A Customized MVA Model for ILP Multiprocessors*

Daniel J. Sorin†, Mary K. Vernon†, Vijay S. Pai‡, Sarita V. Adve‡, and David A. Wood†

†Computer Sciences Dept
University of Wisconsin - Madison
{sorin, vernon, david}@cs.wisc.edu

‡Dept of Electrical & Computer Engineering
Rice University
{vijaypai, sarita}@rice.edu

University of Wisconsin-Madison Computer Sciences Technical Report #1369.
Rice University Electrical and Computer Engineering Technical Report #9803.

Abstract

This paper provides the customized MVA equations for an analytical model for evaluating architectural alternatives for shared-memory multiprocessors with processors that aggressively exploit instruction-level parallelism (ILP). Compared to simulation, the analytical model is many orders of magnitude faster to solve, yielding highly accurate system performance estimates in seconds.

1 Introduction

In [8], we presented an analytical model for evaluating specific types of architectural trade-offs for shared-memory systems with ILP processors. As shown in that paper, the analytical model validates extremely well against detailed simulation and produces results in a few seconds.

The principal aspects of the model are:

- The ILP processor and its associated two-level cache system are viewed as a black box that generates requests to the memory system and intermittently blocks after a dynamically changing number of requests.
- We iterate between two submodels; one represents the blocking behavior due to load misses that cannot be retired until the data returns from memory, and the other submodel represents the blocking behavior due to the hardware constraint on the total number of outstanding memory requests.
- In each submodel, the memory system is viewed as a system of queues (e.g., the memory bus, DRAM modules and associated directories, and network interfaces) and delay centers (e.g., switches in the interconnection network). We create a set of intuitive *customized* mean value analysis (CMVA) equations to obtain estimates of throughput (instructions per cycle) in each submodel. The CMVA technique has proven to be accurate in validation experiments for a number of simpler architectural models [9].

The purpose of this technical report is to provide the details of the customized MVA equations which were omitted in [8] due to space constraints. Section 2 of this report provides the model input parameters. Section 3 provides an overview of the analytical model, and Section 4 presents the customized MVA equations. Further discussion of the model, including validation and applications, can be found in [8].

*This research is supported in part by DARPA/ITO under Contract N66001-97-C-8533 and by the National Science Foundation under Grants HRD-9896132, MIP-9625558, CDA-9623632, CCR-9410457, CCR-9502500, CDA-9502791, and CDA-9617383. Sarita V. Adve is also supported in part by an IBM University Partnership award and by the Texas Advanced Technology Program under Grant No. 003604-025. Vijay S. Pai is supported by a Fannie and John Hertz Foundation Fellowship.

parameter	description
N	number of nodes
m	memory modules per node
M_{hw}	number of MSHRs
$S_{NI_{out,r}}$	NI send occupancy for request
$S_{NI_{out,d}}$	NI send occupancy for data
$S_{NI_{in,r}}$	NI receive occupancy for request
$S_{NI_{in,d}}$	NI receive occupancy for data
$S_{bus,r}$	bus occupancy for request
$S_{bus,d}$	bus occupancy for data
S_{mem}	memory/directory (DRAM) access
S_{tag}	L2 tag check
S_{switch}	per-header network switch occupancy

Table 1. System Architecture Parameters

Parameter	Description
τ	Average time between read, write, or upgrade requests to memory, not counting the time when the processor is completely stalled or is spin-waiting on a synchronization event
CV_τ	Coefficient of Variation of τ
$f_{synch-write}$	Fraction of write requests that are generated by atomic read-modify-write instructions or that coalesce with at least one later read
f_M	Fraction of processor stalls that find M MSHRs with outstanding read requests
$P_{read}, P_{write}, P_{upgrade}$	Probability that a memory request is a read, write, or upgrade
P_{wb}	Probability that a read or write request causes a writeback of a cache block
$P_{L\|x}$	Probability directory is local for a type x transaction; x=read, write, upgrade, writeback
$P_{M\|x,y}$	Probability home memory can supply the data for a type x, y request; x=read, write; y=local home, remote home
$P_{3hop\|x\&not-memory}$	Probability that a request of type x to a remote home is forwarded to a cache at a third node; x=read,write
H	Average number of network switches traversed by a packet
X	Average number of invalidates caused by a write or upgrade to a clean line

Table 2. Application Parameters

2 System Architecture and Model Parameters

2.1 System Architecture

The architecture modeled is a cache-coherent, release consistent shared-memory multiprocessor system where the processing nodes are connected by a mesh interconnection network [8].

2.2 Model Parameters

Model parameters can be classified as either describing the system or describing the application. Table 1 defines the system parameters, while Table 2 summarizes the application parameters. From the parameters in Table 2, we can compute the probabilities of the protocol transactions in Table 3.

The first four parameters in Table 2 characterize the ability of the processor to overlap multiple memory requests while running a given compiled application (or set of applications). These parameters, referred to as ILP parameters, are discussed in more detail below. The other parameters in the table are standard parameters for models of architectures based on directory coherence protocols [1]. Note that the parameters are defined for *homogeneous applications*; that is, each processor has the same value for each parameter in the table, and memory requests are assumed to be equally distributed across the relevant memory modules (local or remote) due to interleaving and effective data layout. There is a natural extension of these

parameters for non-homogeneous applications, but for simplicity in the model exposition we use the given parameters.

The parameter τ is the average time between requests generated by the processor to the (main) memory subsystem, not including the time that the processor is stalled or is spin-waiting on a synchronization event such as a lock release, flag, or barrier completion. We also measure the coefficient of variation of τ, CV_τ. τ is well-defined for simple processors that block on each load and store, whereas the notion that a complex modern processor is stalled has several possible definitions. For the robust parameter τ that is needed for the model, the processor is defined to be stalled when it is *completely stalled*; that is, the functional units are completely idle, no further instructions can be retired or issued until data returns from memory, and all outstanding cache requests are waiting for data from main memory. The fraction of time a processor is completely stalled is one of the performance metrics estimated by the analytic model. The parameter τ does not include this time.

