

UNIVERSITY OF LUGANO
Advanced Learning and Research Institute -ALaRI

PROJECT

COURSE: PERFORMANCE EVALUATION

Shared-Memory Multiprocessor Systems – Hierarchical Task Queue

Mentor: Giuseppe Serazzi

Candidates:

Ana Jankovic, MAS

jankovia@alari.ch

Elena Zamsha, MAS

zamshae@alari.ch

Ghazal Haghani, MSc 2nd

haghanig@alari.ch

Lugano, February 2007.

CONTENTS

1. Description of the system	4
2. Introduction	5
3. Propose of the new technique	5
4. The modelling approach	6
4.1. Java Modelling Tools.....	6
4.2. Input parameters.....	7
4.3. Performance analysis	7
5. Simulations	8
5.1. Impact of access contention	9
5.1.1. Centralized organization	9
5.1.2. Distributed organization.....	10
5.1.3. Hierarchical organization	10
5.2. Impact of system size	11
5.2.1. Centralized organization	11
5.2.2. Distributed organization.....	11
5.2.3. Hierarchical organization	12
6. Results & Conclusions.....	13
7. REFERENCES:.....	14

LIST OF FIGURES

Figure 1 Shared-memory multiprocessors system	4
Figure 2 Hierarchical organization for N=8 processors with branching factor B=2	5
Figure 3 Task transfer process in the hierarchical organization for N=64 processors with branching factor B=4 and transfer factor Tr=1	6
Figure 4 Centralized organization with N=4 sinks	8
Figure 5 Distributed organization with N=4 sinks.....	8
Figure 6 Hierarchical organization with N=4 sinks, Br=2, Tr=1.....	9
Figure 7 Centralized, what-if analysis, N=4 sinks	9
Figure 8 Distributed, simple analysis, N=4 sinks	10
Figure 9 Hierarchical, simple analysis, N=4 sinks	10
Figure 10 Centralized, what-if analysis, N=8 sinks	11
Figure 11 Distributed, simple analysis, N=8 sinks	11
Figure 12 Hierarchical organization with N=8 sinks, Br=2, Tr=1.....	12
Figure 13 Hierarchical, simple analysis, N=8 sinks	12

1. Description of the system

Shared-memory multiprocessors are an important class of parallel processing systems.

Shared memory is a large block of Random Access Memory (RAM) which can be accessed by several different Central Processing Units (CPUs) in a multiple-processor computer system. Processors can access one or more shared memory modules. A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The processors can be physically connected to the memory modules in a variety of ways, but logically every processor is connected to every module. Figure 1 presents basic concept of shared-memory multiprocessors system.

Processor scheduling is an important factor that influences the overall system performance and has received considerable attention from several researchers. On the other hand, sharing-memory multiprocessors system will cause some problems. The first issue is to decide how to dedicate memory between processors to obtain best performances of the system which can be represented by measuring its response time and utilization.

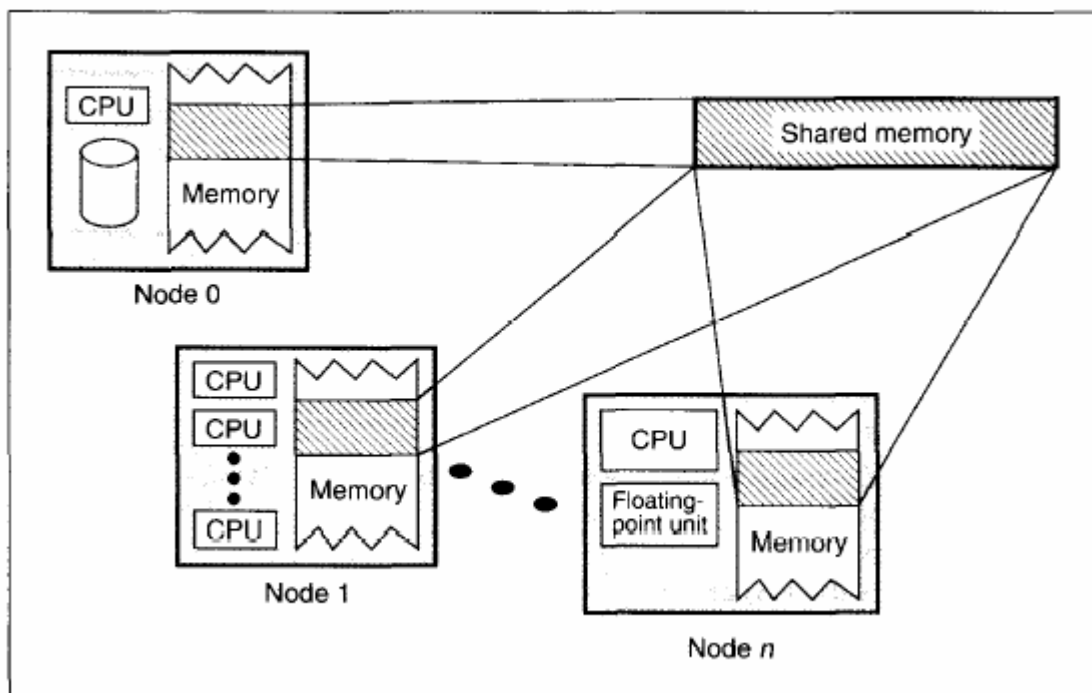


Figure 1 Shared-memory multiprocessors system

Several commercial and research shared-memory machines have been developed including BBN's Butterfly, NYU's Ultracomputer, IBM's RP3, Sequent's Balance and Symmetry, DEC's Firefly, and the Stanford Dash.

