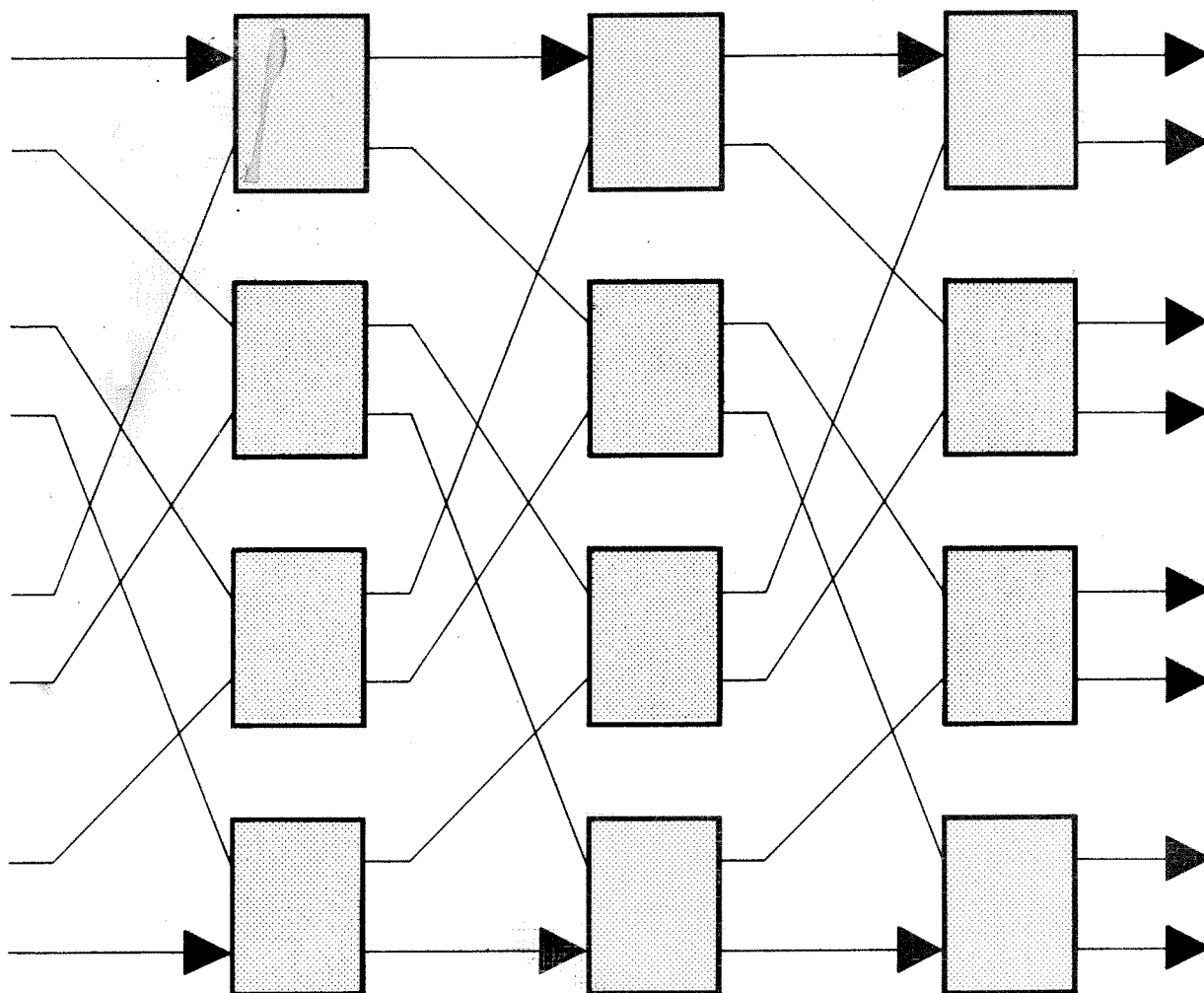# II SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES
## Processamento Paralelo - II SBAC - PP