The $f_{synch-write}$ parameter is the fraction of write requests that are synchronous; that is, they are generated by read-modify-write requests or they coalesce with at least one later read miss. Read misses that coalesce with earlier read requests are completely invisible to the model because they do not generate any memory system traffic, and they do not cause any new blocking behavior. Thus, a parameter for the frequency of read-read coalescing is not needed. Likewise for writes that coalesce with previous misses.

The set of parameters $f_M, M \geq 1$, are the *fractions of processor stall events* that have M MSHRs occupied with read misses. These fractions are defined and measured for a system with a number of MSHRs larger than the maximum value that will be evaluated with the model. We will refer to such a system as an "infinite MSHR" system. Note that if a read miss occurs for a line that has a prior write miss outstanding, then the miss is counted as a read miss when measuring M. Also note that misspeculated reads are counted in M. The f_M parameters are unique to a system with non-blocking loads.

We have verified that the application input parameters are *relatively insensitive to changes in the memory system architectural parameters* that can be varied in the model (e.g., the number of MSHRs, the speed of the bus and interconnection network switches, main memory configuration, etc.). However, the application parameters are sensitive to various parameters of the processor and cache architecture. For example, τ, CV_τ, f_M, and $f_{synch-write}$ are sensitive to the instruction window size.

3 The Analytic Model

The principal output measure computed by the model is the system throughput, measured in instructions retired per cycle (IPC). This throughput is computed as a function of the input parameters that characterize the workload and the memory architecture. The customized MVA equations defined in this report assume that the directory lookup is coupled with memory access, so a single service time applies to the parallel memory and directory lookup.

3.1 Model Overview

We use the term *synchronous* for read requests (and for read-modify-write requests) because the data must return before a load (or read-modify-write) instruction is retired from the instruction window. Other requests (writes, upgrades, writebacks, invalidates, and acknowledgments) are *asynchronous*. Table 3 defines all of the memory system transactions.

A key question in developing the analytic model is how to compute throughput as a function of the dynamically changing number of outstanding memory requests that can be issued before the processor must stall waiting for data to return from memory. We address this issue by iterating between the following two submodels for each value of M, $1 \leq M < M_{hw}$:

- *the synchronous blocking submodel (SB)* that computes the fraction of time the processor is stalled due to load or read-modify-write instructions that cannot be retired until the data returns from memory,
- *the MSHR blocking submodel (MB)* that computes the *additional* fraction of time the processor is stalled purely due to the MSHRs being full.

For $M = M_{hw}$, we compute throughput from a modified version of the MSHR-blocking submodel alone, as explained below. Once these throughputs are computed, we compute the *weighted sum* of the throughputs, weighted by the frequency of each throughput value that would be observed for the number of MSHRs in the system. This frequency can in turn be computed from the model input parameters, f_M. The remainder of this section gives the most pertinent details of the two submodels as well as how slowdown due to synchronization delays is computed; the full set of equations for the submodels is given in Section 4.

	Reads					Upgrades	
	local home		remote home			local	remote
	memory	remote cache	memory	cache at home	cache at non-home		
transaction name	LC	LHCC	RC	RHLCC	RHRCC	LUPG	RUPG

	Writes					Writebacks	
	local home		remote home			local	remote
	memory	remote cache	memory	cache at home	cache at non-home		
transaction name	LCINV	LHCCwr	RCINV	RHLCCwr	RHRCCwr	LWB	RWB

Table 3. Protocol Transactions

3.2 The Two Submodels

Each of the two submodels (SB and MB) contains similar sets of customized MVA equations to compute the response time for a transaction in the memory subsystem (see section 4). The only differences between the submodels are in the equations for the overall residence time, R, and the processor residence time, R_{pe}. We discuss these differences in this section and then discuss the CMVA equations that are common to both submodels in Section 4.

R consists of the residence times at the processor, network, network interface (NI), bus (both local and remote), and memory. It also includes a term, Z, which represents the latencies at resources with negligible contention (e.g., cache tag check). The difference in the equations for R, besides the difference in R_{pe}, is that R_{SB} is the sum of the residence times of the synchronous transactions, whereas R_{MB} is the sum of the residence times of the asynchronous transactions plus the residence times of the reads that are synchronous in the MB submodel (this will be discussed below). These equations are:

$$R_{SB} = R_{pe_{SB}} + R_{NET}^{synch} + R_{NI}^{synch} + R_{bus}^{synch} + R_{mem}^{synch} + Z^{synch}$$

$$R_{MB} = R_{pe_{MB}} + R_{NET}^{asynch} + R_{NI}^{asynch} + R_{bus}^{asynch} + R_{mem}^{asynch} + Z^{asynch}$$
$$+P_{synch-read-MB}(R_{NET}^{synch} + R_{NI}^{synch} + R_{bus}^{synch} + R_{mem}^{synch} + Z^{synch})$$

In the SB submodel, the number of customers per processor is equal to the maximum number of read requests that can be issued before the processor blocks (i.e., one of the observed values of M). The processor (and its associated cache subsystem) is a FCFS queue that initially has mean service time equal to τ. Note that this queue is only idle when M memory read requests are outstanding; otherwise it is generating memory requests at rate $1/\tau$. If the request is a write miss, the customer is routed immediately back to the processor while simultaneously forking an asynchronous memory write or upgrade transaction, using a technique similar to that proposed by Heidelberger and Trivedi [5]. Thus, the R_x^{synch} and Z^{synch} terms are only non-zero for read requests (see Section 4).

In the MB submodel, the number of customers per processor is equal to the number of MSHRs, M_{hw}. MSHRs can be occupied by read, write, or upgrade requests; however, for architectures with non-blocking stores and in-order retirement of loads and for $M < M_{hw}$, the blocking time when MSHRs contain M read requests is accounted for in the SB submodel. The additional blocking time that needs to be computed by the MB model is for the case that the MSHRs contain M_{hw} requests of which *less than* M are read requests. That is, we can measure M read requests for the "infinite MSHR" system, but the system with M_{hw} MSHRs could block with fewer than M reads in the MSHRs because some registers are filled with other requests. All writes and upgrades, plus some read requests, must be synchronous in the MB submodel to account for this additional blocking behavior.

