Proceedings of International Conference
on Computer Architecture

A Subsystem-Oriented Performance Evaluation Tool for Computer Architectures

Chiung-San Lee
Computer Center
National Taipei College of Nursing
Taipei, Taiwan, R.O.C.

Tai-Ming Parng
Dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Jeen-Fong Lin
Dept. of Electronic Engineering
Hwa-Hsia College of
Technology and Commerce

Abstract

Although performance evaluation tools are popular for analyzing and predicting throughput of computer systems, existing evaluation tools usually have a difficult and time-consuming modeling process because (1) modeling requires specific mathematical background; (2) most modern systems are too large and complex to be modeled directly; and (3) modeling for exploring various alternative designs always takes time and patience. Moreover, existing tools do not support well the capability for finding the bottleneck and its cause of a target system being evaluated. To address the above problems, in this paper we propose a subsystem-oriented approach to analytical performance modeling of computer systems. The whole approach is built on a subsystem-oriented performance evaluation methodology and tool, which are, in turn, based on a subsystem specification language.

1. Introduction

The use of evaluation tools to analyze and predict the performance of computer systems is now widespread [1, 2, 5, 6, 7, 9, 11]. These tools incorporate various different techniques, such as simulation [2, 3] and analytical models [1, 9], are suitable for the performance modeling and evaluating such computer systems. There have been published a number of performance evaluation tools [2, 4, 5, 6, 7]. One of the recent work Experimenter [7] is a performance evaluation tool that supports structured experiments for evaluating complex computer systems and enhancing the reuse of previously developed performance models. DESIGN [4] is a software evaluation tool to provide a user-friendly system for both specifying and analyzing software designs. The Graphical Modeling and Analysis (GMA) tool [5] focuses on automatically generating performance models for communication networks, and allows users to manipulate a schematic diagram of network configuration via a high-level graphical interface. Others apply simulation techniques. As well-knowing example is SES/workbench [2]. It is that a high-level simulation tool; its simulation model is based on the probabilistic distributions to characterize workload parameters, and is composed of a sequence of transactions, nodes, and arcs. These analytical and simulation tools are useful to designers.

However, most of existing performance evaluation tools usually have a difficult and time-consuming modeling process and lack a straightforward method for bottleneck and cause identification [11]. First of all, performance analysis requires strong mathematical or statistical background for developing performance models. Second, large and complex target systems always aggravate the difficulty in developing performance models. Third, it always takes a long time in redeveloping several analytical models to evaluate various alternative designs. Finally, lacking the capability of bottleneck analysis is the other major problem associated with existing performance evaluation tools.

To address the above problems, we developed a subsystem-oriented performance evaluation methodology (SOPEM) and realized it with a subsystem-oriented performance evaluation tool (SOPETOOL). In SOPEM, we use a subsystem specification language (SSL) to describe target systems. In general, a computer system is consisted of several *subsystems*, such as cache, bus, and memory; these subsystems will invoke or serve various types of arriving *requests*.

The subsystem specification language SSL is the key to achieving a simple performance modeling process. It is subsystem-oriented and allows designers to specify specification parameters of each subsystem and arriving requests, such as subsystem name, service time, request names, and visiting probabilities of requests. Moreover, it allows an architectural designer to characterize a target system without any mathematical modeling experience. Particularly, it realizes the model reuse feature and makes the tasks of reusing previously developed SSL models simple and straightforward.

SOPEM is an iterative performance evaluation methodology. Its evaluation process works as an iterative cycle in which (1) its kernel language SSL is used to model a target system; (2) the SSL model is translated into computer program based on mean value analysis (MVA) equations [9, 10, 12]; (3) performance results are obtained by running the program; (4) the designer analyzes these results to identify bottleneck and cause; and (5) the designer modifies the SSL model to conduct more evaluations until the target meets the performance requirement.