## ANAIS - VOLUME 2

28 de setembro de 1988 - Vacance Hotel - Águas de Lindóia - SP

II SIMPÓSIO BRASILEIRO DE
ARQUITETURA DE COMPUTADORES
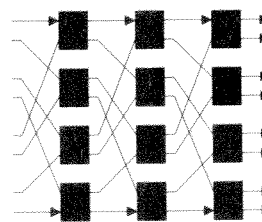Processamento Paralelo - II SBAC - PP

VACANCE HOTEL
ÁGUAS DE LINDÓIA
ESTADO DE SÃO PAULO

26 a 28 de setembro de 1988

# ANAIS — VOLUME 2

**PROMOÇÃO:**

- Sociedade Brasileira de Computação - SBC
- Instituto de Pesquisas Espaciais - INPE

**ORGANIZAÇÃO**

- Instituto de Pesquisas Espaciais - INPE
- Comissão Especial para Processamento Paralelo e Arquiteturas de Computadores da Sociedade Brasileira de Computação

**PATROCÍNIO**

- Instituto de Pesquisas Espaciais -INPE
- Financiadora de Estudos e Projetos - FINEP
- Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq
- ABC Sistemas Eletrônicos S.A./Divisão de Simuladores e Aviônica.
- Petróleo Brasileiro S/A - PETROBRÁS
- Sistema Telebrás - TELEBRÁS

**APOIO**

- Instituto de Estudos Avançados/Centro Técnico Aeroespacial - IEAv/CTA
- Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC
- Nova Lindóia Hotéis e Turismo S/A
- Granvale Turismo Ltda, São José dos Campos

# ON THE INVESTIGATION OF SUPERCOMPUTER ARCHITECTURES IN MULTIPROGRAMMING ENVIRONMENTS USING ANALYTIC MODELS

Daniel A.Menasce
Departamento de Informática
Pontificia Universidade Católica (PUC-RJ)
22453 Rio de Janeiro, Brazil

Virgilio A.F.Almeida
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
30161 Belo Horizonte, Brazil

**RESUMO**

Os supercomputadores tem sido usados para aplicações que exigem uma alta capacidade de desempenho, tais como: previsão atmosférica, simulação de novas aeronaves e automóveis, modelagem de reservatórios de petróleo e outras. Dado o seu alto custo, os supercomputadores executam sistemas operacionais multiprogramados (ex: Cray/UNICOS), que permitem que seus recursos sejam compartilhados por vários usuarios remotos e locais. Portanto, é importante avaliar o desempenho de supercomputadores em ambientes de multiprogramação. A maioria dos estudos de desempenho existentes concentra atenção na avaliação da velocidade de programas que executam isoladamente em um supercomputador.

Modelos analíticos, baseados nas redes de filas e redes de Petri, são desenvolvidos neste trabalho com duas finalidades. A primeira delas é avaliar o desempenho de supercomputadores em ambientes de multiprogramação com várias classes de usuários executando simultâneamente. A outra finalidade consiste da avaliação de uma proposta aqui apresentada para uma modificação da arquitetura dos supercomputadores vetoriais. Varios exemplos numéricos são apresentados para ilustrar os modelos apresentados.

**ABSTRACT**

Supercomputers 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, to name a few. These machines run multiprogrammed time-sharing operating systems, 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 supercomputers performance are concerned sith the evaluation of the effective speed of a program running in isolation on a particular supercomputer. 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, 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.

# 1. 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, to name a few. 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, 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 67). The reader is referred to (Bucher 83), (Bucher 85), (Lubeck 85),(Dongarra 87), for studies, based on actual measurements of benchmarks, which analyze the effective speed of vector computers in uniprogramming environments.

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 86), (Lazowska 84). The reason stems from the concurrency that exists between processors working on the same job. In order to model this concurrency, a SPN model (Molloy 81), (Marsan 84) of a job executing in isolation is used at the lower level. An upper level model, i.e., the QN model, is used to represent the multiprogramming environment. The combination of both modelling techniques leads to a new supercomputer performance model.