2. Introduction

Waiting ready tasks in shared-memory multiprocessors (the main topic of this course work) can be organized in two basic ways – centralized or distributed.

In the **centralized organization** there is a single global queue of ready tasks that is accessible to all processors in the system. However, due to mutually exclusive access, the single global task ready queue becomes the system bottleneck as the number of processors increases, so it seems that centralized organization is not suitable for large parallel systems.

In the **distributed organization**, on the other hand, local ready queues are associated with the processors; this avoids the problem of ready queue access contention but introduces additional problems. The main problem with this organization is to find an appropriate ready task queue for the arrival task (the task assignment problem). A simple random assignment strategy, in which an arriving task is simply assigned to a random ready task queue, causes load imbalance resulting in performance degradation. In the absence of ready task queue access contention, the centralized organization provides better performance mainly due to its load sharing characteristic.

3. Propose of the new technique

The goal is to present a superior technique – **hierarchical organization**, which provides performance similar to the centralized organization even when there is no access contention like in the distributed organization. In this new organization (example is shown in Figure 2), a set of ready task queues is organized as a tree with all the processors with their local queues attached to the bottom level of the tree (as leaf nodes).

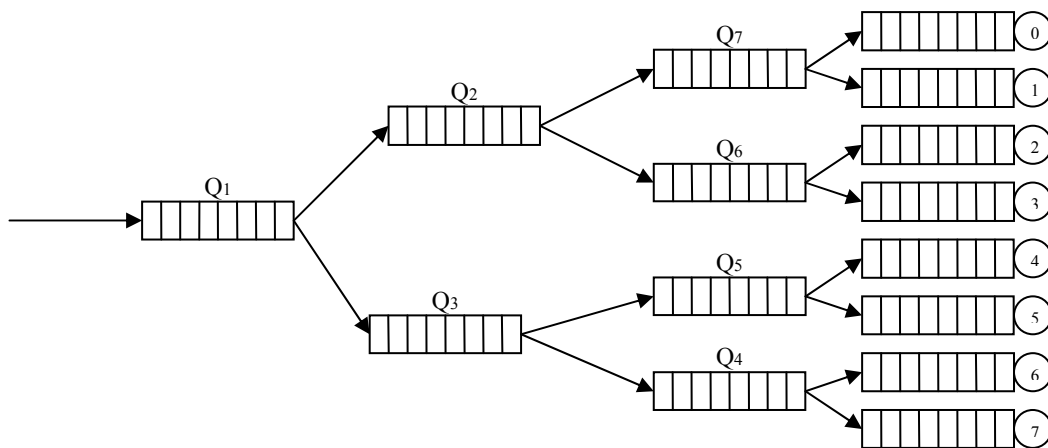


Figure 2 Hierarchical organization for N=8 processors with branching factor B=2

Each ready queue can be viewed as a ready queue in the centralized organization serving only the tree nodes directly below it; these nodes can themselves be ready queues or processors local queues. All incoming tasks are added to the root task queue. If L is the leaf node level, when processor is looking for work, it first checks its associated task queue at level (L-1). If that queue is empty it checks the parent node of this node at level (L-2) and the process is repeated up the tree until it finds a task to be scheduled (unless the root queue is empty).

In order to reduce access contention at higher levels, when a task queue is accessed, a set of tasks is moved one level down the tree – the size of set decreases progressively (at the bottom of the tree it is reduced just to one task). Parameter Tr is transfer factor which indicates the number of tasks transferred from a parent queue to its child queue.

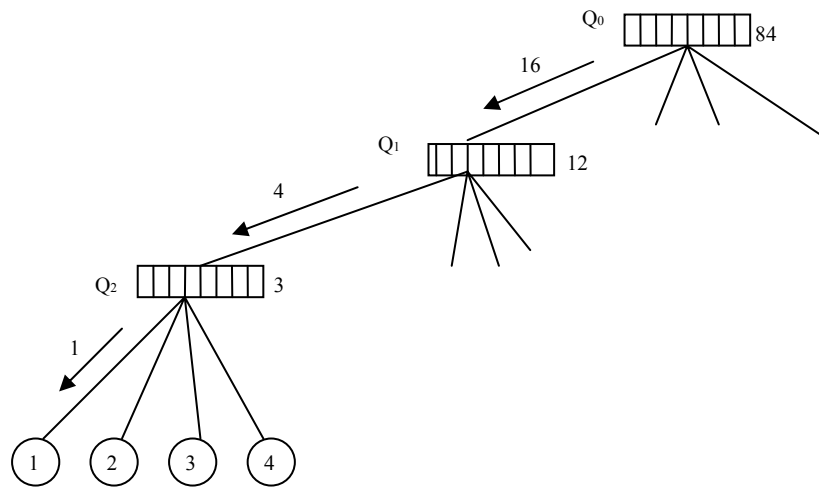


Figure 3 Task transfer process in the hierarchical organization for $N=64$ processors with branching factor $B=4$ and transfer factor $Tr=1$

Hierarchical organization avoids ready queue bottleneck because both the branching and transfer factors can be adjusted. It can also be seen that the set of task queues that form the tree can all be distributed to different memory modules so that concurrent access can be permitted (provided the interconnection network allows the particular permutation).