SOPETOOL implements SOPEM's idea. It extracts subsystem specifications from an SSL file, relieves designers of painful efforts of doing performance modeling by coding a program in a regular programming language, shortens the time for performance evaluation and bottleneck analysis, and shows performance results in easy-to-read textual explanation and graphic chart. Briefly, it is composed of four important stages: parsing, code generation, iterative computation, and results presentation. On the whole, it is a powerful performance evaluation environment to support easy and fast performance evaluation process as well as the capability of bottleneck analysis.

The remainder of this paper, is organized as follows. Section 2 contains an overview of the subsystem-oriented performance evaluation methodology SOPEM. In Section 3, we introduce the subsystem specification language SSL. Section 4 presents an SSL model example of a shared bus multiprocessor system. In Section 5, we give an overview of the subsystem-oriented performance evaluation tool (SOPETOOL). Section 6 contains the conclusions of this work.

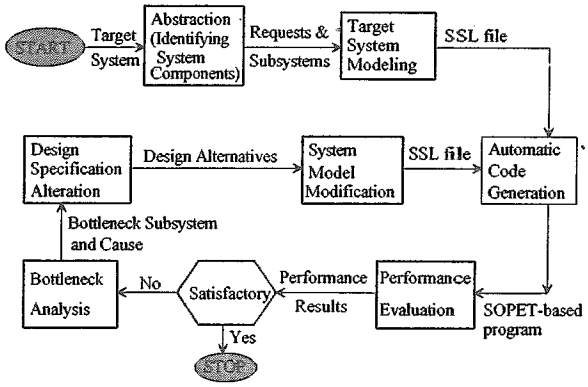

Fig. 1. Iterative evaluation procedure of SOPEM

2. Overview of SOPEM

As mentioned earlier, SOPEM works in an iterative cycle for performance evaluation and design modification for computer systems. It consists of several basic stages for the iterative evaluation and modification, as illustrated in Fig. 1. The abstraction stage is to identify important architecture aspects and produce a conceptual model. The target system modeling stage outputs a target SSL model based on the conceptual model. During the automatic code generation stage, the SSL model is translated into a computer program. Performance evaluation stage runs the program and yields the desired performance data. During the bottleneck analysis stage, a bottleneck subsystem and its cause are identified and displayed. Then, the design specification alteration stage removes the bottleneck by changing design parameters and correcting the problem based on the bottleneck information. Finally, during the system model modification stage the target SSL model is modified for starting another iteration or ending the whole evaluation process if the performance goal is met.

3. Subsystem Specification Language (SSL)

SSL is a subsystem-based language in which subsystems are basic definition units. The subsystem specification for performance modeling is related to various types of arriving requests. Each request type has a particular set of visiting probability, service time, whether it will include other subsystem's holding time, ... etc. These parameters can be directly specified by using SSL. For specifying these parameters values, a set of keywords are predefined for constructing an SSL model.

Several important specification parameters must be given within an SSL model. First, we must specify the probabilities of arriving requests (e.g., the probability of memory-read request arriving at Abus). Second, we must specify the service demand, which is the average service time of a request (e.g., 2 bus clocks for Abus to serve a memory-read request). Third, since some requests may hold more than one subsystems, we must specify which one's holding time will be included. Finally, if special functions of some particular hardware behaviors are needed, we may specify the FUNCTION to include an executable program file as a procedure.

An SSL model is composed of a set of assertions, including one SYSTEM-NAME, several SUBSYSTEMs, and one END. An assertion is one or more statements to define some specification parameters. The SYSTEM-NAME assertion defines the system name. The SUBSYSTEM assertions specify major hardware components of a target system. The last assertion in any SSL file must be END. The END tells the SOPEM that there are no more assertions to be asserted. An example model illustrating the SSL syntax is given below:

```
SYSTEM-NAME: system-name;
SUBSYSTEM  [SubsystemName='subsystem-name',
     NumberOfRequester=integer,
     ServiceTime=real, BlockRequest(RequestName='request-name'),
     ... ];
SUBSYSTEM  [SubsystemName='subsystem-name',
     ServiceTime=real+U(AdditionalTime=real),
     BlockRequest(RequestName='request-name',
     ServiceTimeOfTheRequest=real, Probability=real,
     Utilization=Default+W(SubsystemName='subsystem-name'),
     QueueLength=Default+W(SubsystemName='subsystem-name'),
          WaitingTime=Default+
          FUNCTION('filename@subsystemname/subsystemname'),
     ...
     NonBlockRequest(RequestName='request-name',
          ServiceTimeOfTheRequest=real+U(AdditionalTime=real),
          Probability=real,
     Utilization=Default+W(SubsystemName='subsystem-name'),
     QueueLength=Default+W(SubsystemName='subsystem-name'),
          WaitingTime=Default+
          FUNCTION('filename@subsystemname/subsystemname')),
     ... ];
...
...
END
```