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

# 2. SUPERCOMPUTER ARCHITECTURES

Vector computer architectures are characterized by CPUs composed of three different types of processors:

a. Instruction Processor (IP): it is the unit that fetches, decodes, prepares, and executes some special instructions.

b. Scalar Processor (SP): it is the unit that executes scalar instructions.

c. Vector Processor (VP): it 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 (Lubeck 85). See Figure 1 for a schematic view of the architecture of a conventional supercomputer.
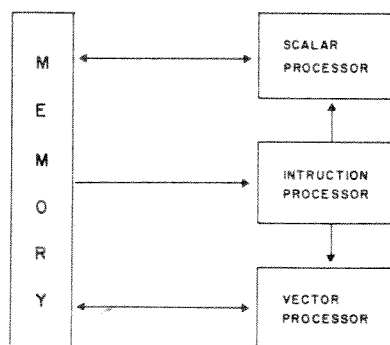


Figure 1 - Organization of a Conventional Supercomputer

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.

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.

There are basically two types of architectures of vector processing units: those which, like the CDC Cyber-205, reference memory directly in their vector instructions, and those which, like the Cray-1, 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 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 in((Hwang 87), Ercegovac 86) and (Weiss 84)).

A vector operation on a vector of length 1000 may take roughly 11 μs on a Cyber-205 and 30 μs on a Cray-1 computer (Bucher 83). 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:

i. 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;

ii. 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;

iii. 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.

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.
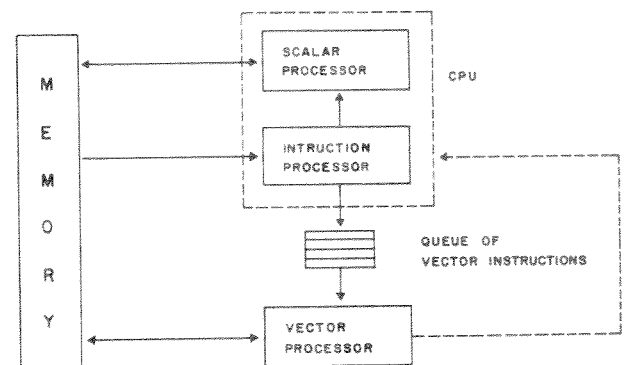


Figure 2 - Proposed Architecture for Supercomputers,

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

## 3. WORKLOAD CHARACTERIZATION MODEL OF SUPERCOMPUTERS

An interesting report on workload characterization for vector computers was carried out by Martin et al.( Martin 83) 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.

In this section we will take a slightly different approach towards 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 be Figure 3. The parameters at these three levels will then be mapped into the analytic model parameters as will be discussed in section four.
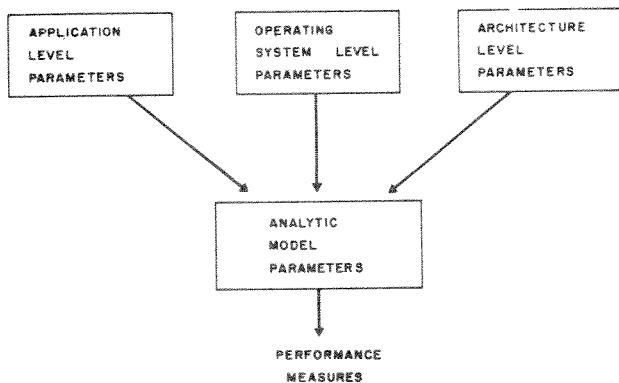


Figure 3 - Workload Characterization Approach

## 3.1. 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 .

$p_I$ : 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

$$p_S + p_V + p_I = 1 \qquad (1)$$

The remaining parameters at this level are:

---

(1) 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.

v : average vector length

ic : instruction count, i.e. number of executed instructions.

A question that arises is whether these data may be easily obtained in practice. The answer is affirmative as can be deduced from (Martin 83),(Bucher 85) 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 in (Martin 83), (Bucher 83) and (Lubeck 85). Also, Table III of (Martin 83) 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 $p_V$ respectively. Finally, $p_I$ may be derived from equation (1)above.

## 3.2. Operating System Parameters

The relevant parameters at this level are:

R : number of different types of 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 $(1 \leq e \leq R)$.