4. The modelling approach

In the following course work will be presented and analyzed performances of three different organizations for the shared-memory multiprocessor systems, mentioned upper.

4.1. Java Modelling Tools

JMT Simulator, also called as **JSIM**, is simulation module of the **Java Modelling Tools (JMT)**, an open-source fully-portable Java suite for performance evaluation, capacity planning and modelling of computer and communication systems. The suite implements numerous state-of-the-art algorithms for the exact, asymptotic and simulative analysis of queuing network models. JSIM is a discrete-event simulator for the analysis of queuing network models. An intuitive sequence of wizard windows helps specifying network properties. The simulation engine supports several probability distributions for characterizing service and inter-arrival times. Load-dependent strategies using arbitrary functions of the current queue-length can be specified. JSIM supports state-independent routing strategies, e.g., Markovian or round robin, as well as state-dependent strategies, e.g., routing to the server with minimum utilization, or with the shortest response time, or with minimum queue-length. The simulation engine supports several extended features not allowed in product-form models, namely, finite capacity regions (i.e., blocking), fork-join servers, and priority classes. The analysis of simulation results employs on-line transient detection techniques based on spectral analysis. What-if analyses, where a sequence of simulations is run for different values of parameters, are also possible.

4.2. Input parameters

We will consider a parallel system with N identical processors and model this system with an abstraction consisting of a set of queues and N sinks. For most results reported in this course work, we fixed N on 4 or 8 (partly because of the time needed to run simulation experiments and partly because many commercial multiprocessor systems use similar number of processors). Concentration is on the scheduling aspect of the system.

Generation of the jobs is a common feature among this three task queue organizations. However, contention for the task queue varies depending on the task queue organization. We will consider a simple **fork type of job structure**, assumed to be composed of a set of independent tasks that can be run on system concurrently. The job completes when all of its component tasks are completed. Tasks within the job do not communicate with each other. The **job arrival process** is assumed to be **exponentially distributed** with parameter $\lambda = 0.75$, and for what-if analysis we used range $[0.75, 1.50]$. Tasks are characterized by processor service demand with **mean $1/\mu$** , and $\mu = 1$.

We model the amount of time needed to access (not including the waiting time to get exclusive access to the ready queue) the ready queue as a **fraction f** of the average task execution time. We further assume that each time an idle processor accesses a ready queue it spends a constant amount of time (i.e., f/μ) independent of the actual number of entries in the ready queue i.e., ready queue access time is deterministic. This is true for simple scheduling algorithms such as the **first-come-first-served** (FCFS) where it simply involves removing the task at the head of the queue. Fraction $f=10\%$ causes the increasing of the service time, which we model with **constant distribution** with **coefficient 1.1**.

We determine our models also as FCFS, and **load independent**. **Routing of tasks** in the system is chosen to be **random**. **Queue length is finite** and fixed on **20**, and described models use **waiting queue, without dropping the tasks**.

Model is described as **one-open-class system**, which means that we will have only one class of the tasks arriving randomly into our system.

4.3. Performance analysis

To compare these three different organizations, we decided to make analysis over two important output parameters – **Response time** and **Utilization**, presenting the results in two sections:

- impact of access contention, and
- impact of system size.

Response time means average time spent by job before leaving the centre (queuing + in service). Utilization means average number of jobs in service.

Impact of access contention will be analysed and the results will be presented with the diagram describing the impact of the ready queue access time as a function of utilization. The analysis for the centralized organization will be done by varying the parameter λ implementing this in what-if analysis, and for the other two organizations, we will fix this parameter on 0.75, expecting the best result.

Impact of system size, will be analysed with the double size of the processors ($N=8$), so we will compare results for $N=4$ and $N=8$ processors in all three distributions. Results will be also presented with the diagrams describing job response time and utilization when the average task service time is doubled.

5. Simulations

First step is to build three different models for centralized, distributed and hierarchical organization in JSIM tool. They are presented in Figures 4, 5 and 6, respectively.

