0.384
BMK24A	1.42	0.09	0.788
BMK24B	5.34	0.24	2.7
BMK24C	12.0	0.647	6.41

Table 4
Response Time of the P-Architecture

N	Analytic	Simulation	Confidence Interval	Error (%)
4	6.35	6.74	6.36 : 7.11	5.7
5	7.69	8.32	7.69 : 8.95	7.4
10	15.31	16.10	16.0 : 16.3	4.9

$$\text{PARTI} = \lim_{N\to\infty} \frac{R_c(N) - R_p(N)}{R_p(N)} * 100. \qquad (20)$$

It turns out that the lower bound on response time for closed QNs is the asymptotic value for the response time (Lazowska et al., 1984). Thus, from (17), (19), and (20) we have

$$\text{PARTI} = \lim_{N\to\infty} \frac{N*(D_1 + D_2) - N*D_{\max}}{N*D_{\max}} \qquad (21)$$

$$= \frac{(D_1 + D_2) - D_{\max}}{D_{\max}}$$

The values of PARTI for all codes of the Los Alamos benchmark are given in Table 5. As can be observed, in some cases the improvement is remarkable, as is the case with code BMK14. The smallest observed improvement was 10.8%, and the largest was 73%. The improvement in response time as a function of the multiprogramming level is depicted graphically for workloads BMK14, BMK4a, and BMK11c in Figure 9, for a context switching time equal to 50 nsec.

For the sake of completeness, the percentage asymptotic response time improvement (PARTI) is now calculated for a "generic" M-M machine. The values considered for the architectural-level parameters (Bucher, 1983) in this case are c_{ip} = 20 nsec, nc_{sp} = 9, ncp_{ip} = 1, nc_{ip} = 1, T_{start} = 1020 nsec, and T_{elem} = 10 nsec. The application-level parameters, except the average vector length, are those defined by workload BMK11B in Table 2. In order to take advantage of the characteristics of an M-M type architecture, we consider that application BMK11B processes long vectors. For v = 512, the PARTI equals 62.8% (D_1 = 13.41 and D_2 = 8.42), and for v = 1,024 the percentage of improvement reaches 86.3% (D_1 = 13.41 and D_2 = 15.53).

Figure 10 displays the different values for the percentage asymptotic response time improvement obtained for workload BMK4A, for different values of the context switching time. As expected, when the context switching time (sw) increases, the improvement obtained with the P-architecture decreases. For very small values of multiprogramming level (of the order of 2) the improvement may be even negative for higher values of sw.

Table 5
PARTI for the Los Alamos Benchmark

Workload	PARTI (%)
BMK1	18.5
BMK4A	48.0
BMK11A	15.8
BMK11B	29.8
BMK11C	39.0
BMK14	73.0
BMK21A	10.8
BMK24A	55.4
BMK24B	50.5
BMK24C	53.4

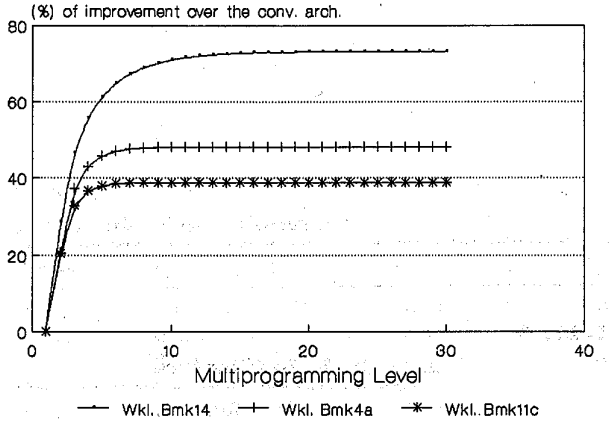


Fig. 9 Response time improvement for workloads BMK14, BMK14A, and BMK11C

Figure 11 shows the variation of the average response time for the vector-bound workload BMK4A, while Figure 12 shows the throughput as a function of the multiprogramming level for the same workload. Notice that in this case, the proposed architecture exhibits an asymptotic throughput 48% higher than the conventional architecture.

We show, in Figure 13, a situation in which two classes of workloads are considered simultaneously. Class 1 is composed of jobs of workload type BMK4A and class 2 is composed of jobs of class BMK11B. The multiprogramming level of class 1 is considered fixed and equal to 15 jobs while the multiprogramming level of class 2 is varied. The throughput for both architectures and for each class is shown in Figure 13. As expected, the throughput of class 1 decreases as the throughput of class 2 increases with the increase in the multiprogramming level of class 2. The total throughput of the P-architecture is considerably larger than that of the C-architecture.