The following four equations account for the read requests that should be synchronous in the MB submodel[1]. The first equation estimates P_j, the probability that only j of the M reads that were measured in the "infinite MSHR" system, $1 \leq j < M$, are in the first M_{hw} MSHRs. The second equation estimates the utilization of the processor in the SB submodel, and this utilization is used in the third equation to compute U'_{pe}, which is an estimate of the probability that a customer leaving the processor in the SB submodel is leaving behind a non-empty processor queue. The third equation has a term, U_{SB}, which is the probability that a processor in the SB submodel is not stalled, and it will be explained below. By multiplying P_j by U'_{pe} and summing over j, we obtain the probability that a read should be considered synchronous in the MB submodel, as shown in the last equation.

P_j = Prob[j reads in MSHRs | at least 1 read in MSHRs] = $\begin{pmatrix} M_{hw}-1 \\ j-1 \end{pmatrix} (P_{write} + P_{upgrade})^{(M_{hw}-j)} P_{read}^{(j-1)}$

$U_{pe} = \frac{M\tau}{R_{SB}}$

$U'_{pe} = \frac{U_{pe}\frac{M-1}{M}}{U_{pe}\frac{M-1}{M}+1-U_{SB}}$

$P_{synch-read-MB}$ = Prob[read is synchronous in MB submodel] = $\sum_{j=1}^{M_{hw}-1} P_j U'_{pe}$

The read misses that are not synchronous in the MB submodel are immediately routed back to the processor (since the processor cannot stall on these read misses in this submodel) while simultaneously forking a read transaction to the memory system, again using a technique similar to that in [5].

As mentioned above, the other difference in the equations between the two submodels concerns R_{pe}. This difference arises from how we represent the processor stall time that is estimated by one submodel in the other submodel. That is, the mean time that each customer occupies the processor in the MB submodel is equal to τ_{MB}, where τ_{MB} is τ adjusted to reflect the fraction of time that the processor is stalled due to load or read-modify-write instructions that cannot be retired (computed from the SB model). That is, $\tau_{MB} = \frac{\tau}{U_{SB}}$. Once the measures are computed from the MB model, the SB model is solved again using $\tau_{SB} = \frac{\tau}{U_{MB}}$, where U_{MB} is the fraction of time that the processor is not stalled in the MB submodel.

$U_{SB} = \frac{M}{R_{SB}}\frac{\tau}{U_{MB}}$

$U_{MB} = \frac{M}{R_{MB}}\frac{\tau}{U_{SB}}$

The alternating solution of each submodel is repeated until the estimated throughputs converge. This approach might be named the "method of surrogate service time inflation," analogous to the method of surrogate delays [4, 6].

The equations for $R_{pe_{MB}}$ and $R_{pe_{SB}}$ are shown below. The processor residence time consists of the customer's service time and the amount of time that the customer waits for the $M-1$ other customers that might be either waiting or in service at the same processor. The first term, $\frac{R_{pe}}{R} - \frac{\tau}{R}$, is the estimated fraction who are waiting, and $\frac{\tau}{R}$ is the estimated fraction in service.

$R_{pe_{MB}} = \tau_{MB}[1 + (M-1)(\frac{R_{pe_{MB}}}{R_{MB}} - \frac{\tau_{MB}}{R_{MB}})] + (M-1)(\frac{\tau_{MB}}{R_{MB}})\tau_{MB_{residual}}$

$R_{pe_{SB}} = \tau_{SB}[1 + (M-1)(\frac{R_{pe_{SB}}}{R_{SB}} - \frac{\tau_{SB}}{R_{SB}})] + (M-1)(\frac{\tau_{SB}}{R_{SB}})\tau_{SB_{residual}}$

$\tau_{SB_{residual}}$ and $\tau_{MB_{residual}}$ represent the residual life of the customer being served at the processor when an arriving customer arrives. As explained in [8], the standard equation for residual life under the assumption of Poisson arrivals is not accurate since arrivals at the processor are not at random points in time;[2] therefore, we use an interpolation suggested by Derek Eager [3].

For the case that $M = M_{hw}$, all processor stalls can be attributed to full MSHRs. In this case, we solve a modified MB model in which there are M_{hw} customers per processor and these customers represent the behavior of all read, write and

[1] The reads in the MB submodel have only a small effect on estimated throughput (less than 4% reduction in throughput for all applications validated in [8], except FFTopt which has a 10% reduction), and they are not discussed in [8].

[2] The estimated mean residual life for random arrivals equals the second moment of service time divided by 2τ [7].

upgrade memory system transactions. For any of these memory requests, the customer leaves the processor and visits the appropriate memory system resources.

Once throughput is computed from the weighted average of the value at each M, synchronization effects are accounted for as described in [8].

4 The Customized MVA Equations

As explained above, the SB and MB submodels use a set of customized MVA (CMVA) equations to compute the mean delay for each type of transaction at the local and remote memory buses, local and remote directories (and associated memory modules), and network interfaces. Fixed delays are assumed at resources that have negligible contention (e.g., cache tag checks, coherence packet generation), and for the approximate delay at each network switch (observed across several applications).

The equations listed in this section, along with the equations for R and R_{pe} that were presented in the previous section, completely define both submodels. The equations in this section are the same for both submodels, and they can easily be modified to model different memory system architectures, as was done in [8].

To make the equations more readable, we have adopted subscripts and superscripts to denote the possible variations in the term to which they are attached. The resource is always the first subscript on a term, whether it is residence time (R), waiting time (W), utilization (U), or service time (S). For example, R_{NI} is the residence time at the network interface. For the NI, there can be an additional keyword (q) appended to the NI to indicate that this term can vary depending on whether the action is at the output queue (q = out) or the input queue (q = in) of the NI. For many terms, there is a subscript of loc or rem to indicate whether the action is at the local node or a remote node. The variable y denotes the transaction type (see Table 3). The variable x denotes the type of message (request or data) that is on the bus or at the NI. Lastly, a superscript variable z denotes either synchronous or asynchronous.