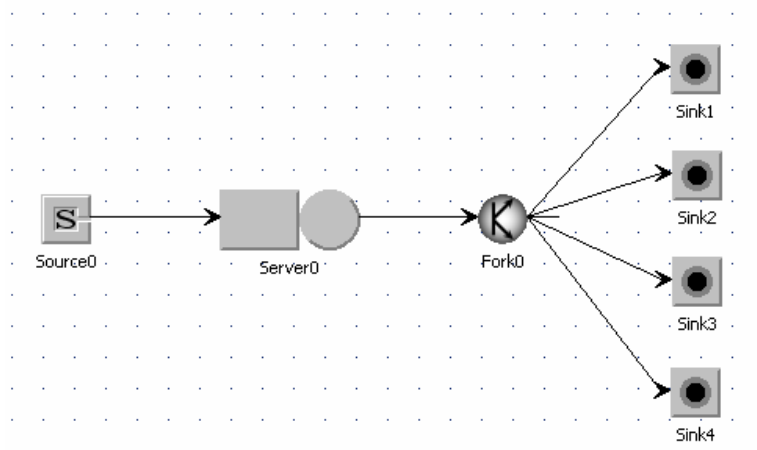


Figure 4 Centralized organization with $N=4$ sinks

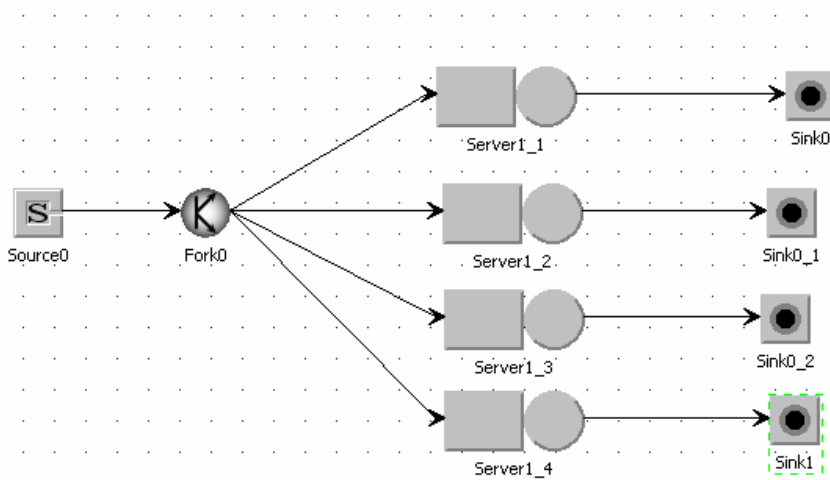


Figure 5 Distributed organization with $N=4$ sinks

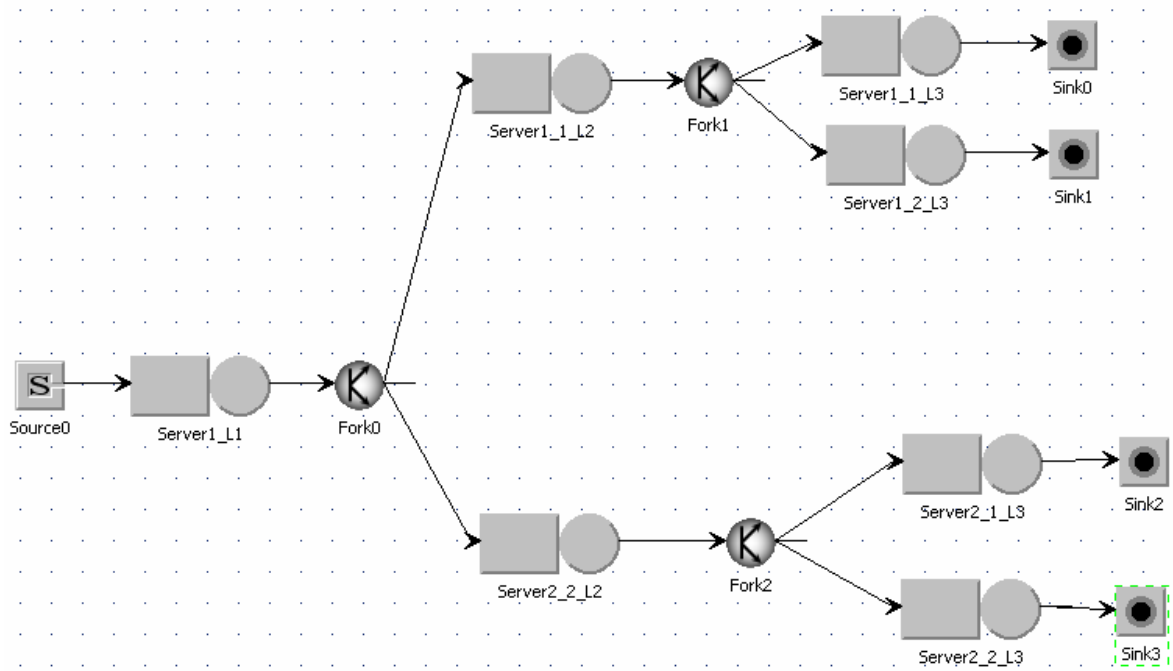


Figure 6 Hierarchical organization with N=4 sinks, Br=2, Tr=1

Second step is to give to all used blocks (source, servers and forks) needed values for the **job arrival distribution parameters** (exponential: $\lambda = 0.75$), **queue lengths** (20 tasks), **service time distribution parameters** (constant: $c=1.1$) and **simulation time** (6 seconds).

5.1. Impact of access contention

5.1.1. Centralized organization

For the centralized model we used **what-if analysis** choosing the **Arrival time** as the relevant parameter with λ in the range of [0.75,1.5] tasks/second (exponential distribution of the jobs). In this range we run simulations in four steps. Area of interest was to analyze response of the system from 4 to 8 tasks, regarding the number of sinks at the output of the model, because it was interesting to watch the performance for the same, or double number of tasks comparing how much we have sinks in the model (N=4).

