Multiple Omega Networks for Parallel Processing[1]

José Eduardo Moreira

moreira@csrd.uiuc.edu
Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
305 Talbot Lab, 104S Wright St
Urbana, IL 61801
phone: 217-244-0052
fax: 217-244-1351

Laboratório de Sistemas Integráveis
Departamento de Engenharia Eletrônica
Escola Politécnica da Universidade de São Paulo
São Paulo, SP - Brazil

ABSTRACT

In this paper we propose the use of multiple Omega networks as an interconnection system for shared memory multiprocessors. This allows us to achieve a much higher bandwidth of communication, accommodating the needs of current high-performance processors, including those with multiple memory ports. We also obtain a very scalable system, by defining a processor-switch-memory building block, that can be used in systems with processor count in the range of a few units to several thousands. The performance evaluation of multiple Omega networks is done through a simple analytical model that allows us to compare their performance to a that of a single network, and investigate alternatives for processors with multiple memory ports. The results show that the performance (in terms of bandwidth and latency of communication) of systems with multiple networks is more stable with respect to variations in systems parameters, such as number of processors and memory access rate, than that of systems with just a single network.

[1]This work was supported in part by the National Science Foundation under grant NSF CCR 89-57310, the Department of Energy under grant DOE DE-FG02-85ER25001, and FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo).

1 Introduction

The function of the processor-memory interconnection in a shared-memory multiprocessor is to provide a logical link between any processor and any memory module. Many different organizations have been proposed and used for this interconnection. At the low and high ends of the bandwidth and cost spectrum we find the time-shared bus and the full crossbar switch, respectively. In between these two extremes, there is a rich variety of alternatives, which have been the subject of extensive research. A very popular class of interconnection networks, which has received considerable attention from both the industry and academia is the multistage interconnection network (MIN) [12]. These networks are composed of multiple stages of crossbar switches, connecting processors and memory modules. We will assume here that the reader is somewhat familiar with the general structure of MINs.

Most analysis of multistage interconnection networks (MINs) consider the situation where the number of processors, P, is equal to the number of memory modules, M ($P = M = N$), and the network has $\log_K N$ stages, with N/K switches (of type $K \times K$) each, for a total of $S = \frac{N}{K} \log_K N$ switches.

The approach described above has some disadvantages. One of them is that today's high-performance processors can require very high memory bandwidth (some even have multiple memory ports) and therefore the one-to-one ratio of memory to processor is not enough to provide the necessary bandwidth. Another disadvantage is that the ratio of processors to switches is not, in general, an integer ($\frac{N}{S} = \frac{K}{\log_K N}$ is an integer only for $N = K^n$ with $K \bmod n = 0$), so that we cannot have a homogeneous processor-switch-memory building block, very desirable for the construction of highly-parallel systems.

In this paper we propose and analyze a multiple interconnection scheme that allows us to build shared memory parallel systems out of homogeneous elements, and provides adequate bandwidth for memory operations by advanced processors. This scheme is composed of multiple Omega (also known as multistage shuffle-exchange) networks operating in parallel.

We start the paper with a brief description of the Omega network. We then explain how multiple Omega networks can operate in parallel, and how a system can be built utilizing homogeneous processor-switch-memory elements. We proceed with some performance evaluation of multiple Omega networks, showing their advantage over single networks, and conclude with some discussion of related work.

2 The Omega Network

An Omega (or multistage shuffle-exchange) network [6, 17] can be described by a pair of integers $\Omega = (N, K)$, where N is the number of input and output ports to the network, and K is the *radix* of the network. We only consider the simplest case, $N = K^n$. In this case, the Omega network consists of n stages, each stage, in turn, is composed by a ***shuffle*** substage and an ***exchange*** substage.

The shuffle substage is simply a reordering of the inputs, obtained by applying the shuffle function of radix K ($S_K(i)$), defined as follows: let $i = a_{n-1}a_{n-2}\ldots a_1 a_0$ be an input number in K-radix representation ($a_{n-1}\ldots a_0$ are K-ary digits), then:

$$S_K(a_{n-1}a_{n-2}\ldots a_1 a_0) = a_{n-2}\ldots a_1 a_0 a_{n-1}$$

This corresponds to a simple cyclic shifting of the number. In the more interesting case that K is a power of 2, $K = 2^k$, each K-ary digit is a set of

k binary digits, and a circular shift on the address corresponds to a k-position binary circular shift.

The exchange substage consists of N/K $K \times K$ crossbar switches. The first crossbar is connected to the first K inputs and outputs, the second crossbar to the following K inputs and outputs, and so on. Figure 1 shows an $\Omega = (16, 4)$ network. Typically, two unidirectional Omega networks are used to build a system: one (forward network) goes from the N processors to the N memory modules, while the other (reverse network) goes from the memory modules back to the processors, as shown in Figure 2.