Of course, one could think about a variation of the proposed architecture such that the running process will only lose control of the CPU if the vector instruction requested will take relatively long time—that is, will operate on a long vector. In order to evaluate this alternative let us define the threshold vector length $TVL(v)$ as a simple linear function of the average vector length, $TVL(v) = \alpha v$.

Therefore, a job interruption will only occur if there is no functional unit available and the vector size is greater than or equal to $TVL(v)$. This kind of control could be implemented by appropriate code generated by the compiler to this effect. Notice that the P-architecture as previously described is a special case of the proposed variation ($\alpha = 0$).

In this case, the expressions for D_1, D_2, and CS have to be modified as follows:

$$\text{New } D_1 = \text{old } D_1 + ic * p_v * p_B * F_{T6} * G, \tag{22}$$

where

$$G = Pr[\delta < TVL(v)] \tag{23}$$

and δ is the random variable that indicates the length of the vector operated by a vector instruction.

$$\text{New } CS = ic * p_v * p_B * sw * (1 - G) \tag{24}$$

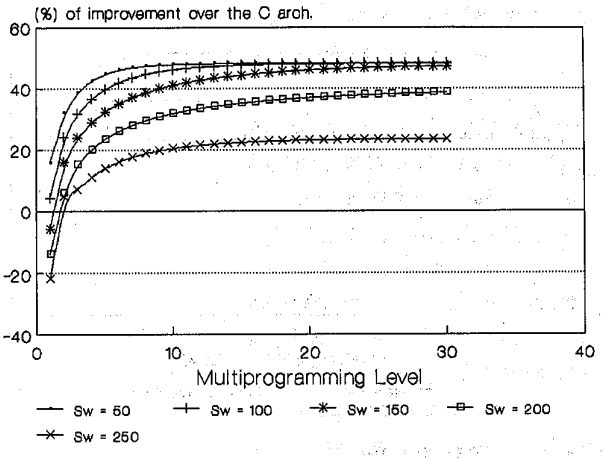
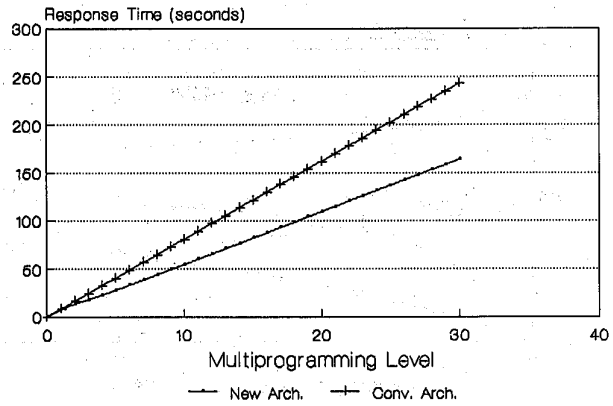