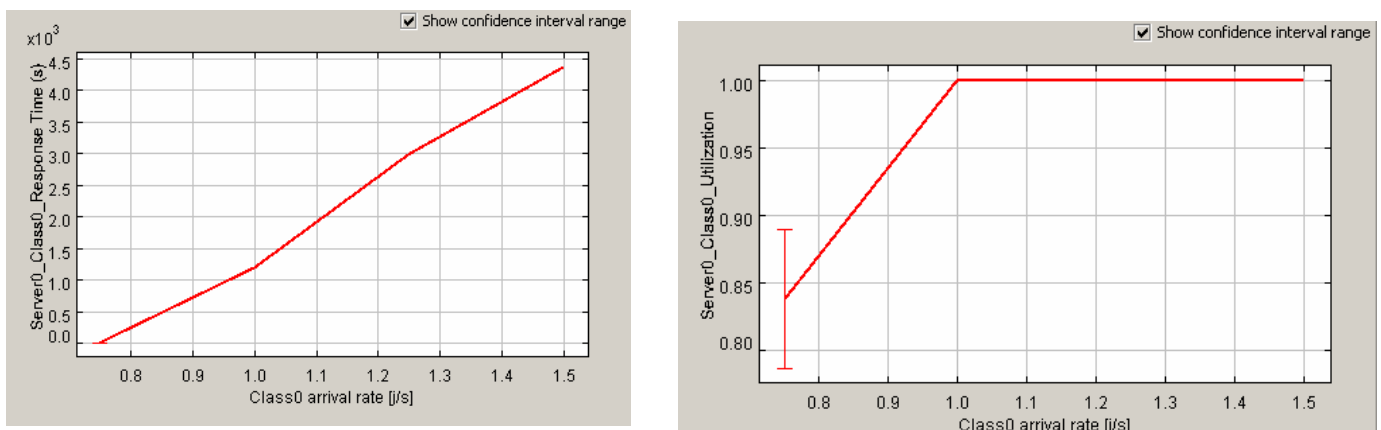


Figure 7 Centralized, what-if analysis, N=4 sinks

5.1.2. Distributed organization

For the distributed model we used simple analysis and simulated the model. Duration of the simulation was also 6 seconds. In Figure 8 on the left side is represented Response time, and on the right side is Utilization.

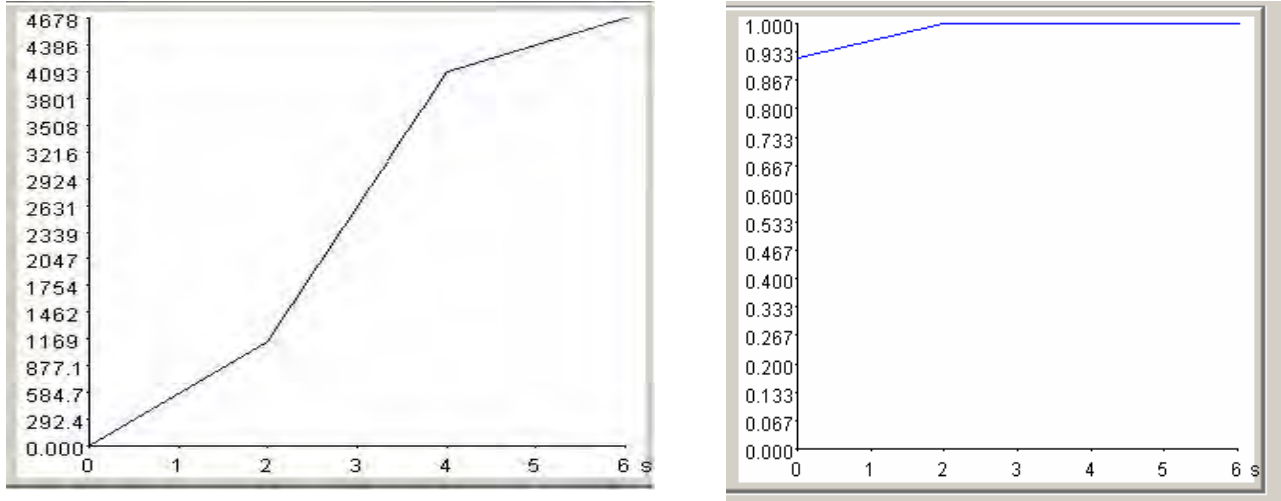


Figure 8 Distributed, simple analysis, N=4 sinks

5.1.3. Hierarchical organization

For the hierarchical model we used simple analysis and simulated the model. Duration of the simulation was 4 seconds long. We wanted to see how will model work for the similar number of the tasks, as sinks we have at the output; $4\text{sec} / (0.75\text{tasks}/\text{sec}) = 5.3$ tasks. In Figure 9 on the left side is represented Response time, and on the right side is Utilization.