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 are capable of switching the context with a single instruction, making this process much faster.

## 3.3. 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.

$nC_{ip}$: 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.

$nf_v$ : average number of vector
functional units in use.

$\emptyset_T(v)$: function that determines the
average execution time of a
vector instruction on vector
of average length v for
architectures of type T (T =
M-M or R-R).

So, according to (Bucher 83),

$$\emptyset_{M-M}(v) = Tstar+v*Telem \qquad (2)$$

where

Tstart: startup time for the
vector operation

$T_{elem}$ : time per result element

$$\emptyset_{R-R}(v)=T_{start}+v=(T_{start}strip/V_{max}+T_{elem}) \quad (3)$$

where $T_{start}$ and $T_{elem}$ are as defined
above and

$V_{max}$: number of elements of the
vector register

$T_{start}strip$: time to load the
vector register.

For instance, (Bucher 83) shows that
for the Cyber-205 supercomputer(which
is of the M-M type), the following
relationship holds

$$\emptyset_{cyber-205}(v) = 1000+10*v \qquad (4)$$

where the constants in the the above
equation are given in nanoseconds(ns).

## 4. ANALYTIC MODEL OF SUPERCOMPUTERS

In order to evaluate and compare
the C-Architecture and the P-Architec-
ture we are going to use a two-level
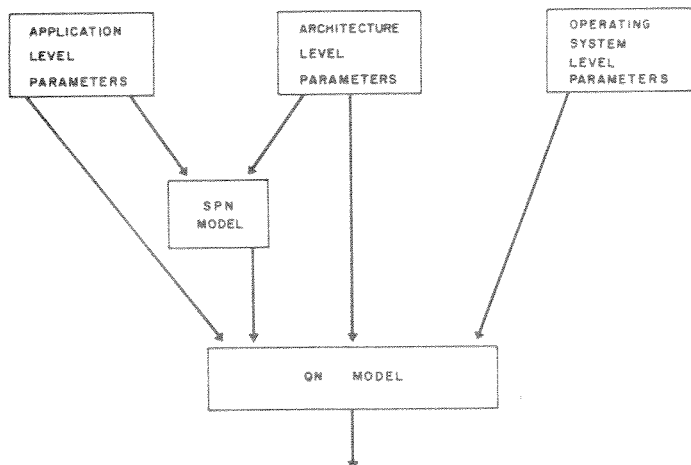modelling approach (Menasce 1981)
indicated by Figure 4.



Figure 4 - Two-level Modelling
Approach

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 (Lazowska 84). 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
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.

### 4.1. Analytic Model for the C-Architecture

Consider the SPN shown in Figure
5 which represents the CPU composed of
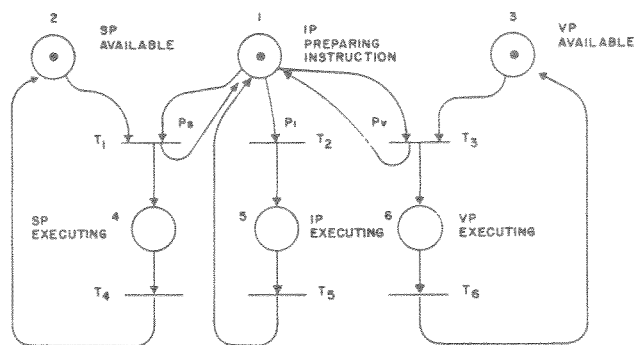the IP, SP and VP.



Figure 5 - SPN for the C-Architecture

The following meanings are
associated with the various places of
the above SPN when there is a token in
the place:

| Place # | Meaning when there is one token in it |
|---|---|
| 1 | IP is preparing an instruction |
| 2 | SP is available |
| 3 | VP is available |
| 4 | SP is busy executing instructions in all its functional units. |
| 5 | IP is executing an instruction |
| 6 | VP is busy executing instruction in all its functional units. |