For example, $U^z_{NIq_{loc,y,x}}$ is the utilization of queue q (out or in) at the local NI by a transaction of type y. The x and z denote whether the packet is a request packet or a data packet and whether the transaction is synchronous or asynchronous. It is important to note that a synchronous transaction can have an asynchronous part (e.g., an acknowledgment message to the home node that occurs in a 3-hop cache to cache read request). For example, $R^{asynch}_{NIin_{rem,RHLCC,x}}$ refers to the response time at the input queue of the remote NI for the asynchronous request or data message from the synchronous transaction RHLCC.

Latencies at Resources with Negligible Contention

$$Z^{synch} = (P_{LC} + P_{RC})(S_{tag}) + (P_{LHCC} + P_{RHLCC} + P_{RHRCC})(S_{tag} + S_{coherence})$$

$$Z^{asynch} = (P_{LCINV} + P_{RCINV} + P_{LUPG} + P_{RUPG} + P_{LWB} + P_{RWB})(S_{tag})$$
$$+(P_{LHCCwr} + P_{RHLCCwr} + P_{RHRCCwr})(S_{tag} + S_{coherence})$$

Network

Note that S_{switch} is the measured average per-switch delay in the network, measured across several applications in a given class of applications. S_{switch} could also be estimated by a more detailed MVA model of the interconnection network.

$$S_{NET} = HS_{switch}$$

$$R^{synch}_{NET} = \sum_y R^{synch}_{NET,y}$$

$$R_{NET}^{asynch} = \sum_y R_{NET,y}^{asynch}$$

$$R_{NET,y}^{synch} = \begin{cases} P_y 2S_{NET} & \text{y=RC,LHCC,RHLCC} \\ P_y 3S_{NET} & \text{y=RHRCC} \end{cases}$$

$$R_{NET,y}^{asynch} = \begin{cases} P_y S_{NET} & \text{y=RWB} \\ P_y 2S_{NET} & \text{y=RCINV,LHCCwr,RHLCCwr,LUPG} \\ P_y 3S_{NET} & \text{y=RHRCCwr} \\ P_y 4S_{NET} & \text{y=RUPG} \end{cases}$$

Network Interface

Below are the visit count equations for the NI. For example, $V_{NIout_{loc,y,r}}^{synch}$ is the visit count at the output queue of the local NI of request messages associated with the synchronous part of a transaction of type y.

$$V_{NIout_{loc,y,r}}^{synch} = 1 \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$V_{NIin_{loc,y,d}}^{synch} = 1 \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$V_{NIout_{loc,y,r}}^{asynch} = \begin{cases} 1 & \text{y=RCINV,RUPG,LHCCwr,RHLCCwr,RHRCCwr} \\ X & \text{y=LCINV,LUPG} \end{cases}$$

$$V_{NIout_{loc,y,d}}^{asynch} = 1 \quad \text{y=RWB}$$

$$V_{NIin_{loc,y,r}}^{asynch} = \begin{cases} X & \text{y=LCINV,LUPG} \\ 1 & \text{y=RUPG} \\ 1 & \text{y=LHCCwr} \end{cases}$$

$$V_{NIin_{loc,y,d}}^{asynch} = 1 \quad \text{y=RCINV,LHCC,LHCCwr,RHLCCwr,RHRCCwr}$$

$$V_{NIout_{rem,y,r}}^{synch} = (\tfrac{1}{N-1}) \quad \text{y=RHRCC}$$

$$V_{NIout_{rem,y,d}}^{synch} = (\tfrac{1}{N-1}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$V_{NIin_{rem,y,r}}^{synch} = (\tfrac{1}{N-1}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$V_{NIin_{rem,y,d}}^{synch} = (\tfrac{1}{N-1}) \quad \text{y=RHRCC}$$

$$V_{NIout_{rem,y,r}}^{asynch} = \begin{cases} (\frac{X}{N-1}) & \text{y=LCINV,LUPG} \\ (\frac{2X}{N-1}) & \text{y=RCINV} \\ (\frac{2X+1}{N-1}) & \text{y=RUPG} \\ (\frac{1}{N-1}) & \text{y=LHCCwr,RHLCCwr} \\ (\frac{2}{N-1}) & \text{y=RHRCCwr} \end{cases}$$

$$V_{NIout_{rem,y,d}}^{asynch} = (\tfrac{1}{N-1}) \quad \text{y=RCINV,LHCC,RHLCC,RHRCC,LHCCwr,RHLCCwr,RHRCCwr}$$

$$V_{NIin_{rem,y,r}}^{asynch} = \begin{cases} (\frac{X}{N-1}) & \text{y=LCINV,LUPG} \\ (\frac{2X+1}{N-1}) & \text{y=RCINV,RUPG} \\ (\frac{1}{N-1}) & \text{y=LHCCwr} \\ (\frac{2}{N-1}) & \text{y=RHLCCwr} \\ (\frac{3}{N-1}) & \text{y=RHRCCwr} \end{cases}$$

$$V_{NIin_{rem,y,d}}^{asynch} = (\tfrac{1}{N-1}) \quad \text{y=RWB,RHLCC,RHRCC}$$

The residence time at the NI is composed of the response times at the output queue and the input queue.

$$R_{NI}^{synch} = R_{NIout}^{synch} + R_{NIin}^{synch}$$

$$R_{NI}^{asynch} = R_{NIout}^{asynch} + R_{NIin}^{asynch}$$

The residence time at the output (input) queue is the sum over all transaction types y of the residence times at the local and remote output (input) queues. For each transaction, we include both types of messages (i.e., request and data) that are generated by the synchronous part of the transaction.

$$R_{NIout}^{z} = \sum_y (R_{NIout_{loc,y,r}}^{z} + R_{NIout_{loc,y,d}}^{z}) + (N-1) \sum_y (R_{NIout_{rem,y,r}}^{z} + R_{NIout_{rem,y,d}}^{z})$$

$$R_{NIin}^{z} = \sum_y (R_{NIin_{loc,y,r}}^{z} + R_{NIin_{loc,y,d}}^{z}) + (N-1) \sum_y (R_{NIin_{rem,y,r}}^{z} + R_{NIin_{rem,y,d}}^{z})$$

The next set of equations describe the residence times of the different types of messages at the input and output queues of the NI. For example, $R_{NIout_{loc,y,x}}^{synch}$ is the residence time at the output queue of the local NI of a message of type x that is associated with the synchronous part of a transaction of type y. This time equals the probability of transaction y times the visit count at this queue times the sum of the per-visit waiting time and the service time that this type x message will experience.