Fig. 10 Response time improvement for various values of *Sw*



BMK4A

Fig. 11 Response time for workload BMK4A

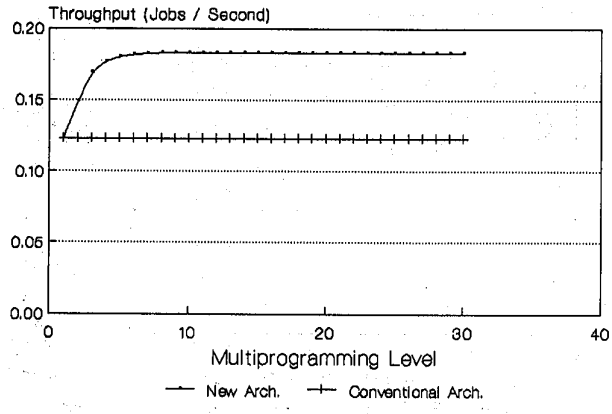


Fig. 12 Throughput for workload BMK4A

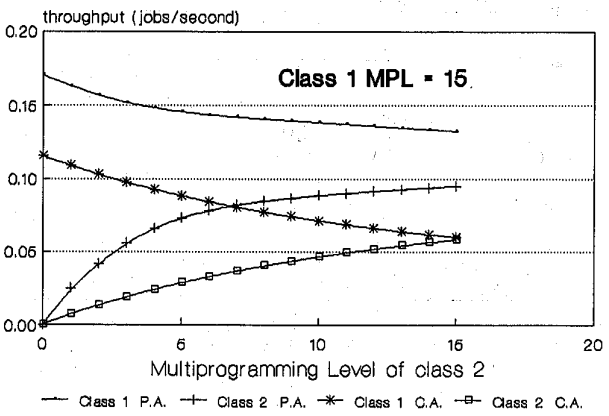


Fig. 13 Throughput of a two-class model

$$\text{New } D_2 = ic * p_v * p_B * F_{T6} * (1 - G) \qquad (25)$$

Figure 14 shows the performance of the variation of the proposed architecture for several values of G and for sw equal to 250 nsec. For $G = 0$, we obtain the same behavior of the original P-architecture. However, it can be noted that for $G > 0$ the performance metrics obtained for the variation of the proposed architecture are worse than those calculated for the P-architecture.

Several other performance studies could be easily carried out with the help of the workload characterization methodology and performance evaluation models presented here. The curves displayed above are to be considered examples of the sort of results one can obtain from the model.

5. CONCLUDING REMARKS

A lot of activity has taken place over the past few years concerning performance measurements of supercomputers using mainly the benchmarking technique (Martin, Bucher, and Warnock, 1983; Lubeck, Moore, and Mendez, 1985; Tang and Davidson, 1988). On the other hand, very little has been done on performance prediction of these machines. The work reported in this paper is, to the authors' knowledge, the first attempt to develop a predictive model of performance of supercomputers in a more general environment, where several programs are simultaneously in execution, that is, in multiprogramming environments. So far, prediction of supercomputer performance has been basically limited to the calculations of the rate of execution in floating point operations (MFLOPS) or to the estimation, through Amdahl's law or extensions to it, of the potential vector speedup of isolated programs running in a certain machine. Neither approach considers the concurrency among several jobs at the various devices of a supercomputer or the internal concurrency of operations within the CPU.

The model developed here defines a minimum set of parameters at the application, architecture, and operating system levels, that is necessary to capture the essence of the behavior of a set of applications running simultaneously on a given supercomputer architecture. Those parameters may be easily obtained in practice, as demonstrated by the fact that our numerical results

were based on measurements taken during the execution of the Los Alamos benchmark.

As stated by Martin and Mueller-Wichards (1987), in order to advance the science of supercomputer performance evaluation, measurements must be made in the context of defined models of architecture and applications. Thus, the analytic model presented here is an appropriate framework for measurements and workload characterization, besides being an important tool for performance prediction and capacity planning of supercomputers. The concurrency of operations inside the CPU was modeled by a Stochastic Petri Net. The results obtained at this level were then used to derive the needed service demand at the CPU, for a higher level queueing network model, which was used to represent the concurrency of jobs at the various devices in a multiprogramming environment. Although not considered in this paper, one could easily take into account in our model the impact on performance caused by interference at memory banks. This is particularly important in pipelined architectures (Bayley, 1987). The effect of memory bank contention delays the CPU operation. In order to include this effect in our model, one could inflate the service demands at the vector processor by a factor which should be calculated using results from memory contention models already proposed in the literature (such as Bayley, 1987). Other issues, such as modeling of memory contention and modeling of complex I/O architectures, can be handled at the queueing network model level using well-known techniques (Buzen and Shum, 1987; Lazowska et al., 1984).

Last, but not least, this paper proposes a novel architecture of supercomputers, which was shown, through our analytic model, to be always superior performance-wise to conventional supercomputer architectures for moderate to high multiprogramming levels. For the Los Alamos benchmark, the improvement reaches 73% for an R-R-type architecture and 86% for an M-M-type architecture.

Appendix A: Queueing Network Models

Queueing Network models are convenient abstractions for studying the performance of computer systems.

"The analytic model presented is an appropriate framework for measurements and workload characterization as well as an important tool for performance prediction and capacity planning for supercomputers. It defines a minimum set of parameters at the application, architecture, and operating system levels to capture the essence of the behavior of a set of applications running simultaneously on a given supercomputer architecture."