The firing time of transitions $T_1$ $T_2$ and $T_3$ represent the time need 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

$$F_p = C_{ip} = nc p_{ip} \qquad (5)$$

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,

$$F_{T4} = c_{ip} * nc_{sp} / nf_S \qquad (6)$$
$$F_{T5} = c_{ip} * nc_{ip} \qquad (7)$$
$$F_{T6} = \emptyset(v) / nf_V \qquad (8)$$

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 axis, 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 i-th scalar instruction of a task, $V_I$ the i-th vector instruction of a task, and $I_I$ the i-th 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.
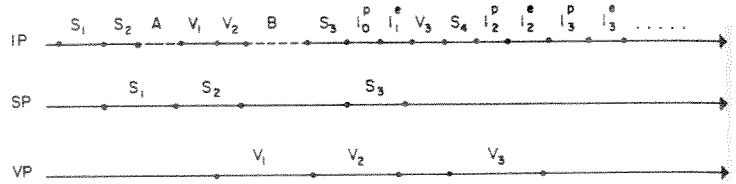


Figure 6 - Execution Sequence at th IP, SP and VP

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 i tervals, and total duration of all type B intervals.

$$D_{CPU} =$$
$$= ic * (c_{ip} * ncp_{ip} + p_I * c_{ip} * nc_{ip} + p_S * p_A * F_{T4} + p_V * p_B * F_{T6})$$
$$(9)$$

where $p_A$ is the probability that a type A idle period occurs when a scalar instruction is to be issued. This is simply the sum of the probabilities (Pr) of two markings (see Appendix A) in the SPN as indicated below

$$p_A = Pr( 1, 0, 1, 1, 0, 0 ) + Pr( 1, 0, 0, 1, 0, 1 ) \qquad (10)$$

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

$$p_B = Pr( 1, 1, 0, 0, 0, 1 ) + Pr( 1, 0, 0, 1, 0, 1 ) \qquad (11)$$

The queueing network that represents the C-Architecture is shown in Figure 7. Server 1 represents the CPU whose service demand for class r $D_{CPU}$ 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.
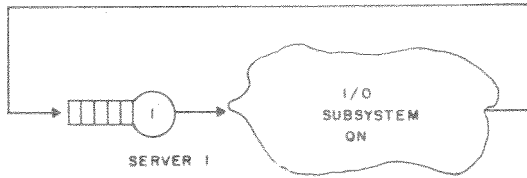
Figure 7 - QN model for the
C-Architecture

Numerical results obtained with this
model are given in section 5.

## 4.2. 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 multiprogram-
ming 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 modelled 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, 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.

$$D_1P = D_1 + CS \qquad (12)$$

where $D_1$ and CS are given by
expressions (12.a) and (13)
respectively.

$$D_1 = ic * (c_{ip} * nc_{ip} + p_I * \dot{c}_{ip} *$$
$$nc_{ip} + p_S * p_A F_{T4} \qquad (12a)$$

where $p_A$ is defined in (10).
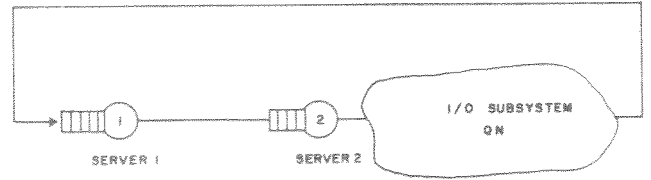
$$CS = ic * p_V * p_B * sw \qquad (13)$$



Figure 8 - QN Model for the P-
Architecture

Expression (12) 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 and therefore expression (12)
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 non-overlapped 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

$$D_2 = ic * p_V * p_B * F_{T6} \qquad (14)$$

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 expression (12.a)
and (14)

## 5.   NUMERAL RESULTS

The Input/Output portion of the
computer system was disregarded  in
the numerical studies conducted for
this paper, since both 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 the Los
Alamos Benchmark ((Martin 83), (Bucher
83) and (Lubeck 85)). For our numerical
example, we use parameters from the
Cray architecture.

