Performance Modelling and Evaluation for the XMP Shared-Bus Multiprocessor Architecture

Chiung-San Lee and Tai-Ming Parng
Dept. of Electrical Engineering
National Taiwan University, R.O.C.

Jew-Chin Lee, Cheng-Nan Tsai, Kwo-Jean Farn
Lung-Chung Chang, Te-Yu Chung, and Lu-Ping Chen
Industrial Technology Research Institude
Hsinchu, Taiwan, R.O.C.

Abstract

This paper presents the performance modelling and evaluation of a shared bus multiprocessor, *XMP*. A key characteristic of XMP is that it employs a special shared bus scheme featuring separate address bus and data bus with split transaction, pipelined cycle (called SSTP scheme). To assist evaluating the architectural alternatives of XMP, the features of the SSTP bus scheme as well as two important performance impacting factors: (1) cache, bus, and memory interferences and (2) DMA transfer, are modelled. We employ a Subsystem Access Time (SAT) modelling methodology. It is based on a Subsystem Access Time Per Instruction (SATPI) concept, in which we treat major components other than processors (e.g. off-chip cache, bus, memory, I/O) as subsystems and model for each of them the mean access time per instruction from each processor. Validated by statistical simulations, the performance model is fed with a given set of representative workload parameters, and then used to conduct performance evaluation for some initial system design issues. Furthermore, the SATPIs of the subsystems are directly utilized to identify the bottleneck subsystems and to help analyze the cause of the bottleneck.

1: Introduction

A shared-bus multiprocessor system is one of important class of computer architecture. Such a system consists of multiple *bus masters*, such as processors and DMA controllers, and *subsystems*, such as shared bus, main memory, and I/O. The design of an multiprocessor system is complex, and prototyping the system is expensive. Therefore, multiprocessor systems present an important opportunity for performance models to contribute to the actual design process.

XMP is a new shared-bus multiprocessor system [9]. It supports a special shared bus scheme which features separated address-bus (Abus) and data-bus (Dbus) with split transaction, pipelined (called SSTP bus scheme) cycle [8, 9]. Here, "separate" means the Abus and Dbus can be controlled independently and used by two different transactions at the same time. In other words, the Abus is used by the master to send the address and the Dbus is used by the subsystem to transmit the returned data. The XMP system has been developed in ITRI.

This paper presents a performance model that we have used to study the feasibility and several issues for the XMP architecture design. The issues that must be considered in the preliminary design of the XMP system include the optimal number of processors connected to a shared bus, number of memory modules, memory latency, and so on. To obtain accurate prediction results, the performance model takes two important performance impacting factors into accounts: (1) the cache, bus, memory interference, (2) DMA transfer.

The performance model we develop for XMP is based on the subsystem access time (SAT) modelling methodology [11]. The SAT modelling approach takes a subsystem oriented point of view and models the mean access time of each subsystem for the execution of each processor instruction (called Subsystem Access Time Per Instruction, SATPI). The SATPIs of all subsystems in XMP, in addition to being used for predicting performance, can also be used for identifying which subsystem is the bottleneck and then exploring different configurations of the subsystem for potential performance improvements.

The rest of this paper is organized as follows. In Section 2, we introduce the XMP architecture of interest and its bus requests. Section 3 presents the performance model based on SAT modelling. In Section 4, given a set of workload parameters, we utilize the SAT models to predict the system performance for the XMP architecture design and identify the bottleneck for potential performance improvement. Finally, the conclusion is given in Section 5.

2: The XMP Architecture

The XMP system consists of several processors, main memory subsystems, and I/O subsystems connected together by a shared bus based on a separated address bus

0-8186-6555-6/94 $04.00 © 1994 IEEE

and a data bus (Fig. 1). Each processor has a local memory (private cache) from which it normally reads and to which it normally writes. However, if the private cache does not contain the necessary data item, a request to the shared bus occurs. The request, which may be a memory read or write, is queued until the shared bus is ready. In such a system with multiple caches, the same information can be shared and may reside in a number of copies in the main memory and in some of the caches. To keep different copies of the shared data consistent, we assume that the cache coherence is maintained through the MESI protocol [5], and the private cache is of write-back type [5, 8]. In addition, the processor executes I/O instructions to obtain I/O status or to program a DMA (Directed Memory Access) controller for data transfer between main memory and an I/O device.

The XMP architecture adopts a two-level cache structure, a very popular architecture for reducing bus traffic [8, 13, 15]. We assume that each processor is equipped with an on-chip cache to reduce the off-chip requests. In this paper the on-chip cache is referred to as the L1 cache, while the off-chip one as the L2 cache.