4. SSL Model of a Shared-Bus Multiprocessor System

4.1. Overview of a shared-bus multiprocessor system

We consider a multiprocessor system, called XMP [9, 11], with several processors, main memory subsystems, and I/O subsystems connected together by a shared bus consisting of an address bus and a data bus. Each processor has a 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. 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 target architecture adopts a two-level cache structure for reducing bus traffic. 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.

The XMP architecture is consisted of L2 cache, Abus, Dbus, and memory subsystems. Fig. 2 presents a queueing network model of the XMP architecture. A number of processors and a DMA controller are represented as 'delay' nodes; the delay denotes the time needed for processor activity between two consecutive instruction executions. The subsystems are represented as 'service' nodes; each service node is equipped with a parameter which indicates the subsystem access time (including the queueing delay time and hardware processing time).

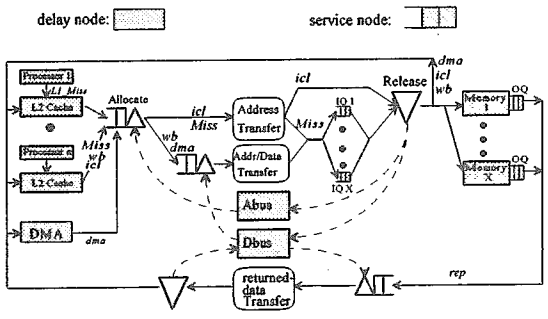

Fig. 2. Queueing network model of the XMP system

The following types of requests are included in the XMP model:

- *L1-hit*, representing that the processor requests and obtains the data in its on-chip cache without additional delay.
- *L1-Miss*, representing that the processor requests a data which fails in the L1 cache.
- *L2-hit*, representing that the data access of the processor fails in the L1 cache and succeeds in the L2 cache without issuing a bus request.
- *Miss*, or *read-miss*, representing that the processor requests a data which is currently not in its L2 cache and must access memory via the Abus.
- *rep*, or *memory-reply*, representing that the memory subsystem puts the requested data onto the Dbus when the memory read operation is complete.
- *icl*, or *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.
- *wb*, or *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.
- *dma*, or *DMA-transfer*, representing that the DMA controller transfers data to/from memory via the Abus and Dbus.

Note that the *L1_hit*, *L1-Miss* and *L2_hit* requests flow directly back into the processors and are not included in the models of the subsystems.

From Fig. 2, we can see that the request types of all subsystems include *L1_Miss*, *Miss*, *icl*, *wb*, *dma*, and *rep*. *L1_Miss* and *icl* will visit the L2 cache; *L1_Miss* is a blocking request, because the processor waits until it receives the response to the instruction issuing the request. There are five request types that visit the shared bus; two of them, *Miss* and *icl*, are blocking ones. Three request types, *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 (1) no data response is required for the *icl* request, and (2) the tag update in the L2 cache will not cause any delay to the requester.