$$R_{NIout_{loc,y,x}}^{synch} = P_y V_{NIout_{loc,y,x}}^{synch} (W_{NIout_{loc}} + S_{NIout,x}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$R_{NIin_{loc,y,x}}^{synch} = P_y V_{NIin_{loc,y,x}}^{synch} (W_{NIin_{loc}} + S_{NIin,x}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$R_{NIout_{loc,y,x}}^{asynch} = P_y V_{NIout_{loc,y,x}}^{asynch} (W_{NIout_{loc}} + S_{NIout,x}) \quad \text{y=RWB,RCINV,RUPG,LCINV,LUPG, LHCCwr,RHLCCwr,RHRCCwr}$$

$$R_{NIin_{loc,y,x}}^{asynch} = P_y V_{NIin_{loc,y,x}}^{asynch} (W_{NIin_{loc}} + S_{NIin,x}) \quad \text{y=RCINV,RUPG,LHCC,LCINV,LUPG, LHCCwr,RHLCCwr,RHRCCwr}$$

$$R_{NIout_{rem,y,x}}^{synch} = P_y V_{NIout_{rem,y,x}}^{synch} (W_{NIout_{rem}} + S_{NIout,x}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$R_{NIin_{rem,y,x}}^{synch} = P_y V_{NIin_{rem,y,x}}^{synch} (W_{NIin_{rem}} + S_{NIin,x}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$R_{NIout_{rem,y,x}}^{asynch} = P_y V_{NIout_{rem,y,x}}^{asynch} (W_{NIout_{rem}} + S_{NIout,x}) \quad \text{y=LHCC,RHLCC,RHRCC,LHCCwr,RHLCCwr, RHRCCwr,LCINV,LUPG,RCINV,RUPG}$$

$$R_{NIin_{rem,y,x}}^{asynch} = P_y V_{NIin_{rem,y,x}}^{asynch} (W_{NIin_{rem}} + S_{NIin,x}) \quad \text{y=RWB,RHLCC,RHRCC,LHCCwr,RHLCCwr, RHRCCwr,LCINV,LUPG,RCINV,RUPG}$$

The following equations describe the utilization of the queues of the NI. The notation is similar to that for the residence times of the NI.

$$U_{NIq_{loc,x}} = \sum_y U_{NIq_{loc,y,x}}^{synch} + \sum_y U_{NIq_{loc,y,x}}^{asynch}$$

$$U_{NIq_{rem,x}} = \sum_y U_{NIq_{rem,y,x}}^{synch} + \sum_y U_{NIq_{rem,y,x}}^{asynch}$$

The utilization of resource q of the local (remote) NI by messages of type x during a transaction of type y is equal to the throughput of type y transactions ($\frac{P_y}{R}$) multiplied by the average number of type x messages per type y transaction that visit this resource ($V_{NIq_{loc/rem,y,x}}^{z}$), multiplied by the service time of these messages. The z distinguishes between synchronous and asynchronous transactions.

$$U_{NIq_{loc,y,x}}^{z} = (\frac{P_y}{R}) V_{NIq_{loc,y,x}}^{z} S_{NIq,x}$$

$$U^z_{NIq_{rem,y,x}} = (\frac{P_y}{R})V^z_{NIq_{rem,y,x}}S_{NIq,x}$$

The next equations are used to calculate the waiting time of a customer who arrives at one of the NI's queues. $W_{NIq_{loc}}$ is the waiting time at queue q of the local NI. It is the sum of the waiting time due to locally generated messages ($W^{loc}_{NIq_{loc}}$) and remotely generated messages ($W^{rem}_{NIq_{loc}}$). The subscript is the resource where the waiting occurs, and the superscript is for the messages that cause the waiting. For $W_{NIq_{rem}}$, the waiting time at queue q of the remote NI is due to "others" (other processors - the processor that generated the arriving request and all other processors except the one that is local to the given remote NI) and "remote" (the processor at the given remote NI).

$$W_{NIq_{loc}} = W^{loc}_{NIq_{loc}} + W^{rem}_{NIq_{loc}}$$

$$W_{NIq_{rem}} = W^{others}_{NIq_{rem}} + W^{rem}_{NIq_{rem}}$$

$W^{loc}_{NIq_{loc}}$ is the waiting time at queue q of the local NI due to locally generated messages. It is composed of the waiting times at this queue due to locally generated messages from the synchronous part and the asynchronous part of all transactions.

$$W^{loc}_{NIq_{loc}} = \sum_y (W^{loc,y^S}_{NIq_{loc}} + W^{loc,y^A}_{NIq_{loc}})$$

$$W^{rem}_{NIq_{loc}} = \sum_y (W^{rem,y^S}_{NIq_{loc}} + W^{rem,y^A}_{NIq_{loc}})$$

$$W^{others}_{NIq_{rem}} = \sum_y (W^{others,y^S}_{NIq_{rem}} + W^{others,y^A}_{NIq_{rem}})$$

$$W^{rem}_{NIq_{rem}} = \sum_y (W^{rem,y^S}_{NIq_{rem}} + W^{rem,y^A}_{NIq_{rem}})$$

The next equation is used to calculate the waiting time at queue q of the local NI due to the synchronous part of the transactions of type y generated by the local processor. Breaking the equation down, we note that $(\frac{R^S_{NIq_{loc,y,x}}}{R} - U^S_{NIq_{loc,y,x}})$ is the average number of customers in the queue minus the number of customers being served (i.e., the utilization, a number between zero and one). We have to wait for all of these customers to complete, so this queue length is multiplied by the service time. For the customer in service, we wait for its residual life in service, which is equal to half the service time for a deterministic service time. Without the summation and factor of $(M-1)$, we have the traffic due to only one customer's messages of type x. Therefore, we sum over the message types, and we multiply by a factor of $(M-1)$ because an arriving customer waits for the traffic of the $(M-1)$ other customers of the processor local to the NI. The other waiting time equations are similar to the one described.