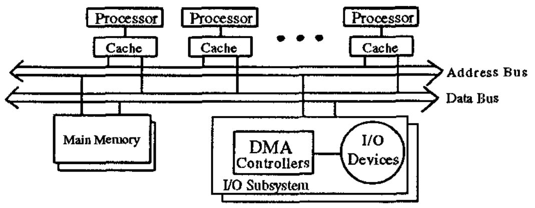

Fig. 1. The XMP architecture with a separate address bus and data bus

The DMA controller is usually started by using I/O instructions and the following data transfers (referred to as DMA-transfer request) can proceed without being interventeded by the processors. It is reasonable to assume that the processor execution of an application process is completely overlapped with the DMA transfers of another application process. Therefore, the DMA transfers cause bus and memory interferences to processor's requests. For modelling the DMA transfers, we use a parameter, *the number of bits of DMA transfer per instruction*, to characterize the effect of DMA transfers.

In this study, we consider a special shared bus scheme which features separate address-bus (Abus) and data-bus (Dbus) (i.e., the address-bus and data-bus can be controlled or used independently) with split transaction support [4, 8, 9]. A split transaction, pipelined (STP) bus should support two types of bus operations: sending the address and receiving or sending the data. The split transactions are usually defined by what they do to memory: a memory read transaction transfers data from memory, and a memory write transaction writes the data to memory. The read transaction is broken into a read-request transaction that contains the address and command indicating a read, and a memory-reply transaction that contains the returned data. During the memory read split transactions, after the address is sent, the address bus is made available for other processors before the accessed memory data is ready to be returned. A memory write, which requires the processor to send both the address and data and doesn't require the returned data, is not a split transaction. With separated Abus and Dbus, the shared bus has three operating modes: Abus-only, Dbus-only, and Abus-and-Dbus. The Abus-only mode is used in some bus transactions, such as read-request transactions, that transfer only the memory address. The Dbus-only mode is used in memory-reply transactions that respond the returned data to the requester. The Abus-and-Dbus mode is used during main memory write transactions, such as a dirty cache line writeback. In order to support the bus with split transaction cycles, in the memory module an input queue (IQ) and output queue (OQ) are used to save the incoming requests and returned data. In this bus scheme, the separated Abus and Dbus may serve two different requests at the same time, and can support higher throughput than non-separated bus. An STP bus with the separated Abus and Dbus (called SSTP bus) is yet a newer high-performance bus adopted in the XMP system.

For analysis purposes, the following types of requests are included in the analytical model:

- *read-miss*, representing that the processor requests a data which is currently not in its L2 cache and must access memory via the Abus.
- *memory-reply*, representing the process that puts the requested data onto the Dbus when the memory read operation is complete.
- *invalidation*, representing that the processor writes and hits a 'shared' state line, and then its L2 cache issues a signal via the Abus to invalidate other cache line copies.

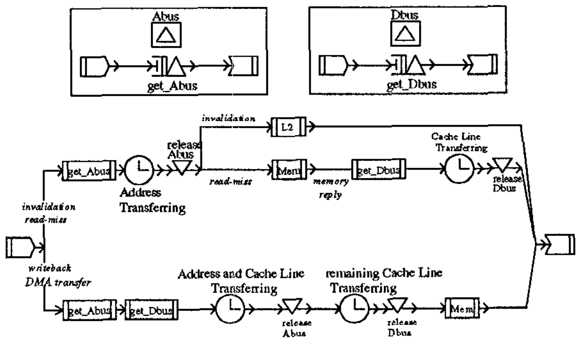

Fig. 2. XMP bus and bus requests

- *writeback*, representing that the L2 cache must write back a dirty cache line into the memory via the Abus and Dbus. The memory write operation completes asynchronously and write acknowledgment does not interfere with other bus requests.
- *DMA-transfer*, representing that the DMA controller transfers data to/from memory via the Abus and Dbus.

In Fig. 2, each 'arc' connecting two nodes represents a path along which a request may flow from one node to the other. For a request departing a node from which two or more arcs emanate, the request must choose based on certain probability one of the arcs to pass. The probability can be obtained according to the frequency distributions of processor instructions from existing literatures [1, 3, 7, 13, 15-17] for a wide variety of workloads.

3: SAT Modelling of the XMP Architecture

In this subsection, we apply the SAT modelling methodology presented in [11] for the XMP architecture with SSTP bus scheme. There are three major subsystems, namely the L2 cache, shared bus, and memory, in the target model. Performance metrics, such as response time, SATPI, average execution time per instruction, partial utilization, partial queue length, and waiting time, will be derived.