4.2. *SSL model of XMP*

The SSL model of the XMP system is shown in Fig. 3.

```
SYSTEM-NAME: XMP
SUBSYSTEM [SubsystemName='CPU', NumberOfRequester=8,
      ServiceTime=0.8
      BlockRequest(RequestName='L1_Miss') ];
SUBSYSTEM [SubsystemName='L2', ServiceTime=2.0,
      BlockRequest(RequestName='Miss',
            ServiceTimeOfTheRequest=Default,
            Probability=0.20, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='icl',
            ServiceTimeOfTheRequest=Default,
            Probability=0.104, Utilization=Default,
            QueueLength=Default, WaitingTime=0.4),
SUBSYSTEM [SubsystemName='Abus',
            ServiceTime=1.0+U(AdditionalTime=2),
      BlockRequest(RequestName='Miss',
            ServiceTimeOfTheRequest=Default,
            Probability=0.20,
            Utilization=Default+W(SubsystemName='Dbus'),
            QueueLength=Default+W(SubsystemName='Dbus'),
            WaitingTime=Default),
      BlockRequest(RequestName='icl', ServiceTimeOfTheRequest=Default,
            Probability=0.013, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='wb',
            ServiceTimeOfTheRequest=Default,
            Probability=0.03,
            Utilization=Default+W(SubsystemName='Dbus'),
            QueueLength=Default+W(SubsystemName='Dbus'),
            WaitingTime=Default),
      NonBlockRequest(RequestName='dma',
            ServiceTimeOfTheRequest=Default,
            Probability=0.03,
            Utilization=Default+W(SubsystemName='Dbus'),
            QueueLength=Default+W(SubsystemName='Dbus'),
            WaitingTime=Default) ];
SUBSYSTEM [SubsystemName='Dbus',
            ServiceTime=4.0+U(AdditionalTime=2),
      BlockRequest(RequestName='rep',
            ServiceTimeOfTheRequest=Default,
            Probability=0.20, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='wb',
            ServiceTimeOfTheRequest=Default,
            Probability=0.03, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='dma',
            ServiceTimeOfTheRequest=Default,
            Probability=0.03, Utilization=Default,
            QueueLength=Default, WaitingTime=Default) ];
SUBSYSTEM [SubsystemName='mem', ServiceTime=7.0,
      BlockRequest(RequestName='Miss', ServiceTimeOfTheRequest=9.0,
            Probability=0.20, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='wb',
            ServiceTimeOfTheRequest=7.0,
            Probability=0.03, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='dmaR',
            ServiceTimeOfTheRequest=9.0,
            Probability=0.03, Utilization=Default,
            QueueLength=Default, WaitingTime=Default),
      NonBlockRequest(RequestName='dmaW',
            ServiceTimeOfTheRequest=7.0,
            Probability=0.03, Utilization=Default,
            QueueLength=Default, WaitingTime=Default) ];
END
```

Fig. 3. SSL Model of the XMP multiprocessor system

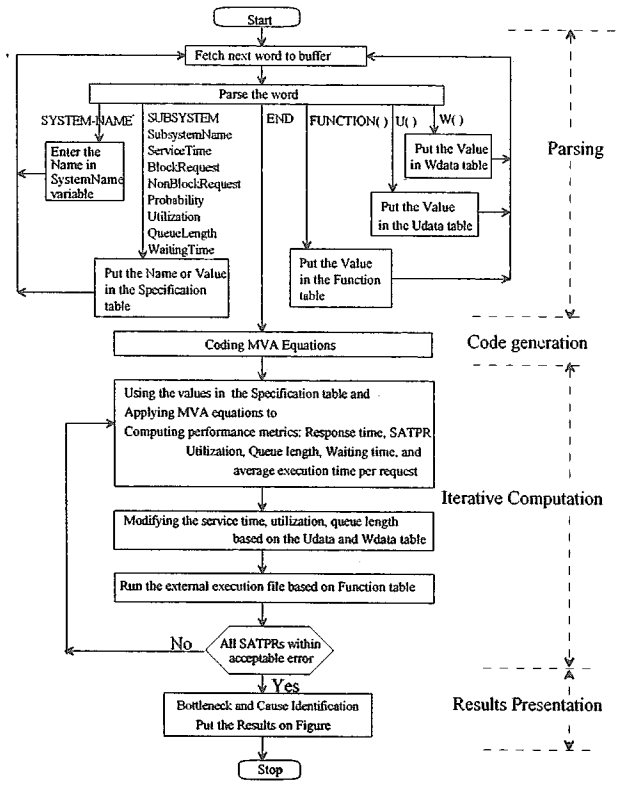

Fig. 4. Structure of SOPETOOL

5. Overview of SOPETOOL

5.1. Structure of SOPETOOL

SOPETOOL is an integrated tool for performance evaluation. It consists primarily of four stages: parsing, code generation, iterative computation, and results presentation, as shown in Fig. 4.