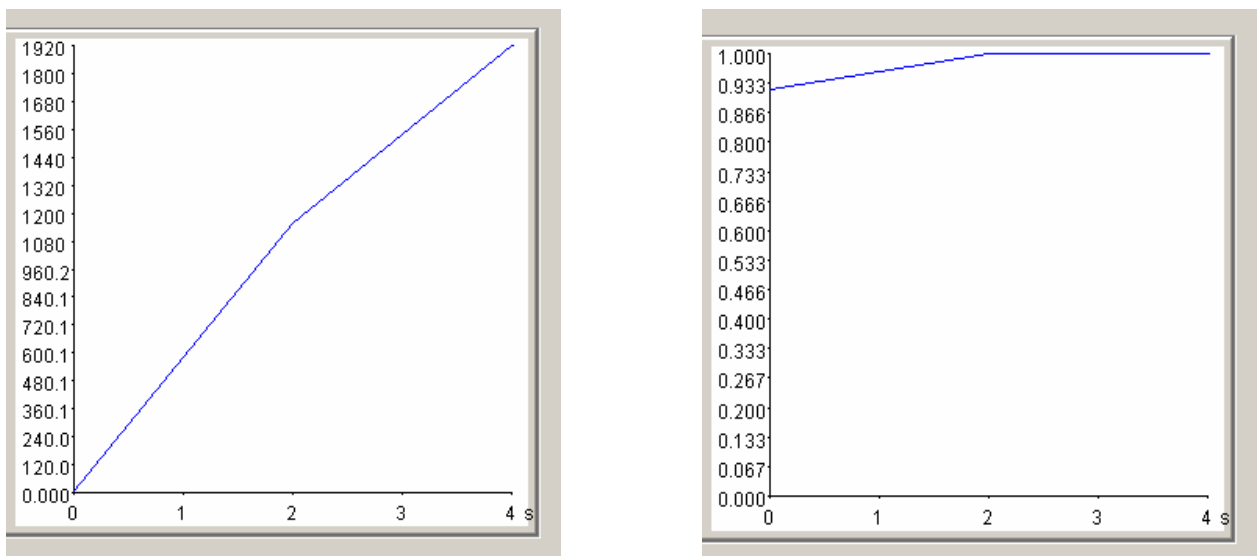


Figure 9 Hierarchical, simple analysis, N=4 sinks

5.2. Impact of system size

5.2.1. Centralized organization

For the centralized model with N=8 sinks at the output of the model, we used again **what-if analysis** choosing the **Arrival time** as the relevant parameter with λ in the range of [0.75,1.5] tasks/second, and run simulations in four steps. Area of interest was to analyze response of the system from 4 to 8 tasks, regarding the number of sinks at the output of the model, because now it was interesting to watch the performance for the half or the same number of tasks comparing how much we have sinks in the model (N=8).

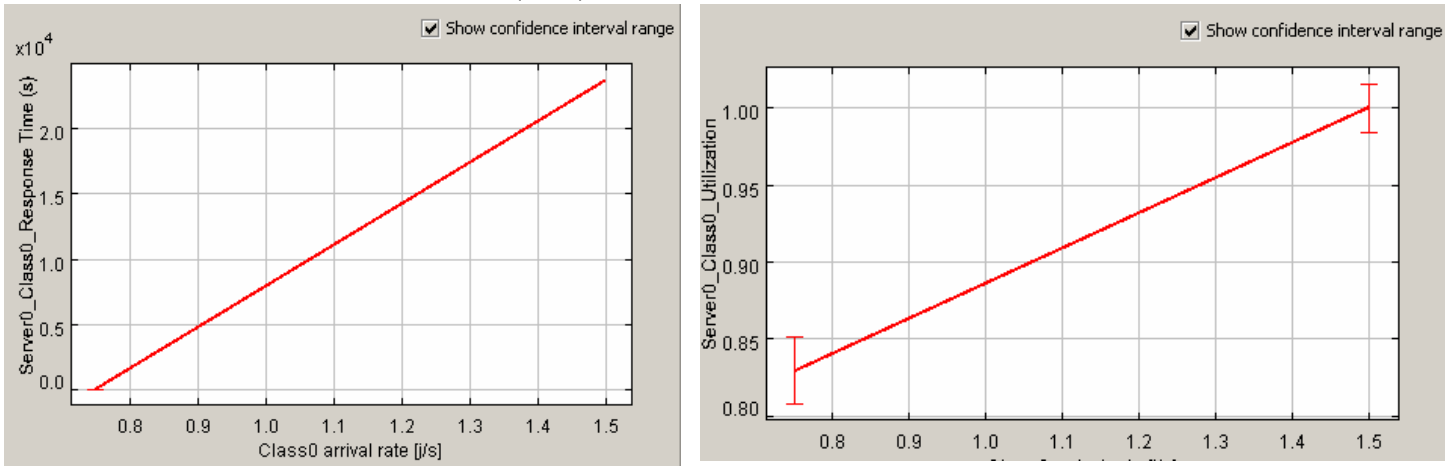


Figure 10 Centralized, what-if analysis, N=8 sinks

5.2.2. Distributed organization

For the distributed model with N=8 sinks at the output we used simple analysis and simulated the model. Duration of the simulation was 6 seconds, regarding the number of the tasks in the system, because we wanted to have the same number of tasks as sinks at the output; $6\text{sec} / (0.75\text{tasks/sec}) = 8$ tasks (N=8). In Figure 11 on the left side is represented Response time, and on the right side is Utilization.