3.1: Modelling the target architecture

In the following, let H_i denote the set of all request types that will visit the *i*th subsystem and B_i, a subset of H_i, denote the set of blocking request types that will access the *i*th subsystem. We use the notation presented in Table I for the SAT modelling of the target architecture. From Table I and Fig. 2, we can see that the request types of the three subsystems include *L1_Miss*, *Miss*, *icl*, *wb*, *dma*, and *rep* requests. Two request types, *L1_Miss* and *icl*, will visit the L2 cache; the *L1_Miss* is a blocking request, because the processor waits until it receives a response for the instruction that cause the *L1_Miss* request. There are five request types that visit the shared bus; two of them, *Miss* and *icl*, will block the requesting master. Three request types, including *Miss*, *wb*, and *dma*, visit the memory subsystem; only the *Miss* request will cause the requester to be blocked. Note that the *icl* is a blocking request to the bus, but a nonblocking one to the L2 cache. This is because no data response is required for the *icl* request and the internal operation for the L2 cache to update its tag will not cause any delay to the requester.

TABLE I
Notation for SAT Modelling of the Target Architecture

Request Types:

- $H_{L2}=\{L1\_Miss, icl\}$, $H_{bus}=\{Miss, icl, dma, wb, rep\}$, $H_{mem}=\{Miss, dma, wb\}$ denote the sets of request types that visit the L2 cache, shared bus and memory. *L1_Miss, Miss, icl, dma, wb, rep* are the L1-miss, read-miss, invalidation, DMA-transfer, writeback, and memory-reply requests. The request of type *dma* includes *dmaR* (DMA read from memory) and *dmaW* (DMA write into memory).
- $B_{L2}=\{L1\_Miss\}$, $B_{bus}=\{Miss, icl\}$ and $B_{mem}=\{Miss\}$ denote the sets of blocking request types that visit the L2 cache, shared bus, and memory.

Input Parameters:

- n is total number of processors connected to the shared bus.
- t_{cpu} and t_{L2} are the internal processing time of the processor and the latency of the L2 cache.
- t_{memR} and t_{memW} are the service time of main memory for read and write requests, respectively. t_{arbi} is the shared bus, i.e., Abus or Dbus, arbitration time, when a request arrives at the bus and the bus is ready.
- t_{addr} is the time for transmitting an address on the Abus, excluding the time for Abus arbitration.
- t_{data} is the time for transmitting data of a cache line size on the Dbus, excluding the time for Dbus arbitration.
- p_{L1_Miss} and p_{L2_Miss} are the probability of L1-cache miss and L2-cache miss, respectively.
- p_{Miss} is the probability of read-miss requests from an L2 cache including requests of read-to-M line; note that $p_{Miss}=p_{L1\_Miss}p_{L2\_Miss}$.
- p_{rep} is the probability of memory-reply per instruction; note that it is equal to $p_{L1\_Miss}p_{L2\_Miss}$.
- p_{wb} is the probability that an L2-cache miss results in a writeback of a dirty cache line.
- p_{icl} is the probability that a request is an invalidate-cache-line operation.
- p_{dma} is the probability of DMA transfer per instruction. We use p_{dmaR} and p_{dmaW} to denote the probabilities that the DMA reads from and writes into the main memory, respectively. Note that $p_{dma} = p_{dmaR} + p_{dmaW}$.

The output parameters are defined in the development of the equations.

(1) Service Time: As mentioned previously, in general, some subsystems take a fixed length of time to serve an arriving request, while others take a time that depends on the request type as well as on state of the subsystem (whether it is busy or not). As a result, the service time, denoted as $S_{i,j}$, of the *i*th subsystem for a request of type *j* is either a given constant value or a function of other model parameters. Most service times of subsystems are deterministic, but some are request- or utilization-dependent. The service times of the processors and L2 cache, which are assumed to be constant, are denoted as

t_{cpu} and t_{L2}, respectively. The service time of the memory subsystem depends on the type of requests being served. The service time for a read request, denoted as t_{memR}, includes the latency of input queue, memory, and output queue, while a write request, denoted as t_{memW}, includes the latency of input queue and memory.

The service time of the Abus and the Dbus are dependent on their own utilization. The service time of the Abus can be derived as follows: let U_{Abus} denote the partial utilization of the Abus raised by the requests from a particular processor. A new request from the other *n-1* processors can be issued during the remaining part of time (when the Abus is not busy), i.e., $(1-U_{Abus})$. Thus, $(nU_{Abus}-U_{Abus})$ is the partial utilization of the Abus taken to serve the requests from other processors. The probability that the Abus is busy when a request from a particular processor arrives is $(nU_{Abus}-U_{Abus})/(1-U_{Abus})$.