Parsing is to break down an assertion or statement into its parts and process additional information. It is done by a keyword-based parser, which works word by word, using space and comma as delimiters. The processed word is brought into a buffer, where additional information is extracted. The extracted information is stored into internal tables with respect to particular keywords. These keywords have SYSTEM-NAME, SUBSYSTEM, SubsystemName, Utilization, END, FUNCTION(), U(), W(), ... etc.; internal tables have **Specification** table, **Udata** table, **Wdata** table, and **Function** table. **Udata** table stores the additional information related to U() keyword; similarly, **Wdata** and **Function** tables store the additional information related to W() and FUNCTION() keywords, respectively. Other additional information of remaining keywords are stored into the **Specification** table, as shown in Fig. 4. Finally, when the END keyword is detected, code generator is triggered.

The code generation stage is to generate an executable program based on MVA equations [9-12] and SOPETOOL internal tables. The internal tables supply all important specification with respect to a target system as input/output parameters to MVA equations. Most of MVA's parameters are fetched from the **Specification** table and performance results are also stored into the **Specification** table. Others related to user defined parameters, such as external function call name, additional information, are gotten from **Udata**, **Wdata**, and **Function** tables. In summary, during this stage, MVA equations are translated into a program according to target system's specification stored in SOPETOOL's internal tables for performance evaluation.

The iterative computation stage is to run the generated program for obtaining performance results and processing user defined information. During this stage, a heuristic iterative algorithm is applied to converse the performance results. Moreover, the user defined information is also processed to generate particular service time, utilization, queue length and waiting time. In addition to processing user defined information, SOPETOOL also provides a general mechanism that permits designers to run their own computation procedures. These external procedures must be compiled into executable files; and then these files can be run by SOPETOOL during the iterative computation stage. Returned values from an external procedure are also saved into the **Specification** table. Note that the communication between SOPETOOL and the external procedure is implemented by using two data files.

At the end of iteration, performance results are presented. SOPETOOL shows the results in textual explanation for presenting real quantity and graphic chart for presenting easy-to-understand data. Performance results, such as response time, subsystem access time per request (SATPR), utilization, queue length, and waiting time, are presented in textual values. The result of bottleneck analysis is shown in bar chart or pie chart.

Fig. 5 shows the main buttons of SOPETOOL user interface. These buttons are 'Subsystem access time', 'Show data', 'Queue length', 'Waiting time,' and 'Exit'. The 'Exit' button allows an option to stop and exit SOPETOOL.