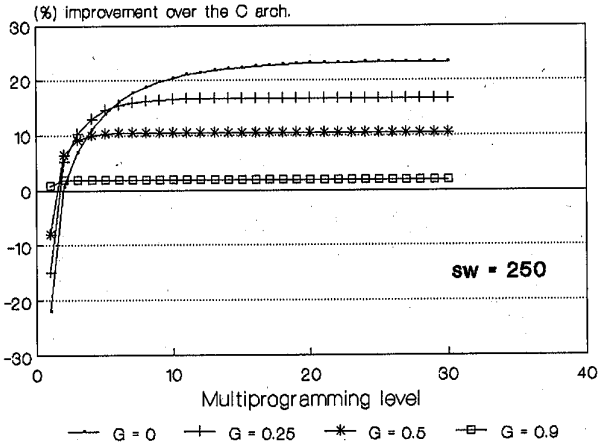


Fig. 14 Response time improvement for the new P-architecture

Queueing networks consist of (1) a collection of service centers which represent servers and queues, and (2) customers, which represent the consumers of service (i.e., jobs, tasks, processes). When the network solution can be expressed as a product of terms, where each term represents the state of an individual queue of the network, then the solution is called a product form solution. One of the major algorithms (Lazowska et al., 1984) for evaluating product form networks is the MVA (mean value analysis) algorithm, which, for one class of customers, is described by the following three iterative equations:

$n_k(0) := 0$ for $k = 1, 2, \ldots, K$

for $n = 1, 2, \ldots, N$ **do as follows**

1) $R_k(n) := D_k(1 + n_k(n-1))$ $k = 1, 2, \ldots, K$
2) $X(n) := n/\Sigma_k R_k(n)$
3) $n_k(n) := R_k(n)X(n)$ $k = 1, 2, \ldots, K$

In this algorithm, N represents the number of customers, K represents the number of servers, $R_k(n)$ represents the residence time at the service center k when the system has n customers, $X(n)$ represents the system throughput when the system has n customers, $n_k(n)$ represents the average queue length at device k when the system has n customers. D_k represents the service demand of a typical customer at service center k. This is the total amount of time the customer requires in service at that center.

Appendix B: Markov Chain Equivalent to the SPN for the C-Architecture

Basically, a Petri net PN $\equiv (P, T, A, M_0)$ is a graphical model composed of places (P), transitions (T), arcs (A), and an initial marking (M_0). In addition to its static properties, a PN has dynamic properties that result from its execution. The execution of a Petri net is controlled by the position and movements of tokens (•) in the Petri net. A PN executes by firing transitions. A transition is enabled to fire when all of its input places contain a token. A continuous stochastic Petri net SPN $\equiv (P, T, A, M_0, L)$ is formed by associating a firing rate L with each transition. Once transition T_i is enabled, its mean firing time duration is $F_i = 1/L_i$, exponentially distributed. It is known (Molloy, 1981) that any finite

place, finite transition, marked stochastic PN is isomorphic to a Markov process. In an SPN, with a given initial marking M_0, the reachability set is defined as the set of all markings that can be reached from M_0 by means of a sequence of transition firing. For our specific SPN (Fig. 5), the reachability set and the corresponding Markov chain are shown in Figure 15.

Petri Net Reachability Set

Marking	P1	P2	P3	P4	P5	P6
M1	1	1	1	0	0	0
M2	1	0	1	1	0	0
M3	0	0	1	1	1	0
M4	0	1	1	0	1	0
M5	1	1	0	0	0	1
M6	0	1	0	0	1	1
M7	1	0	0	1	0	1
M8	0	0	0	1	1	1

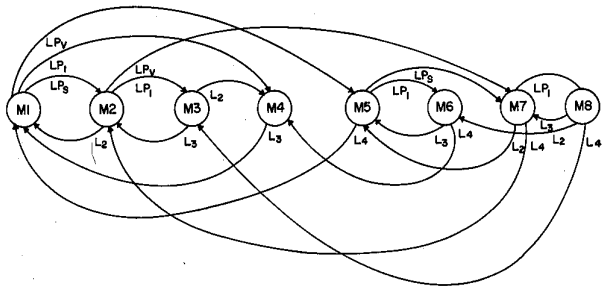


Fig. 15 Underlying Markov diagram

ACKNOWLEDGMENT

The authors wish to thank Milos Ercegovac of UCLA, Lawrence Dowdy of Vanderbilt University, Jairo Panetta of IEAv/CTA/Brazil, and the referees for their helpful comments and suggestions on the previous version of this paper.

BIOGRAPHIES

Daniel A. Menascé received the B.S.E.E. and M.Sc. degrees in computer science from the Pontificia Universidade Católica (PUC) at Rio de Janeiro, Brazil, and the Ph.D. degree in computer science from the University of California at Los Angeles (UCLA). He is currently an associate professor of computer science at the Pontificia Universidade Católica (PUC) at Rio de Janeiro, Brazil, and the current president of the Brazilian Computer Society. Professor Menascé is also a consultant to several governmental agencies. His current interests include supercomputer architecture, performance analysis of parallel machines, analytic models of computer systems and data-base management systems.