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

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

Table I - Architecture Level Parameters(R-R)

The Application Level Parameters are indicated in Table II below. The identification of the workloads is the one used in the Los Alamos Benchmark. The meaning of the rightmost column will be discussed shortly.

| wor-kload | $p_v$ | $p_s$ | ic (in millions) | 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 II - Application Level Parameters

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:

$$R_v * (p_v * v) / (p_s + p_v * v) \qquad (15)$$

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 II one can see that codes BMK11C, BMK14, BMK4A, BMK24B, BMK25C, BMK1, and BMK24A are vector bound applications, codes BMK11A and BMK11B are balanced and code BMK21A is scalar bound. The switch time, sw, used in all examples is 50 ns.

From the input parameters given in tables I and II above one may solve the SPN and calculate the service demands $D_1$ and $D_2$ for the p-Architecture according to expressions (12), (13) and (14). Recall that the service demand for the C-Architecture is the $D1$ plus $D_2$ as indicated in expression (9). Table III shows the values of $D_1$ and $D_2$, in seconds, obtained by solving the SPN. These values are compatible with those obtained in the Los Alamos Benchmark (Los Alamos 83), which validates our model in a uniprogramming environment.

| Work-load | $D_1$ (sec) | CS (sec) | $D_2$ (sec) | P.A.R.T.I. (%) |
|---|---|---|---|---|
| BMK1 | 24.1 | 0.40 | 4.472 | 18.5 |
| BMK4A | 2.633 | 0.784 | 5.465 | 48.0 |
| BMK11A | 8.61 | 0.122 | 1.368 | 15.8 |
| BMK11B | 6.372 | 0.17 | 1.9 | 29.8 |
| BMK11C | 1.477 | 0.336 | 3.797 | 39.0 |
| BMK14 | 1.078 | 0.078 | 0.789 | 73.0 |
| BMK21A | 3.538 | 0.0425 | 0.384 | 10.8 |
| BMK24A | 1.42 | 0.09 | 0.788 | 55.4 |
| BMK24B | 5.34 | 0.24 | 2.7 | 50.5 |
| BMK24C | 12.0 | 0.647 | 6.41 | 53.4 |

Table III - Resource Demands and P.A.R.T.I. for the Los Alamos Benchmark

Experimentation with an event-driven simulation program has provided validation of our analytic models in multiprogramming environments. In the Table below we present 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 two throughputs validates our results.

| N | Analytic | Simulation | C.I. | Error |
|---|---|---|---|---|
| 4 | 6.35 | 6.74 | 6.36:7.11 | 5.7 |
| 5 | 7.69 | 8.34 | 7.69:8.95 | 7.4 |
| 10 | 15.31 | 16.10 | 16.0:16.3 | 4.9 |

Table IV: Throughput of the p-Architecture

The values of the Percentage Asymptotic Response Time Improvement for some codes of the Los Alamos Benchmark are given in Table III. As it can be observed, in some cases the improvement is quite remarkable, as is the case with code BMK14. The smallest observed improvement was 10.8 % while the largest one was 73%. The response time improvement as a function of the multiprogramming level is depicted graphically for workloads BMK14, BMK4a and BMK11c in Figure 10.
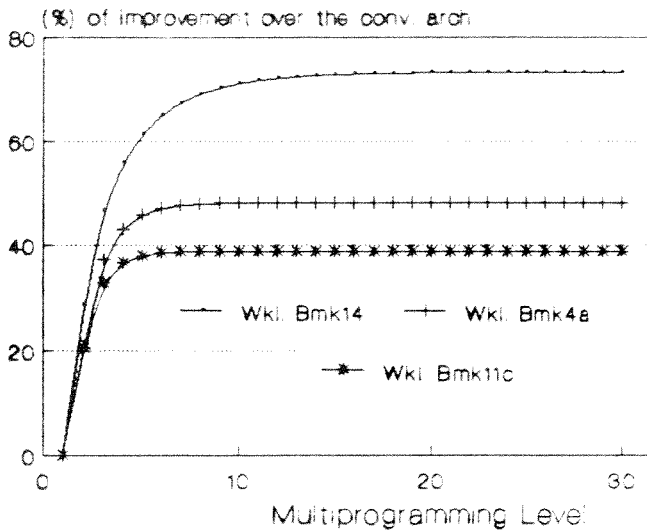


Figure 9 - Response Time Improvement for Workloads BMK14, BMK4a, and BMK11c.