A bus tenure is a duration that a master is controlling (using) the bus. At any point during a master's bus tenure, the bus arbitration process can proceed concurrently with the data transferring process. We approximate the arbitration time of a request's bus service tenure by considering it to be proportional to the probability that bus is ready when a request arrives. Therefore, the mean time for transferring an address on Abus, S_{Abus}, is

$$S_{Abus} = \left(1 - \frac{nU_{Abus}-U_{Abus}}{1-U_{Abus}}\right) t_{arbi} + t_{addr}.$$

Similarly, the mean time for transferring a cache line on Dbus, S_{Dbus}, is

$$S_{Dbus} = \left(1 - \frac{nU_{Dbus}-U_{Dbus}}{1-U_{Dbus}}\right) t_{arbi} + t_{data}.$$

Note that the equations of partial utilizations U_{Abus} and U_{Dbus} will be derived and presented later.

(2) Response Time: The mean response time of a type j blocking request to the ith subsystem, $R_{i,j}$, is dependent on the waiting time caused by those requests that are currently queued or being served in the subsystem and on the service time of the blocking request itself and can be estimated as

$$R_{i,j} = \sum_{k\in H_i} W_{i,j,k} + S_{i,j};\ i \in \{L2, bus, mem\}, j \in B_i \quad (1)$$

where $\sum_{k\in H_i} W_{i,j,k}$ is the mean waiting time of a blocking request of type j at the ith subsystem caused by all request types in H_i.

(3) Subsystem Access Time Per Instruction: The subsystem SATPI of the ith subsystem is the sum of the response time of those blocking requests from a particular processor and can be represented as

$$T_i = \sum_{j\in B_i} p_j R_{i,j} \qquad i \in \{L2, bus, mem\} \qquad (2)$$

where p_j denotes the probability of type j requests.

(4) Average Execution Time Per Instruction: The average execution time per instruction seen by each processor equals to the sum of processor internal processing time, i.e., t_{cpu}, and all SATPIs and can be expressed as

$$T_{exe} = t_{cpu} + \sum_{i\in\{L2,bus,mem\}} T_i\,. \qquad (3)$$

(5) Partial Utilization: The partial utilization of the ith subsystem by any request of type k from a particular processor, $U_{i,k}$, is the ratio of the service time by the requests over total elapsed time, T_{exe}, and can be estimated as

$$U_{i,k} = p_k S_{i,k} / T_{exe};\ i \in \{L2, bus, mem\},$$

$k\in H_i$ where p_k is the probability that the request is of type k; $S_{i,k}$ is the ith subsystem service time for the request of type k.

The partial utilization of the ith subsystem by a particular processor is U_i and can be estimated as

$$U_i = \sum_{k\in H_i} U_{i,k}. \qquad i \in \{L2, bus, mem\}$$

(6) Partial queue length: The average partial queue length of the ith subsystem includes the average number of requests that is currently being served by the subsystem. Therefore, the mean partial queue lengths of requests of type k at the ith subsystem, $Q_{i,k}$, by a particular processor can be approximated by

$$Q_{i,k} = p_k(W_{i,k,l}+S_{i,k})/ T_{exe}; \quad i \in \{L2, bus, mem\}, \quad k\in H_i.$$

(7) Waiting Time: An arriving request at the ith subsystem will wait for the mean residual service time, i.e., mean remaining service time, of the request in service, plus one mean subsystem service time for every other requests in the queue when it arrives. We assume that the service time at each subsystem is deterministic; so the residual service time is approximately one-half its service time. Thus, the waiting time of a request of type j at the ith subsystem caused by a request of type k, for $i \in \{bus, mem\}$, and $j,k\in H_i$, is

$$W_{i,j,k} = \begin{cases} (n-1)((Q_{i,k} - U_{i,k}) S_{i,k}+U_{i,k} S_{i,k}/2); & \text{if } j=k \\ n((Q_{i,k} - U_{i,k}) S_{i,k}+U_{i,k} S_{i,k}/2); & \text{otherwise} \end{cases}$$

The L2 cache must serve the requests from its local processor and from the shared bus. The requests from the shared bus, such as invalidation, takes priority in the L2 cache. So if an invalidation request from the shared bus arrives concurrently with a read request from the processor, the bus request is served first. In other words, the $W_{L2,L1\_Miss,icl}$ is the average waiting time in the L2 cache of a request from its local processor and their computation formula can be derived as follows:

$$W_{L2,L1\_Miss,icl} = t_{L2} p_{icl} \sum_{k=1}^{\infty} \text{Prob.[CPU succeeds on } k\text{th trial]}$$
$$= t_{L2}\, p_{icl} \frac{1-P_{success}}{P_{success}}\ ;$$

where the probability, $P_{success}$, that a processor successfully accesses its L2 cache is given by

$$P_{success} = (1 - p_{icl})^{n-1}.$$

Note that the derivation of $W_{L2,L1\_Miss,icl}$ does not follow the Equation (7), because the L2 cache favors the shared bus requests.