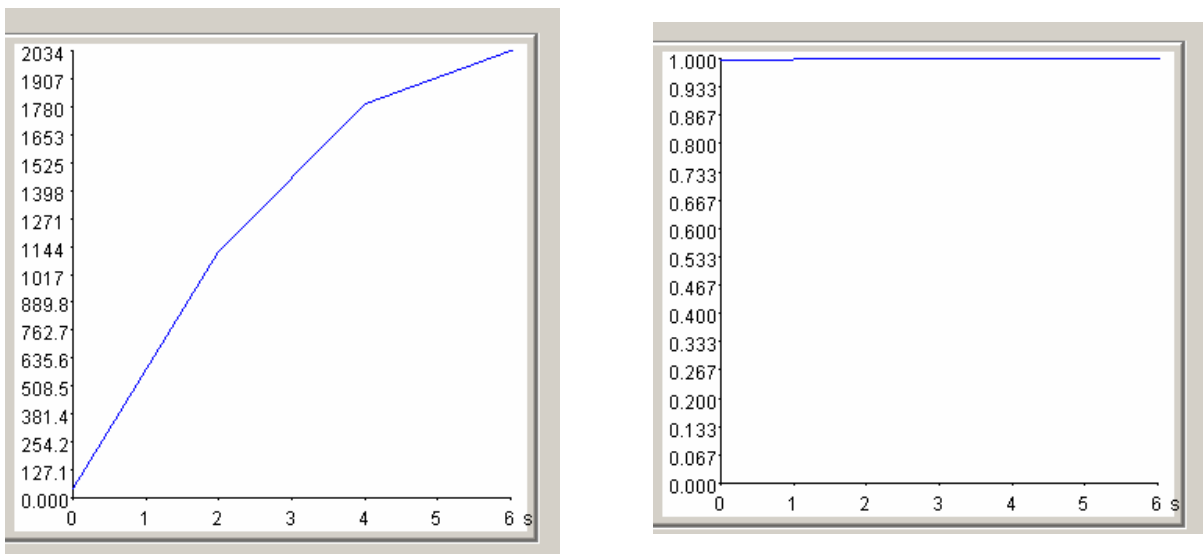


Figure 11 Distributed, simple analysis, N=8 sinks

5.2.3. Hierarchical organization

In Figure 12 is presented model of the hierarchical distribution with N=8 sinks. On this picture it is also possible to see the work-sheet of the JSIM tool.

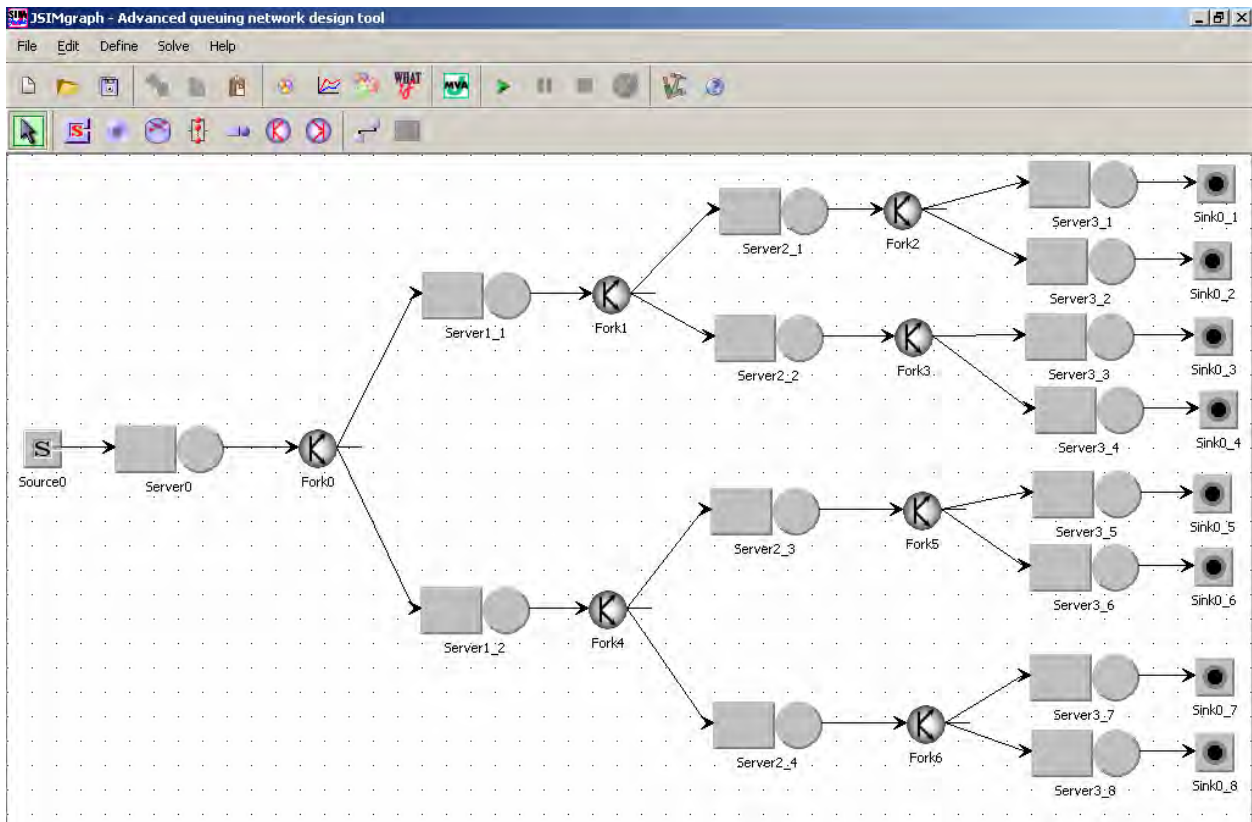


Figure 12 Hierarchical organization with N=8 sinks, Br=2, Tr=1

For this analysis we have chosen again simple analysis, 12 seconds long, because we wanted to compare how will this model work with double number of the tasks, comparing with the number of sinks in the model; $12\text{sec} / (0.75\text{tasks/sec}) = 16$ tasks, that means $2*N$. In Figure 13 on the left side is represented Response time, and on the right side is Utilization.