Figure 10 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.
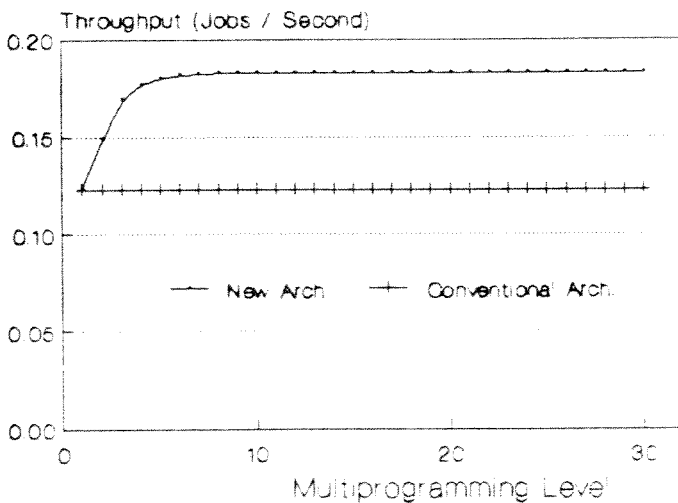


Figure 10 - Throughput for Workload BMK4a.

We show now, in Figure 11 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 the figure. 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.
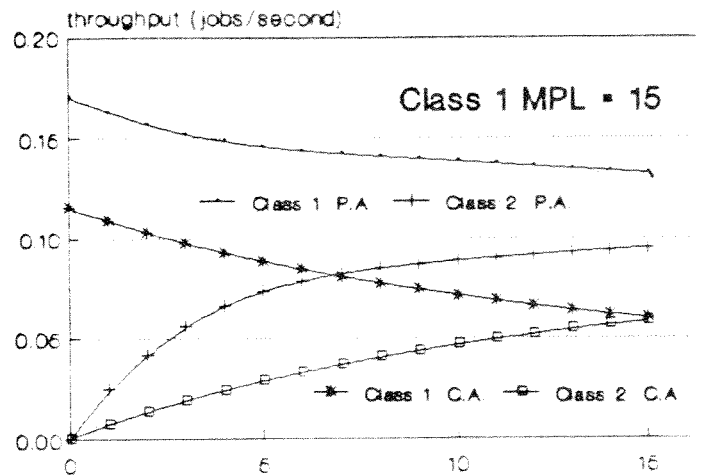


Figure 11 - Throughput of a two-class model.

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 just an example of the sort of results one can obtain from the model.

3.    CONCLUDING REMARKS

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, i.e., in multiprogramming environments. So far, prediction of supercomputer performance has been basically limited to the calculation 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 nor 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 Muller-Wichards (Martin 87), 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, it is a trivial matter to take into account at the QNM level other aspects, such as modeling of memory contention and modeling of complex I/O architectures, using well known techniques (Jacobson 82, Almeida 87, Buzen 87).

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 the Los Alamos benchmark, the range of improvement goes from 10,8% to 73%.

## REFERENCES

(1) Almeida 86, "Performance Analysis of a Scheme for Concurrency/Synchronization Using Queueing Network Models", V.Almeida and L. Dowdy, International Journal of Parallel Programming, Vol.15, No. 6, 1986.

(2) Almeida 87, "Aproximate Solution Techniques for Queueing Network Models of Concurrent Processing and others Non-Product Form Problems", Ph.D.Dissertation, Vanderbilt University, August 1987.

(3) Amdahl 67, "The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," AFIPS Conf.Proc., Vol.30, 1967.

(4) Bucher 83, "The Computational Speed of Supercomputers," Ingrid Y.Bucher, Proceedings of the ACM Sigmetrics Conference, 1983.

(5) Bucher 85, "Performance Assessment of Supercomputers," Bucher, Ingrid Y.and Margaret L.Simmons, in Vector and Parallel Processor: Architecture, Applications, and Performance Evaluation, et.Myron Ginsberg, North Holland, 1985.