(8) Subsystem Utilization: The utilization of one L2 cache (U_{L2}) is the sum of the time fraction that the L2 cache takes to serve the requests from the processor and from the shared bus. It is represented as

$$U_{L2} = U_{L2,L1\_Miss} + U_{L2,icl}.$$

The total utilization of Abus and Dbus are equal to $n*U_{Abus}$ and $n*U_{Dbus}$, respectively.

The total utilization of main memory is estimated as

$$U_{mem} = n(U_{mem,Miss} + U_{mem,wb} + {}_{mem,dmaR} + U_{mem,dmaW}).$$

The processor utilization can be obtained by t_{cpu}/T_{exe}.

(9) Throughput: The processor throughput and system throughput are measured as million instruction per second (MIPS) and can computed by $1/T_{exe}$ and n/T_{exe}, respectively.

(10) Bottleneck Analysis: In addition to providing performance evaluation results, the other important application of the SAT modelling methodology is to identify the potential causes of bottlenecks existing in a complex multiprocessor system. Based on the SAPTI concept, the SAT modelling methodology has an important advantage over other modelling approaches in that it tells the architecture designer which of the various subsystems is significant in determining the system performance. Specifically, from Equation (3), we observe that the relative contribution of the various SATPIs to the overall system throughput is immediately obvious.

In other words, a subsystem with the longest SATPI is the one that limits processor performance and creates a bottleneck to the system. Particularly, a large difference between subsystem SATPIs is a symptom indicating existence of a bottleneck. Since every multiprocessor system has at least one bottleneck, we can find that the subsystem b is a bottleneck, if

$$T_b = \max\{T_{L2}, T_{bus}, T_{mem}\}.$$

We can continue to identify the cause the bottleneck of the bth subsystem based on the Equation (1) and (2). The T_b is composed of many time components including the waiting time of each type of requests arriving at the bth subsystem and the service time. We can find the maximum term of these components and identify the type of request that causes the bottleneck.

We have developed statistical simulations using SES/workbench [14] to obtain average execution time per instruction and validate the results from our analytical model. By comparing the results obtained from both analytical and simulation models, we find that in about 86% of all cases the magnitude of the percentage difference in average execution time per instruction from the two techniques is less than 3%. We can also see that in less than 1% of all cases the magnitude of the percentage difference is greater than 5%.

4: Performance Analysis of the XMP System

In this section, we conduct performance evaluation for the target architecture using the SAT modelling results. We will describe the workloads for feeding the SAT models and discuss in detail the performance prediction and bottleneck identification for the target architecture. Also, we will explore the impact of the following performance influencing factors: memory latency, cache size, data bus width, as well as DMA transfer.

4.1: Workloads for the analytical Model

We take most of the workloads for the SAT model of the target architecture from existing literatures. Many papers [1, 3-4, 7-8, 13, 15-18] published some architecture-dependent parameters. For examples, Przybylski [13] collected a variety of miss rates for various two-level cache size/structures; Akella and Siewiorek [3] measured the number of bits of DMA transfer per instruction executed by a CPU from VAX11/780 system. To provide input parameters to the SAT modelling, we obtain the values of p_{Miss}, p_{icl}, p_{wb}, and p_{dma}, from these papers that are representative of the workload for which we want to predict system throughput.

The cache miss rate and DMA transfer rate per instruction are also important workloads. In general, the cache miss rate is inversely proportional to the cache size in the target architecture. Based on the results of Przybylski's collected, the L2 cache size of 256K bytes size has an average cache miss rate of 1%. We assume that the number of bits of DMA transfer per instruction is in the range of 1 to 8 bits and the block data size per DMA transfer is based on the cache line size. In the case of cache line of 32 bytes, the probability of DMA transfer, i.e., p_{dma}, ranges from 1/512 to 1/32.

4.2: Performance Prediction and Bottleneck Identification

Our SAT modelling can serve to predict the processor throughput and identify potential bottlenecks. We obtain the processor throughput by computing the average execution time per instruction, as the number of processors and L2 cache size are varied. Fig. 3 shows these results. For a system with 8 processors and 256-Kbyte L2 cache, the processor throughput and system throughput are about 40 MIPS and 320 MIPS, respectively. However, as we can see, the processor throughput decreases with the increase of number of processors. In other words, there exists a bottleneck causing performance degradation in the target architecture.