$$W^{loc,y^S}_{NIq_{loc}} = \sum_x (M-1)[(\frac{R^S_{NIq_{loc,y,x}}}{R} - U^S_{NIq_{loc,y,x}})S_{NIq,x} + (U^S_{NIq_{loc,y,x}})(\frac{S_{NIq,x}}{2})]$$

The next equation has a factor of M, instead of $M-1$, since a given local customer can wait for the asynchronous requests generated by all M customers of the local processor (i.e., including the arriving customer). Note that $\frac{1}{R}P_y$ is the rate at which an asynchronous transaction y is forked. Multiplying this throughput by the residence time at a queue gives the mean queue length for an asynchronous transaction. Thus, the form of these customized equations is very similar for synchronous and asynchronous transactions.

$$W^{loc,y^A}_{NIq_{loc}} = \sum_x M[(\frac{R^A_{NIq_{loc,y,x}}}{R} - U^A_{NIq_{loc,y,x}})S_{NIq,x} + (U^A_{NIq_{loc,y,x}})(\frac{S_{NIq,x}}{2})]$$

$$W^{rem,y^z}_{NIq_{loc}} = \sum_x (N-1)M[(\frac{R^z_{NIq_{rem,y,x}}}{R} - U^z_{NIq_{rem,y,x}})S_{NIq,x} + (U^z_{NIq_{rem,y,x}})(\frac{S_{NIq,x}}{2})]$$

$$W^{others,y^S}_{NIq_{rem}} = \sum_x [(M-1) + M(N-2)][(\frac{R^S_{NIq_{rem,y,x}}}{R} - U^S_{NIq_{rem,y,x}})S_{NIq,x} + (U^S_{NIq_{rem,y,x}})(\frac{S_{NIq,x}}{2})]$$

$$W^{others,y^A}_{NIq_{rem}} = \sum_x [M + M(N-2)][(\frac{R^A_{NIq_{rem,y,x}}}{R} - U^A_{NIq_{rem,y,x}})S_{NIq,x} + (U^A_{NIq_{rem,y,x}})(\frac{S_{NIq,x}}{2})]$$

$$W^{rem,y^z}_{NIq_{rem}} = \sum_x M[(\frac{R^z_{NIq_{loc,y,x}}}{R} - U^z_{NIq_{loc,y,x}})S_{NIq,x} + (U^z_{NIq_{loc,y,x}})(\frac{S_{NIq,x}}{2})]$$

Bus

The notation and equations for the bus and the other memory subsystem resources are very similar to those of the NI. These equations are given without further explanation.

$$S^{synch}_{bus_{loc,y,r}} = S_{bus,r} \quad \text{y=LC,RC,LHCC,RHLCC,RHRCC}$$

$$S^{synch}_{bus_{loc,y,d}} = S_{bus,d} \quad \text{y=LC,RC,LHCC,RHLCC,RHRCC}$$

$$S^{asynch}_{bus_{loc,y,r}} = S_{bus,r} \quad \text{y=LHCC,RHLCC,RHRCC,LUPG,LCINV,RUPG,RCINV,LHCCwr,RHLCCwr,RHRCCwr}$$

$$S^{asynch}_{bus_{loc,y,d}} = S_{bus,d} \quad \text{y=LHCC,RHLCC,RHRCC,LCINV,RCINV,LWB,RWB,LHCCwr,RHLCCwr,RHRCCwr}$$

$$S^{synch}_{bus_{rem,y,r}} = S_{bus,r} \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$S^{synch}_{bus_{rem,y,d}} = S_{bus,d} \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$S^{asynch}_{bus_{rem,y,r}} = S_{bus,r} \quad \text{y=RHLCC,RHRCC,LCINV,LUPG,RCINV,RUPG,LHCCwr,RHLCCwr,RHRCCwr}$$

$$S^{asynch}_{bus_{rem,y,d}} = S_{bus,d} \quad \text{y=RHLCC,RHRCC,RWB,RCINV,LHCCwr,RHLCCwr,RHRCCwr}$$

$$V^{synch}_{bus_{loc,y,r}} = \begin{cases} 1 & \text{y=LC,RC,RHLCC,RHRCC} \\ 2 & \text{y=LHCC} \end{cases}$$

$$V^{synch}_{bus_{loc,y,d}} = 1 \quad \text{y=LC,RC,LHCC,RHLCC,RHRCC}$$

$$V^{asynch}_{bus_{loc,y,r}} = \begin{cases} 1 & \text{y=RCINV,RHLCCwr,RHRCCwr} \\ 2 & \text{y=RUPG} \\ 3 & \text{y=LHCCwr} \\ 2X+2 & \text{y=LUPG} \\ 2X+1 & \text{y=LCINV} \end{cases}$$

$$V^{asynch}_{bus_{loc,y,d}} = \{\ 1 \quad \text{y=LCINV,RCINV,LWB,RWB,LHCC,LHCCwr,RHLCCwr,RHRCCwr}$$

$$V^{synch}_{bus_{rem,y,r}} = \begin{cases} (\frac{1}{N-1}) & \text{y=RC,LHCC} \\ (\frac{2}{N-1}) & \text{y=RHLCC} \\ (\frac{3}{N-1}) & \text{y=RHRCC} \end{cases}$$

$$V^{synch}_{bus_{rem,y,d}} = (\tfrac{1}{N-1}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$V^{asynch}_{bus_{rem,y,r}} = \begin{cases} (\frac{2}{N-1}) & \text{y=LHCCwr} \\ (\frac{4}{N-1}) & \text{y=RHLCCwr} \\ (\frac{5}{N-1}) & \text{y=RHRCCwr} \\ (\frac{4X+1}{N-1}) & \text{y=RCINV} \\ (\frac{4X+2}{N-1}) & \text{y=RUPG} \\ (\frac{2X}{N-1}) & \text{y=LCINV,LUPG} \end{cases}$$

$$V^{asynch}_{bus_{rem,y,d}} = \begin{cases} (\frac{1}{N-1}) & \text{y=RWB,RCINV,LHCC,LHCCwr,RHLCCwr,RHRCCwr} \\ (\frac{2}{N-1}) & \text{y=RHLCC,RHRCC} \end{cases}$$