(6) Buzen 87, "A Unified Operational Treatment of RPS Reconnect Delays", J.Buzen and A.Shum, Proceedings of ACM Sigmetrics, 1987.

(7) Dongarra 87, "Computer Benchmarking: Paths and Pitfalles", J.Dongarra, J.Martin, and J.Worlton, IEEE Spectrum, July 1987.

(8) Ercegovac 86, "Vector Processing", Ercegovac, Milos and Thomas Lang, in Supercomputers, Class VI Systems, Hardware and Software, ed. S.Fernbach, Elsevier Sciense Publishers (North-Holland), 1986.

(9) Hwang 87, "Computer Architecture and Parallel Processing," Kai Hwang and Faye' A.Briggs, McGraw-Hill International Editions, 3rd Printing, 1987.

(10) Jacobson 83, "Analyzing Queueing Network with Simultaneous Resource Possession", P.Jacobson and E. Lazowska, CACM, Vol.25, No.2, February 1982.

(11) Lazowska 84, "Quantitative System
Performance: Computer System
Analysis Using Queueing Network
Models," Lazowska, E.D., J.
Zahorjan, G.S.Graham, and K. C.
Sevcik, Prentice Hall,Englewood
Cliffs, N.J., 1984.

(12) Los Alamos 83, "Los Alamos Natio-
nal Laboratory Computer Benchmar-
king 1983"' J.Griffin and M.
Simmons, LA-10151-MS, 1983.

(13) Lubeck 85, "A Benchmark Comparison
of Three Supercomputers: Fujitsu
VP-200, Hitachi S810/20, and Cray
X-MP/2", Lubecj Olaf, James Moore,
and Raul Mendez, IEEE Computer,
December 1985.

(14) Marsan 84, "A Class of Generalized
Stochastic Petri Nets for the
Performance Evaluation of
Multiprocessors", A.Marsan, M.
Balbo, and G.Conti, ACM, Vol.2,
No.2, 1984.

(15) Martin 83, "Workload Characteriza
tion for Vector Computers: Tools
and Techniques," Martin. Joanne
L., Ingrid Y. Bucher, and Tony T.
Warnock, Los Alamos National
Laboratory Research Report LA-UR-
83-305, Los Alamos, New Mexico,
USA, 1983.

(16) Martin 87, "Supercomputer Perfor-
mance Evaluation: Status and
Directions", J.Martin and D.
Mueller-Wichards, The Journal of
Supercomputing, Vol.1, No.1, May
1987.

(17) Menasce 81, "Optimistic versus
Pessimistic Concurrency Control
Mechanisms in Data Base Management
Systems", Information Systems,
Pergamon Press, Vol.7, No.1, 1981.

(18) Menasce 82, "Operational Analysis
of Multiclass Systems with
Variable Degree Multiprogramming
and Memory Queueing", D.Menasce
and V.Almeida, Computer Performan
ce, Vol.3, No.3, September 1982.

(19) Molloy (81), "On the Integration
of Delay and Throughput Measures
in Distributed Processing System",
Ph.D.Thesis, UCLA 1981.

(20) Peterson 81, Petri Net Theory and
the Modeling of Systems, Prentice
Hall, 1981.

(21) Weiss 84, "Instruction Issue
Logic in Pipelined Computers,"
S.Weiss Shlomo and James E.Smith,
IEEE TC Vol.C-33, No 11,November
1984.

**APPENDIX A**: Markov Chain Equivalent
to the SPN for the C-
Architecture

Basically, a Petri Net PN (P, T,
A, $M_O$) is a graphical model composed
of places (P), transitions (T), arcs
(A), and an initial marking ($M_O$). 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 (P, T, A,
M , 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 81) that any finite pla-
ce, finite transition, marked
stochastic PN is isomorphic to a
Markov process. In a SPN, with a
given initial marking $M_O$, the
reachability set is defined as the set
of all markings that can be reached
from $M_O$ by means of a sequence of
transition firing. For our specific
SPN (figure 5), the reachability set
and the corresponding Markov chain are
shown belox.

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 |