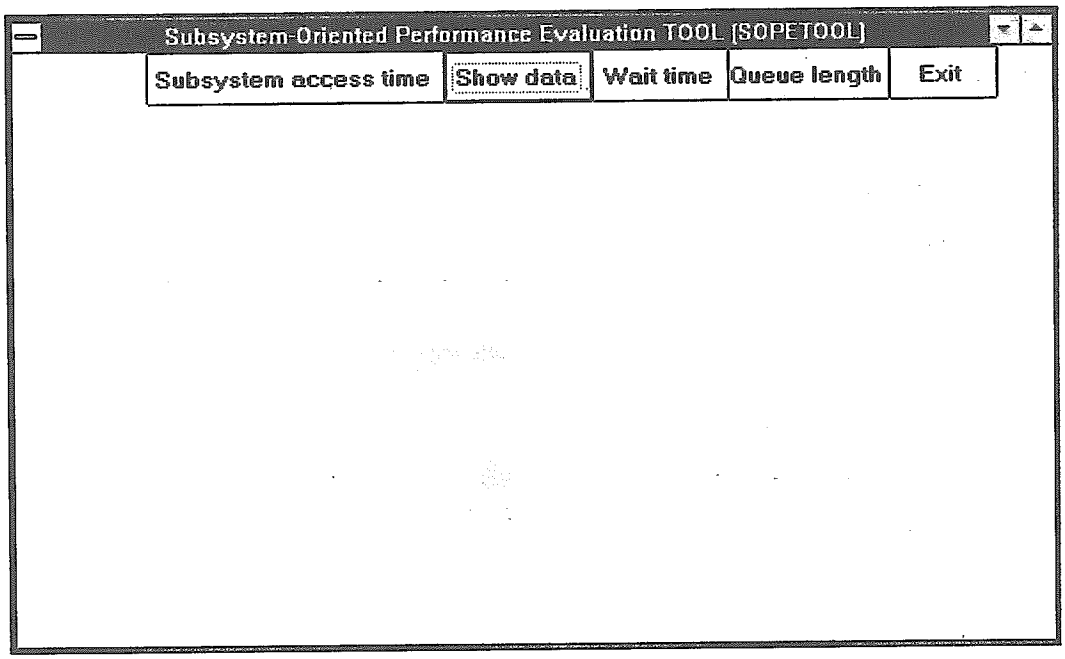

Fig. 5. Main buttons of the SOPETOOL

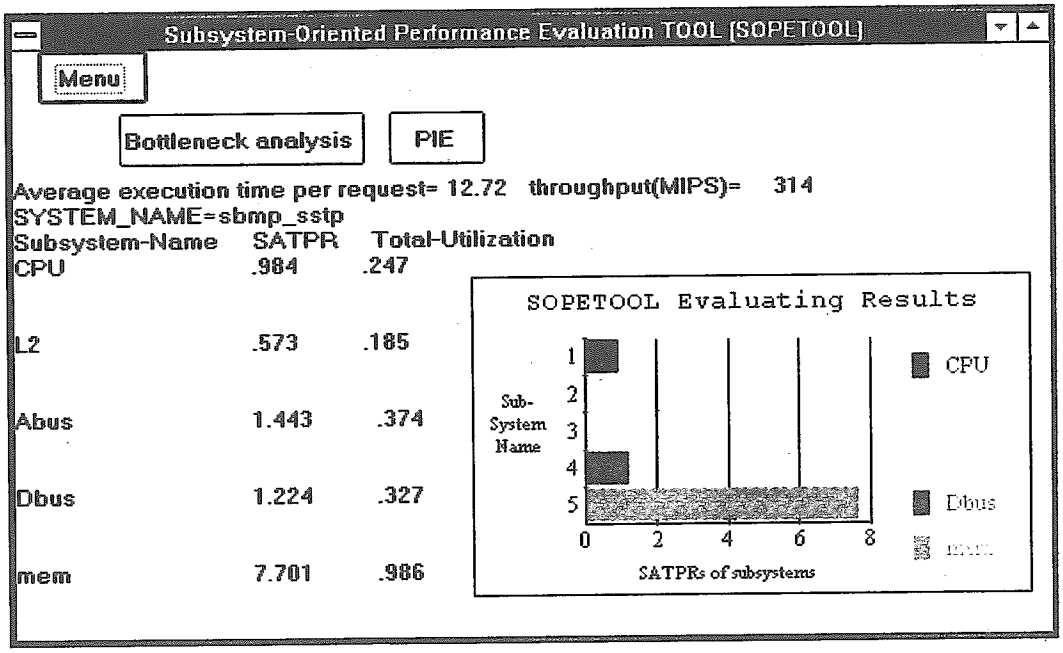

Fig. 6. Subsystem access time per instruction request of all subsystem

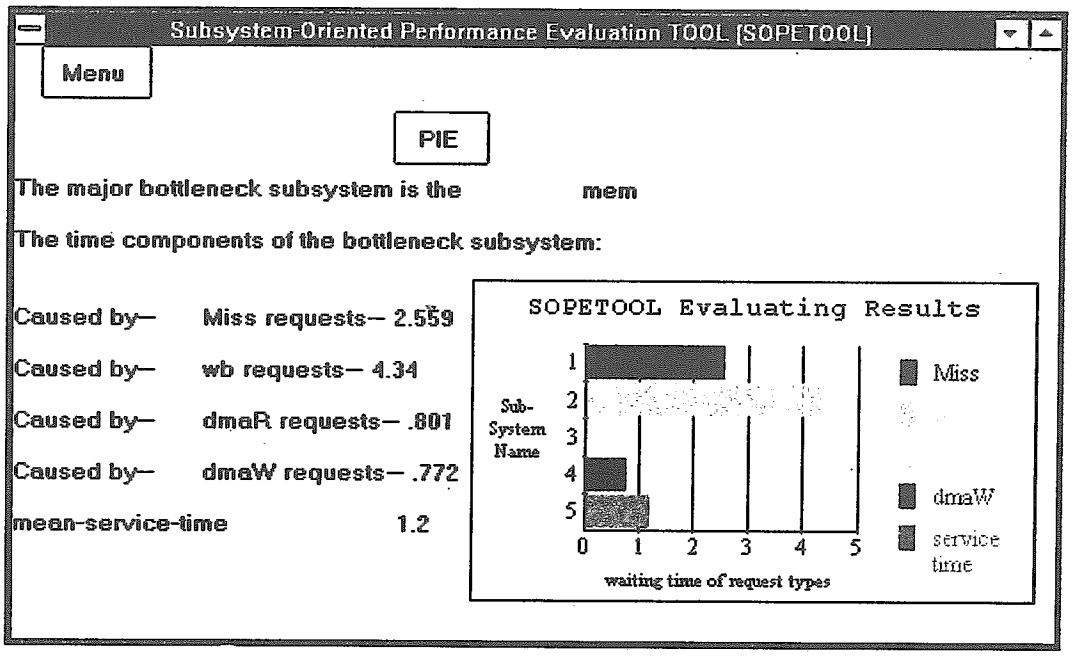

Fig. 7. Bar chart of waiting times in bottleneck subsystem

The 'Queue length' and 'Waiting time' buttons are used to show the number and time of each type of requests waiting in all subsystems. The 'Show data' button can be used to display the input parameters defined in the SSL model. The 'Subsystem access time' button is for displaying subsystem access time [9, 11, 12] of all subsystems and for the bottleneck and cause identification.

5.2. Application Example of SOPETOOL

To demonstrate the ease and usefulness of SOPETOOL in performance evaluation and bottleneck analysis. In this section we employ SOPETOOL to evaluate a particular shared bus multiprocessor (SBMP) system composed of CPU, L2 cache, Abus, Dbus, and memory subsystems. The SBMP architecture and its SSL model have been introduced in Section 4, respectively. The evaluation process begins by feeding the SSL model file into SOPETOOL; then performance results are collected. Fig. 6 shows the average instruction execution time of a processor and SATPRs of all subsystems; we can find that the memory subsystem contributes a longest SATPR to processor execution. Therefore, the memory subsystem is the bottleneck as shown in the bar chart of Fig. 7 and the pie chart of Fig. 8.

6. Conclusion

In this paper, we have presented that subsystem-oriented performance evaluation methodology (SOPEM) is a systematic performance evaluation methodology, which consists mainly of a process for organized development of computer systems. The process includes initial abstraction, modeling target system, generating execution code, evaluating system performance, identifying bottleneck and cause, and followed by a correcting architectural design stage for performance improvement. SOPEM's three main tasks are: modeling a target system in subsystem specification language (SSL), analyzing the performance results, and modifying the SSL model for more evaluation iterations. These tasks will iterate until the system meets a set of predefined performance criteria.