Virgilio A. F. Almeida received the M.Sc. degree from Pontificia Universidade Católica (PUC) at Rio de Janeiro, Brazil, and the Ph.D. from Vanderbilt University, Nashville, Tennessee, both in computer science. He is currently an adjunct professor of computer science at the Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil, and consultant to industry. His

current interests include supercomputer architecture, performance analysis of parallel processors, modeling of computer systems, queueing network models, and stochastic Petri net models.

SUBJECT AREA EDITOR

Dieter Mueller-Wichards

REFERENCES

Almeida, V., and Dowdy, L. 1986. Performance analysis of a scheme for concurrency/synchronization using queueing network models. *Internat. J. Parallel Programming* 15(6).

Amdahl, E. 1967. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS conf. proc.* Vol. 30.

Arvind and Iannucci, R. 1987. Two fundamental issues in multiprocessing. Computation Structures Group Memo 226-6. Laboratory for Computer Science, MIT.

Bayley, D. 1987. Vector computer memory bank contention. IEEE TC. Volume C-36. Number 3.

Bucher, I. 1983. The computational speed of supercomputers. In *Proc. of the ACM sigmetrics conf.*

Bucher, I., and Simmons, M. 1985. Performance assessment of supercomputers. In *Vector and parallel processors: architecture, applications, and performance evaluation,* edited by M. Ginsberg. Amsterdam: North-Holland.

Buzen, J., and Shum, A. 1987. A unified operational treatment of RPS reconnect delays. In *Proc. of ACM sigmetrics conf.*

Cray Research, Inc. 1982. Cray X-MP series mainframe reference manual. HR-0032.

Cray Research, Inc. 1988. UNICOS operating system for Cray supercomputers. CCMP-1108.

Dongarra, J., Martin, J., and Worlton, J. 1987. Computer benchmarking: paths and pitfalls. *IEEE Spectrum* July 1987.

Ercegovac, M., and Lang, T. 1986. Vector processing. In *Supercomputers, class VI systems, hardware and software,* edited by S. Fernbach. Amsterdam: Elsevier (North-Holland).

ETA Systems, Inc. 1986. ETA10 system reference manual. PUB 1005, Rev A. St. Paul.

ETA Sytems, Inc. 1987. Mainframe subsystem instruction specification for the ETA 10. PUB 000211. St. Paul.

Griffin, J., and Simmons, M. 1983. Los Alamos National Laboratory computer benchmarking 1983. Los Alamos Laboratory Research Report LA-10151-MS.

Hwang, K., and Briggs, A. 1987. *Computer architecture and parallel processing.* New York: McGraw-Hill.

Lazowska, E., Zahorjan, J., Grahan, G., and Sevcik, K. 1984. *Quantitative system performance: computer system analysis using queueing network models.* Englewood Cliffs, N.J.: Prentice Hall.

Lubeck, O., Moore, J., and Mendez, R. 1985.

A benchmark comparison of three supercomputers: Fujitsu VP-200, Hitachi S810/20, and Cray X-MP/2. *IEEE Comput.* 18(12).

Marsan, A., Balbo, M., and Conte, G. 1986. *Performance models of multiprocessor systems.* Cambridge: The MIT Press.

Martin, J., Bucher, I., and Warnock, T. 1983. Workload characterization for vector computers: tools and techniques. Los Alamos Laboratory Research Report LA-UR-83-305.

Martin, J., and Mueller-Wichards, D. 1987. Supercomputer performance evaluation: status and directions. *J. Supercomput.* 1(1).

Menascé, D., and Almeida, V. 1982. Operational analysis of multiclass systems with variable degree of multiprogramming and memory queueing. *Comput. Performance* 3(3).

Menascé, D., and Nakanishi, T. 1981. Optimistic versus pessimistic concurrency control mechanisms in data base management systems. In *Information systems.* New York: Pergamon Press.

Molloy, M. 1981. On the integration of delay and throughput measures in distributed processing systems. Ph.D. thesis, UCLA.

Peterson, J. 1981. *Petri net theory and the modeling of systems.* Englewood Cliffs, N.J.: Prentice Hall.

Tang, J., and Davidson, E. 1988. An evaluation of CRAY-1 and CRAY X-MP performance on vectorizable Livermore Fortran kernels. Presented at the 1988 ACM International Conference on Supercomputing, St. Malo, France.

Weiss, S., and Smith, J. 1984. Instruction issue logic in pipelined computers. IEEE TC Volume C-33. Number 11.