$$R^{synch}_{bus} = \textstyle\sum_y (R^{synch}_{bus_{loc,y,r}} + R^{synch}_{bus_{loc,y,d}}) + (N-1) \sum_y (R^{synch}_{bus_{rem,y,r}} + R^{synch}_{bus_{rem,y,d}})$$

$$R^{asynch}_{bus} = \textstyle\sum_y (R^{asynch}_{bus_{loc,y,r}} + R^{asynch}_{bus_{loc,y,d}}) + (N-1) \sum_y (R^{asynch}_{bus_{rem,y,r}} + R^{asynch}_{bus_{rem,y,d}})$$

$$R^{synch}_{bus_{loc,y,x}} = P_y V^{synch}_{bus_{loc,y,x}} (W_{bus_{loc}} + S^{synch}_{bus_{loc,y,x}}) \quad \text{y=LC,RC,LHCC,RHLCC,RHRCC}$$

$$R^{asynch}_{bus_{loc,y,x}} = P_y V^{asynch}_{bus_{loc,y,x}} (W_{bus_{loc}} + S^{asynch}_{bus_{loc,y,x}}) \quad \text{y=LUPG,LCINV,LWB,RWB,RCINV,RUPG,LHCC, RHLCC,RHRCC,LHCCwr,RHLCCwr,RHRCCwr}$$

$$R^{synch}_{bus_{rem,y,x}} = P_y V^{synch}_{bus_{rem,y,x}} (W_{bus_{rem}} + S^{synch}_{bus_{rem,y,x}}) \quad \text{y=RC,LHCC,RHLCC,RHRCC}$$

$$R^{asynch}_{bus_{rem,y,x}} = P_y V^{asynch}_{bus_{rem,y,x}} (W_{bus_{rem}} + S^{asynch}_{bus_{rem,y,x}}) \quad \text{y=RWB}$$

$$U_{bus_{loc,x}} = \sum_y U^{synch}_{bus_{loc,y,x}} + \sum_y U^{asynch}_{bus_{loc,y,x}}$$

$$U_{bus_{rem,x}} = \sum_y U^{synch}_{bus_{rem,y,x}} + \sum_y U^{asynch}_{bus_{rem,y,x}}$$

$$U^z_{bus_{loc,y,x}} = (\frac{P_y}{R}) V^z_{bus_{loc,y,x}} S^z_{bus_{loc,y,x}}$$

$$U^z_{bus_{rem,y,x}} = (\frac{P_y}{R}) V^z_{bus_{rem,y,x}} S^z_{bus_{rem,y,x}}$$

$$W_{bus_{loc}} = \sum_y (W^{loc,y^S}_{bus_{loc}} + W^{loc,y^A}_{bus_{loc}} + W^{rem,y^S}_{bus_{loc}} + W^{rem,y^A}_{bus+loc})$$

$$W_{bus_{rem}} = \sum_y (W^{others,y^S}_{bus_{rem}} + W^{others,y^A}_{bus_{rem}} + W^{rem,y^S}_{bus_{rem}} + W^{rem,y^A}_{bus+rem})$$

$$W^{loc,y^S}_{bus_{loc}} = \sum_x (M-1)[(\frac{R^S_{bus_{loc,y,x}}}{R} - U^S_{bus_{loc,y,x}}) S^S_{bus_{loc,y,x}} + (U^S_{bus_{loc,y,x}})(\frac{S^S_{bus_{loc,y,x}}}{2})]$$

$$W^{loc,y^A}_{bus_{loc}} = \sum_x M[(\frac{R^A_{bus_{loc,y,x}}}{R} - U^A_{bus_{loc,y,x}}) S^A_{bus_{loc,y,x}} + (U^A_{bus_{loc,y,x}})(\frac{S^A_{bus_{loc,y,x}}}{2})]$$

$$W^{rem,y^z}_{bus_{loc}} = \sum_x (N-1) M[(\frac{R^z_{bus_{rem,y,x}}}{R} - U^z_{bus_{rem,y,x}}) S^z_{bus_{rem,y,x}} + (U^z_{bus_{rem,y,x}})(\frac{S^z_{bus_{rem,y,x}}}{2})]$$

$$W^{others,y^S}_{bus_{rem}} = \sum_x [(M-1) + M(N-2)][(\frac{R^S_{bus_{rem,y,x}}}{R} - U^S_{bus_{rem,y,x}}) S^S_{bus_{rem,y,x}} + (U^S_{bus_{rem,y,x}})(\frac{S^S_{bus_{rem,y,x}}}{2})]$$

$$W^{others,y^A}_{bus_{rem}} = \sum_x [M + M(N-2)][(\frac{R^A_{bus_{rem,y,x}}}{R} - U^A_{bus_{rem,y,x}}) S^A_{bus_{rem,y,x}} + (U^A_{bus_{rem,y,x}})(\frac{S^A_{bus_{rem,y,x}}}{2})]$$

$$W^{rem,y^z}_{bus_{rem}} = \sum_x M[(\frac{R^z_{bus_{loc,y,x}}}{R} - U^z_{bus_{loc,y,x}}) S^z_{bus_{loc,y,x}} + (U^z_{bus_{loc,y,x}})(\frac{S^z_{bus_{loc,y,x}}}{2})]$$

Memory

If a decoupled directory is desired, the following equations can also be used for the directory, with the appropriate changes to the visit counts.

$$S^{synch}_{mem_{loc,y}} = S_{DRAM} \quad \text{y=LC}$$

$$S^{asynch}_{mem_{loc,y}} = S_{DRAM} \quad \text{y=LCINV,LWB,LHCC}$$

$$S^{synch}_{mem_{rem,y}} = S_{DRAM} \quad \text{y=RC}$$

$$S^{asynch}_{mem_{rem,y}} = S_{DRAM} \quad \text{y=RWB,RCINV,RHLCC,RHRCC}$$

$$V^{synch}_{mem_{loc,y}} = \frac{1}{m} \quad \text{y=LC,LHCC}$$

$V^{asynch}_{mem_{loc,y}} = \frac{1}{m}$ y=LCINV,LUPG,LWB,LHCC,LHCCwr

$V^{synch}_{mem_{rem,y}} = \frac{1}{(N-1)m}$ y=RC,RHLCC,RHRCC

$V^{asynch}_{mem_{rem,y}} = \frac{1}{(N-1)m}$ y=RCINV,RUPG,RWB,RHLCC,RHRCC,RHLCCwr,RHRCCwr