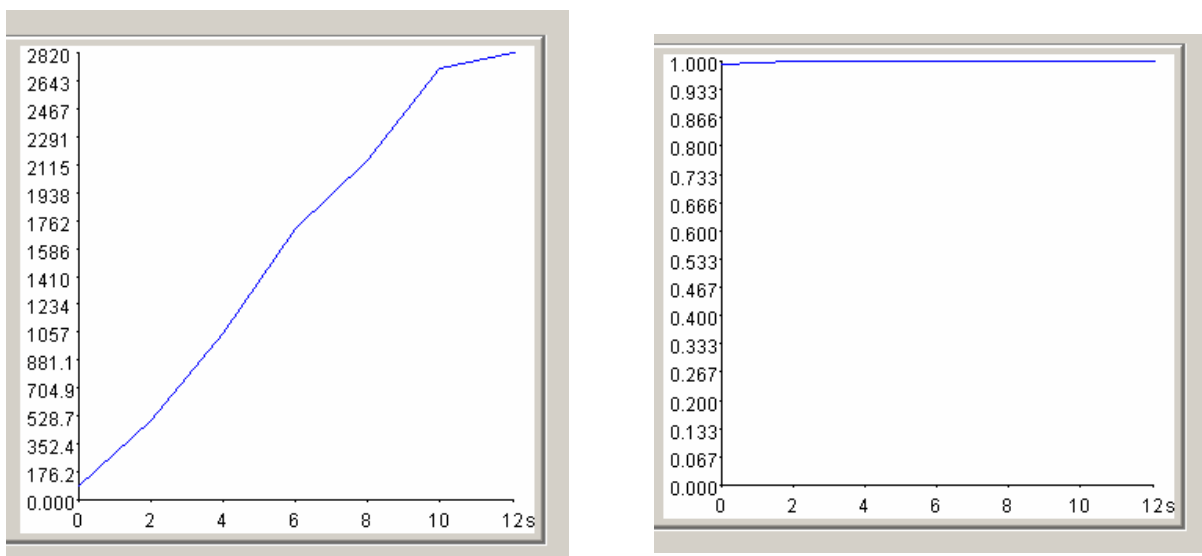


Figure 13 Hierarchical, simple analysis, N=8 sinks

6. Results & Conclusions

Comparing three different techniques for modelling waiting ready tasks in shared-memory multiprocessors, which were described, analyzed and simulated in this course work, we can conclude that centralized organization showed very good results for small number of processors. Response time was growing rapidly and utilization reached value of 1 in the short period. In organizing ready tasks, we can say that centralized task queue organization is preferred in the absence of access contention. However, the major drawback is that the central task queue becomes a bottleneck for large system sizes and therefore it is not suitable for large parallel systems or fine granularity task scheduling.

Introducing more processors in the system, centralized model was not good representative anymore, the utilization couldn't reach value of 1 in provided range for job arrival distribution, but on the other hand, distributed organization brought better results. With distributed organization we eliminated the task queue bottleneck disadvantage of the centralized organization by associating a local task queue for each processor. A main disadvantage of this organization is that there may be a load imbalance in the sense that some task queues may be empty while there are tasks waiting to be scheduled in other queues.

In the moment of making last simulations, and providing detailed performance analysis, we could conclude, finally, that in the both situations, with small and doubled number of processors (in the case of the JSIM model – sinks), **hierarchical approach is the best choice**. For $N = 4$ sinks at the output of the presented system, hierarchical model was good enough like the centralized, providing similar performances while eliminating the ready queue bottleneck. The properly designed hierarchical organization inherits the load sharing property of the centralized organization, while distributing the task queues as in the distributed organization. Mainly due to its load sharing property, the hierarchical organization exhibits less sensitivity to task service time variance like centralized organization in absence of access contention.

On the other hand, for $N = 8$ sinks hierarchical model was giving the best performances, even comparing with the distributed model. The results showed that both, hierarchical and distributed, organizations scale nicely for all system utilizations without creating system bottlenecks, but hierarchical approach retains its performance advantage over distributed one.

This new approach, hierarchical organization, presented in our course work, has also good characteristics in the case of larger number of the incoming jobs (or tasks) in the system, comparing with the number of the processors (in fact sinks) in the model.

Queue problems are optimized very well - implementing hierarchical organisation for modelling waiting ready tasks in shared-memory multiprocessors, problem of ready queue access contention and load imbalance problem now became negligible.

7. REFERENCES:

- [1] E. Lazowska, J. Zahorjan, *Quantitative System Performance - Computer System Analysis Using Queuing Network Models*, Prentice-Hall, Inc., New Jersey
- [2] S. Dandamudi, P. Cheng, *A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems*, IEEE transactions and distributed systems, vol. 6, No 1, January 1995
- [3] B. Nitzberg, V. Lo, *Distributed Shared Memory: A Survey of Issues and Algorithms*, University of Oregon, Oregon
- [4] B. Sinclair, *Analysis of Shared Memory Multiprocessors*, Version 2.3, United States, Jun 2005
- [5] M. Bertoli, G. Casale, G. Serazzi, *The JMT Simulator for Performance Evaluation of Non-Product-Form Queuing Networks*, Politecnico di Milano, Italy
- [6] *Tutored by G. Serazzi, Java Modelling Tools, users manual*, Version 0.3, Performance Evaluation Lab, Politecnico di Milano, Italy, June 2006
- [7] *Tutored by G. Serazzi, Java Modelling Tools, system manual*, Version 0.1, Performance Evaluation Lab, Politecnico di Milano, Italy, June 2006