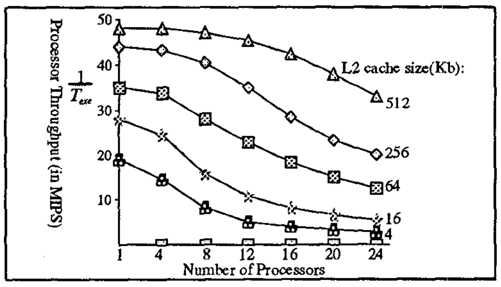

Fig. 3. Processor throughput versus number of processors

For bottleneck identification, we compute the SATPI for each subsystem. Fig. 4 shows the results with different number of processors, assuming that 256-Kbyte L2 cache and 120-nsec memory are used. In each case, the SATPI of memory is the largest among these subsystems including the L2 cache, shared bus, and memory. In addition, since the increase of the SAPTI in the memory subsystem with varying number of processors keeps being the largest, the bottleneck must be present in the memory subsystem.

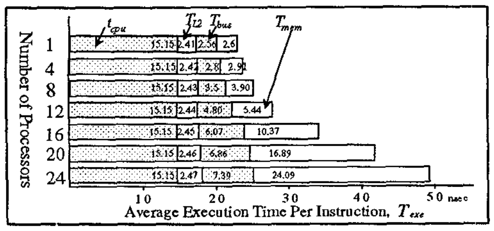

Fig. 4. The SATPIs versus number of processors (with L2 cache size of 256 Kbytes and memory latency of 120 nsec)

In addition, we can estimate an adequate number of processors in the target system based on processor utilization. The value of per processor utilization (t_{cpu}/T_{exe}) can be obtained according to the results shown in Fig. 4. For a 4-processor system, the per processor utilization is equal to 65%. When the processor's number is more than 12, the per processor utilization is less than 50%. Based on the suggestions of Jain [10], the optimum value of utilization lies in the range from 50% to 75%. Therefore, the adequate number of processors for the given set of workload parameters is 12 or smaller.

4.3: Impact of Memory Latency and Number of Memory Modules

In this subsection, we apply the SAT models of the memory subsystem in analyzing the bottleneck in the memory subsystem and exploring the impacts of memory latency and number of memory modules on system throughput.

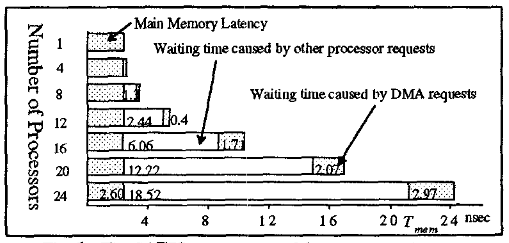

Fig. 5a. The SATPI components of the memory subsystem (with memory latency of 120 ns and one memory modules)

The SATPI of the memory subsystem (T_{mem}) can be used to find the cause of the bottleneck which has been identified to exist in the memory subsystem in the previous subsection. Based on the types of memory requests, T_{mem} can be decomposed into three parts: main memory latency, the waiting time caused by DMA requests, and the waiting time caused by other processor requests. Fig. 5a shows the values of T_{mem} and its components versus various number of processors. It shows that the SATPI is primarily due to the memory waiting time if the number of processors is larger than 8. In other words, too many requests waiting in the memory subsystem is the cause of the bottleneck. This means that there is a mismatch between the memory service rate and the memory request rate.

The bottleneck in the memory subsystem can be removed either by using faster memory chips to increase memory service rate or by using multiple memory modules to distribute the memory requests into the modules. Thereby, the memory waiting time is reduced and the performance of the memory subsystem can be improved. Fig. 5b and 5c show the system throughput impact of memory latency and number of memory modules, respectively. From Fig. 5b, we can see a knee point at memory latency of 90 nsec. Before the knee, the throughput does not increase significantly. This is the point beyond which the throughput decreases rapidly as a function of memory latency. As the knee of the throughput curve is considered the optimal operating point [13], therefore, the main memory latency of 90 nsec under our workload is a good choice.