$R^{synch}_{mem} = mR^{synch}_{mem_{loc}} + (N-1)mR^{synch}_{mem_{rem}}$

$R^{asynch}_{mem} = mR^{asynch}_{mem_{loc}} + (N-1)mR^{asynch}_{mem_{rem}}$

$R^{synch}_{mem_{loc}} = \sum_y R^{synch}_{mem_{loc,y}}$ y=LC

$R^{asynch}_{mem_{loc}} = \sum_y R^{asynch}_{mem_{loc,y}}$ y=LCINV,LWB,LHCC

$R^{synch}_{mem_{rem}} = \sum_y R^{synch}_{mem_{rem,y}}$ y=RC

$R^{asynch}_{mem_{rem}} = \sum_y R^{asynch}_{mem_{rem,y}}$ y=RCINV,RWB,RHLCC,RHRCC

$R^{z}_{mem_{loc,y}} = P_y V^{z}_{mem_{loc,y}}(W_{mem_{loc}} + S^{z}_{mem_{loc,y}})$ y=LC,LCINV,LWB,LHCC

$R^{z}_{mem_{rem,y}} = P_y V^{z}_{mem_{rem,y}}(W_{mem_{loc}} + S^{z}_{mem_{loc,y}})$ y=RC,RCINV,RWB,RHLCC,RHRCC

$U^{z}_{mem_{loc,y}} = \frac{P_y}{R}(V^{z}_{mem_{loc,y}} S^{z}_{mem_{loc,y}})$

$U^{z}_{mem_{rem,y}} = \frac{P_y}{R}(V^{z}_{mem_{rem,y}} S^{z}_{mem_{rem,y}})$

$W_{mem_{loc}} = W^{loc}_{mem_{loc}} + W^{rem}_{mem_{loc}}$

$W_{mem_{rem}} = W^{others}_{mem_{rem}} + W^{rem}_{mem_{rem}}$

$W^{loc}_{mem_{loc}} = \sum_y (W^{loc,y^S}_{mem_{loc}} + W^{loc,y^A}_{mem_{loc}})$

$W^{rem}_{mem_{loc}} = \sum_y (W^{rem,y^S}_{mem_{loc}} + W^{rem,y^A}_{mem_{loc}})$

$W^{others}_{mem_{rem}} = \sum_y (W^{others,y^S}_{mem_{rem}} + W^{others,y^A}_{mem_{rem}})$

$W^{rem}_{mem_{rem}} = \sum_y (W^{rem,y^S}_{mem_{rem}} + W^{rem,y^A}_{mem_{rem}})$

$W^{loc,y^S}_{mem_{loc}} = (M-1)[(\frac{R_{mem_{loc,y^S}}}{R} - U_{mem_{loc,y^S}})S_{mem_{loc,y^S}} + (U_{mem_{loc,y^S}})(\frac{S_{mem_{loc,y^S}}}{2})]$

$W^{loc,y^A}_{mem_{loc}} = M[(\frac{R_{mem_{loc,y^A}}}{R} - U_{mem_{loc,y^A}})S_{mem_{loc,y^A}} + (U_{mem_{loc,y^A}})(\frac{S_{mem_{loc,y^A}}}{2})]$

$W^{rem,y^z}_{mem_{loc}} = (N-1)M[(\frac{R_{mem_{rem,y^z}}}{R} - U_{mem_{rem,y^z}})S_{mem_{rem,y^z}} + (U_{mem_{rem,y^z}})(\frac{S_{mem_{rem,y^z}}}{2})]$

$W^{others,y^S}_{mem_{rem}} = [(M-1) + M(N-2)][(\frac{R_{mem_{rem,y^S}}}{R} - U_{mem_{rem,y^S}})S_{mem_{rem,y^S}} + (U_{mem_{rem,y^S}})(\frac{S_{mem_{rem,y^S}}}{2})]$

$W^{others,y^A}_{mem_{rem}} = [M + M(N-2)][(\frac{R_{mem_{rem,y^A}}}{R} - U_{mem_{rem,y^A}})S_{mem_{rem,y^A}} + (U_{mem_{rem,y^A}})(\frac{S_{mem_{rem,y^A}}}{2})]$

$W^{rem,y^z}_{mem_{rem}} = M[(\frac{R_{mem_{loc,y^z}}}{R} - U_{mem_{loc,y^z}})S_{mem_{loc,y^z}} + (U_{mem_{loc,y^z}})(\frac{S_{mem_{loc,y^z}}}{2})]$

References

[1] S. Adve, V. Adve, M. Hill, and M. Vernon. Comparison of Hardware and Software Cache Coherence Schemes. In *Proc. 18th Int'l Symp. on Computer Architecture*, pages 298–308, June 1991.

[2] V. Adve and M. Vernon. The Influence of Random Delays on Parallel Task Execution Times. In *Proc. ACM SIGMETRICS*, pages 61–73, May 1993.

[3] D. Eager. Private communication, Nov. 1997.

[4] P. Heidelberger and K. Trivedi. Analytic Queueing Models for Programs with Internal Concurrency. *IEEE Trans. on Computers*, C-32(1):73–82, Jan. 1982.

[5] P. Heidelberger and K. Trivedi. Queueing Network Models for Parallel Processing with Asynchronous Tasks. *IEEE Trans. on Computers*, C-31(11):1099–1109, Nov. 1982.

[6] P. Jacobson and E. Lazowska. Analyzing Queueing Networks with Simultaneous Resource Possession. *Communications of the ACM*, 25(2):142–151, Feb. 1982.

[7] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik. *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Englewood Cliffs, NJ, May 1984.

[8] D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood. Analytic Evaluation of Shared-Memory Parallel Systems with ILP Processors. In *Proc. 25th Int'l Symp. on Computer Architecture*, June 1998.

[9] M. Vernon, E. Lazowska, and J. Zahorjan. An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols. In *Proc. 15th Int'l Symp. on Computer Architecture*, pages 192–202, 1988.