Moreover, we have introduced a performance evaluation tool, namely SOPETOOL, based on SOPEM. With SOPETOOL, we are able to estimate large performance models in a relatively easy way. SOPETOOL analyzes a performance model written in the SSL and process the model in four stages: parsing, code generation, iterative computation, and results presentation. Finally, SOPETOOL is employed to evaluate a shared bus

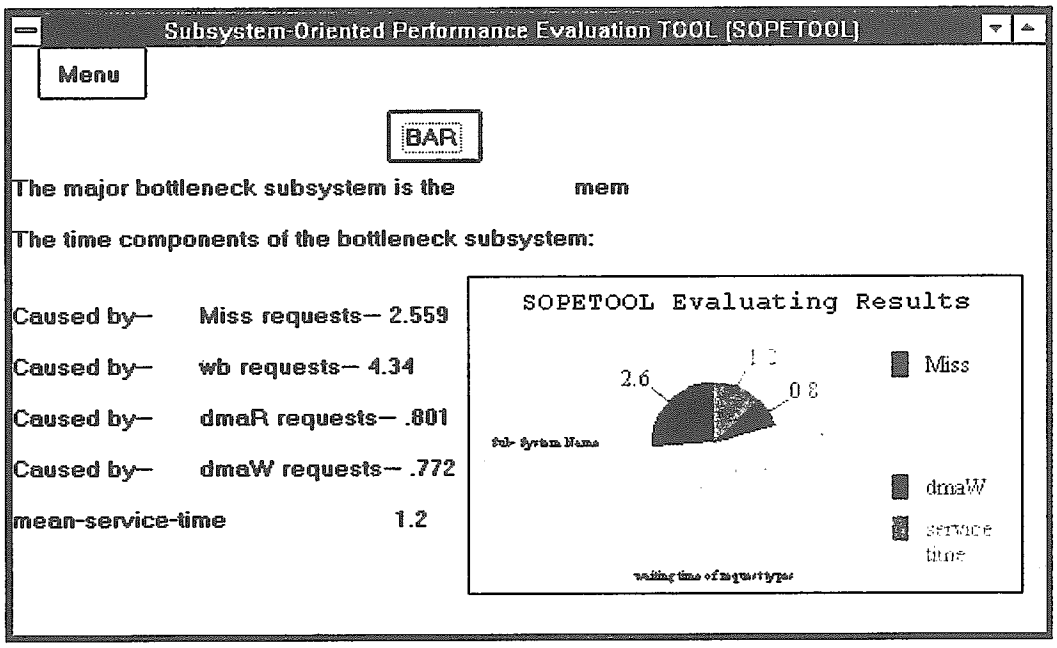

Fig. 8. Pie chart of waiting times in bottleneck subsystem

multiprocessor system and performance results are presented in both text and chart.

7. References

[1] G. Chioia, "A Graphical Petri Net Tool for Performance Analysis," Modeling Techniques and Performance Evaluation, Elsevier Science Publishers B. V. (North-Holland), PP. 323-333, 1987.

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

[3] S. Lacobovici and C. Ng, "VLSI and System Performance Modeling," IEEE Micro, pp. 57-72, Aug. 1987.

[4] K.L. Wong, "Design - A Tool for software Performance Engineering," CMG Transaction, Sep. 1983.

[5] L.V. Zijl, D. Mitton, and S. Crosby, "A Tool for graphical network modeling and analysis," *IEEE Software*, pp. 47-54, Jan. 1992.

[6] S. Crosby, D. Mitton, and L.V. Zijl, "A Graphical tool for the modeling of packet and circuit switched communication networks," *Modeling Techniques and Tools Performance Evaluation*, pp. 105-119, 1992.

[7] J. Hillston, "A Tool to enhance model exploitation," *J. of Performance Evaluation 22*, pp. 59-74, 1995.

[8] P. Schweitzer, "Approximate analysis of multiclass closed networks of queues," *J. ACM*, vol. 29, no. 2, Apr, 1981.

[9] C. S. Lee and Parng T. M., "Bottleneck identification methodology for performance oriented design of shared-bus multiprocessors," , *IEICE Trans. Information and Systems*, vol. E78-D, no. 8, pp. Aug. 1995.

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

[11] C.S. Lee, "A Subsystem-Oriented approach to analytical performance evaluation of computer systems," Dissertation of Electrical Engineering at National Taiwan University, Dec. 1995.

[12] C.S. Lee and T.M. Parng, "Subsystem-Oriented Performance Analysis Methodology for Shared-Bus Multiprocessors," *IEEE Trans. on Parallel and Distributed Systems* (Accepted for publication).