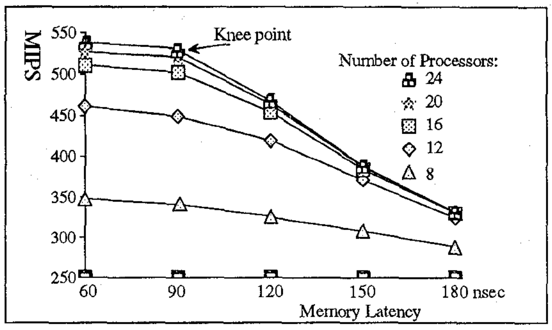

FIg. 5b. System throughput versus memory latency

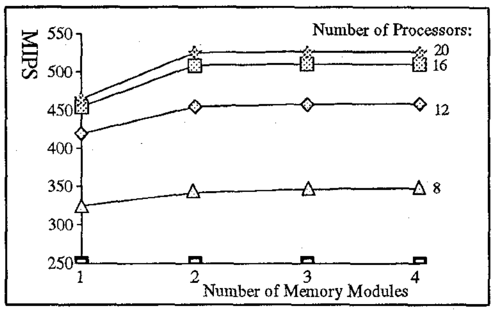

Fig. 5c. System throughput versus number of memory modules (with memory latency of 120 nsec)

From Fig. 5c, using two memory modules can improve the system throughput. However, the curves indicate that no further performance improvement can be achieved for number of memory modules greater than two. Fig. 5b and 5c also imply that even if we minimize the memory access time, the system throughput may still be limited because of other types of subsystem access delays, such as bus delay. (According to Fig. 4, the shared-bus access delay is the second factor that reduces performance, and will be further investigated in detail later.) Fig. 5d shows the SATPI of the memory by using the 90-nsec memory and two memory modules. By comparing Fig. 5a with 5d, we find that using high speed memory and multiple memory modules are effective methods for improving the performance in the memory subsystem.

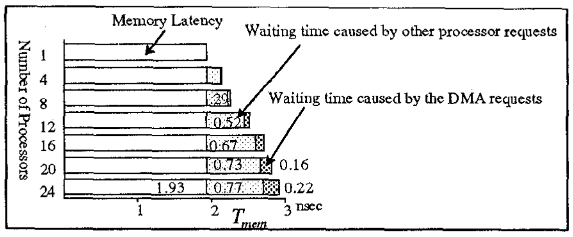

Fig. 5d. The SATPI components of the memory subsystem (with memory latency of 90 nsec and two memory modules)

4.4: Exploring the cause of bottleneck in shared bus

To find the factor in the shared-bus subsystem that degrades the system performance, we decompose the SATPI of the shared bus, T_{bus}, into two components: the waiting time and the holding time. The bus holding time is the sum of arbitration time and address/data transferring time. Fig. 6a shows the results. Since the arbitration time is overlapped with bus service tenure, increasing the number of processors results in decrease in the bus holding time. Furthermore, Fig. 6a also shows that the waiting time on the Dbus is longer than on the Abus, because the time of transferring a cache line on the Dbus usually takes a longer time than the time of transferring an address on the Abus. In other words, the SATPI of the shared bus is more sensitive to the Dbus holding time than to the Abus. Therefore, in the shared-bus subsystem, the Dbus with a long waiting time is one of the major factors that drops the processor throughput.

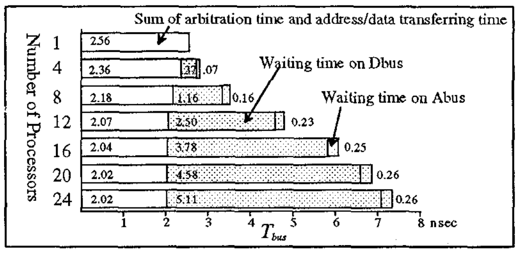

Fig. 6a. Shared bus access time (Dbus width of 8 bytes)

4.5: Impact of DMA transfer

Another special consideration in our performance model is the impact of DMA transfer. Fig. 7 shows the evaluation results. It is clear that if the number of bits of DMA transfer per instruction is less than 1 bit (i.e., less than 1 Mbit/sec per MIPS following Amdahl's Law [1]), the system throughput remains almost constant for up to 12 processors. As the number of bits of DMA transfer per instruction is increased up to 8 bits (i.e., 1 Mbyte/sec for 1 MIPS following the results measured from VAX machine by Akella and Siewiorek [3]), the system throughput drops about 45%. In addition, Fig. 5a also shows the impact of DMA transfer on memory access time; one bit of DMA transfer per instruction is assumed. If the number of processors is less than 8, the memory access time impact of DMA transfer is negligible. For 16 processors system, the effect of DMA transfer on the SATPI of the memory subsystem is significant (increase to about 16.5%). For applications which require higher DMA transfer rate than 1 bit per instruction, the impact of DMA transfer on the SATPI of memory will increase.

Thus we believe that the impact of DMA transfer for most applications will be significant and should be included in the performance model of a shared bus multiprocessor system.

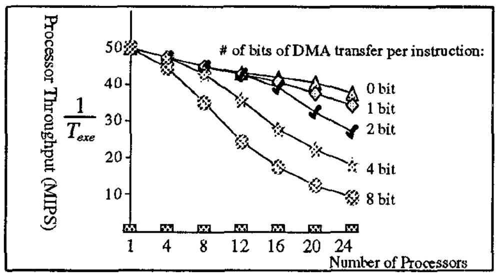

Fig. 7. Impact of DMA transfer one processor throughput

5: Conclusion

We have developed an analytical model based on subsystem access time (SAT) modelling approach for the XMP shared-bus multiprocessor system. The system is composed of bus masters, such as processor and DMA controller, and subsystems, including cache, bus, and memory. A key characteristic of the target system, namely the support of separate address-bus (Abus) and databus (Dbus) with split transaction, pipelined cycle, is modelled. Furthermore, several performance factors: cache interference, bus interference, memory interference, and the impact of DMA transfer are also considered in the model. The SAT modelling can facilitate computing the average subsystem access time per instruction (SATPI) of all subsystems and evaluating some initial system design issues. By comparing these SATPIs, we can easily identify the subsystem having the longest SATPI to be the bottleneck and then explore potential improvements of system performance.

Using the analytical model, we have explored the impact of DMA transfer on system performance. We find that the effect of DMA transfer drops the system throughput about up to 45% according to the workloads measured by Akella and Siewiorek from VAX machines [3]. Consequently, it is essential to include the impact of DMA transfer in a performance model.

We have also explored the application of the SATPI of the shared bus subsystem. Result shows that the shared bus performance is quite sensitive to the Dbus delay (more than the Abus delay).

References

[1] T. Adams and R. Zimmerman, "An analysis of 8086 instruction set usage in MS DOS programs," *Proc. Third Symposium on Architectural Support for Programming Language and Oper. Syst.*, pp. 152-161, 1989.

[2] G. M. Amdahl, "Validity of the single processor approach to archiving large scale computing capability," *American Federation of Informance Processing Societies*, pp. 483-485, 1967.

[3] J. Akella and D. P. Siewiorek, "Modeling and measurement of the impact of input/output on system performance," *Proc. of the 18th Annu. Symp. Comput. Architecture,* pp 390-399, 1991.

[4] M. C. Chiang and G. S. Sohi, "Evaluating design choices for shared bus multiprocessors in a throughput-oriented environment," *IEEE trans. Comput.*, vol. 14, no. 3, pp. 297-317, Mar. 1992.

[5] A. L. Decegama, "The Technology of parallel processing - parallel processing architectures and VLSI hardware, volume I," *Prentice-Hall International Editions*, 1989.

[6] P. J. Denning and J. P. Buzen, "The Operational analysis of queueing network model," *ACM Comput. Surveys*, vol. 10, pp. 25-261, Sep. 1978.

[7] S. J. Eggers and R. H. Katz, "A Characterization of sharing in parallel programs and its applicability to coherency protocol evaluation," *Proc. of the 15th Annu. Symp. Comput. Architecture,* pp. 373-382, 1988.

[8] J. L. Hennessy and D. A. Patterson, "Computer architecture a qualitative approach," *Morgan Kaufmann,* 1990.

[9] Intel XA-MP Architecture Specification ver 3.0, July 1991.

[10] R. Jain, "The Art of computer systems performance analysis," *John Wiley & Sons,* 1991.

[11] C. S. Lee and T. M. Parng, "A Subsystem-oriented performance analysis methodology for shared-bus multiprocessors," *NTUEE Tech. Report*, Sep. 1993.

[12] S. Leutenegger and M. K. Vernon, "A mean-value performance analysis of a new multiprocessor architecture," *in Proc. ACM SIGMETRICS Conf. Measurement and Modelling of Comput. Syst.,* May 1988.

[13] S. A. Przybylski, "Cache and memory hierarchy design: a performance-directed approach," *Morgan Kaufmann,* 1990.

[14] SES/workbench User's manual and Reference's Manual, Jan. 1991.

[15] R. T. Short and H. M. Levy, "A Simulation of two-level caches," *Proc. of the 15th Annu. Symp. Comput. Architecture,* pp. 81-88, 1988.

[16] J. P. Singh, H. S. Stone, and D. F. Thiebaut, "A Model of workloads and its use in miss-rate prediction for fully associative caches," *IEEE Trans. Comput.*, pp. 811-825, July 1992.

[17] T. F. Tsuei and M. K. Vernon, "A Multiprocessor bus design model validated by system measurement," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 6, pp. 712-727, Nov. 1992.

[18] M. K. Vernon, E. D. Lazowska, and J. Zahorjan, "An Accurate and efficient performance analysis technique for multiprocessor snooping cache-consistency protocols," *Proc. of the 15th Annu. Symp. Comput. Architecture,* pp. 308-315, June 1988.