Routing in an Omega network is easily accomplished. We first label each stage, starting with $n-1$ for the stage closest to the input, and decreasing by one as we move down the network, until we label with 0 the stage closest to the output. Suppose now that a request entering the network at input $i = a_{n-1}a_{n-2}\ldots a_1a_0$ wants to reach output $o = b_{n-1}b_{n-2}\ldots b_1b_0$. The message then enters the network through stage $n-1$. When this message reaches the exchange substage of stage l, it then selects output b_l of the crossbar switch. At the last stage of the network (stage 0), this request will be emerging at the desired output.

Some of the important parameters of a shared-memory multiprocessor system built around an Omega network are the number of processors P, the number of memory modules M, and the number of switches S, in each of the networks (forward and reverse). From the way the network is constructed, we have:

$$P = M = N$$

$$S = \frac{N}{K}n = \frac{N}{K}\log_K N$$

Therefore, the ratio of processors to switches is $P/S = K/log_K N$. Today's reasonable values for K are 8, 16 and 32, and the above ratio is larger than 1 even for system with thousands of processors. Table 1 shows the processor-switch ratio (and other parameters) for some systems utilizing an Omega network. We only list those systems for which P/S is an integer. These systems have the interesting property that they can be built utilizing a single type of building block: a processor-switch-memory element composed of two switches (one for the forward and the other for the reverse network), P/S memory modules and P/S processors. The systems are not scalable, though, because the P/S ratio (an therefore the building block) is different for each system size.

Since a high performance processor is usually the most expensive and most difficult to design component in the system, we would like to have a lower processor/switch ratio, if this buys us some additional performance (and indeed it does, as we will show latter). We also want to have a building block that can be used for systems of different sizes. We recognize that the general case of systems that can be built utilizing homogeneous elements composed of p processors, s switches and m memory modules deserves attention, but in this paper we only treat the particular case of systems utilizing elements with 1 processor, 2 switches (1 for the forward network, and 1 for the reverse network), and m memory modules.

We also only consider systems utilizing *complete* networks: those that use all the inputs and outputs. Systems with a varied number of processors can be built by utilizing only a subset of a complete network (we call these *incomplete* networks). The Cedar multiprocessor [5], for instance, utilizes a 2-stage Omega network of 8×8 switches. Normally, this network would have 64 inputs and outputs, but Cedar only utilizes a subset of it to connect 32 processors to 32 memory modules.

3 Multiple Omega Networks

3.1 Performance Motivation

An important performance drawback of the Omega network is that the number of inputs and outputs of the network is the same. This can cause considerable conflicts in the routing of requests. Let us assume that each input (of the whole network or of a single switch) can accept a new request every cycle. We then say that a *conflict* occurs whenever two or more inputs want to reach the same output at the same time, and we define *throughput* (of the network or switch) as the number of requests that reach the output in each cycle, divided by the number of inputs (the maximum throughput is therefore 1.0).

Independently of the configuration of a particular Omega network, it cannot produce more throughput than a crossbar. If we assume uniformly distributed random selection of outputs by the inputs, and always one request per input every cycle, the throughput of a $N \times N$ crossbar approaches the limit of $1-e^{-1} = 0.632$ as N approaches ∞ (for a 16×16 crossbar the value is 0.644) [7]. The Omega network cannot do better than this, and as the number of stages in the network increases, the throughput ratio between the Omega and the crossbar decreases. See Figure 8 for a plot of these throughputs (the formulas used to obtain these figures are derived in section 4).

The limiting throughput (as $N \rightarrow \infty$) of m $N \times N$ crossbars operating in parallel to connect N processors to mN memories is given by $m(1-e^{-1/m})$ [7], notice that m $N \times N$ parallel crossbars is essentially the same as one $N \times mN$ crossbar. This is an upper limit for the throughput of multiple Omega networks in parallel, and Figure 9 is a plot of this limit for different values of m. As we can see, the use of multiple interconnection networks can substantially increase the bandwidth between processors and memories.

In view of the above discussion, interconnection networks for shared memory multiprocessors need to provide a memory/processor ratio better than one. In fact, since high-performance processors may have multiple memory ports, the ratio of memory/processor has to be larger than p, where p is the number of memory ports per processor. This increased bandwidth between processor and memory become even more important as the speed of memories and switches fails to keep up with the speed improvement of processors.

3.2 Using Multiple Networks

One means to achieve a higher memory/processor ratio is to use multiple Omega networks in parallel. We will show how this can be done, with the additional benefit of making the ratio of number of switches to number of processors fixed for a wide range of system sizes. This implies that a processor-switch-memory element can be used as a building block for highly-parallel systems.

Table 1 shows the processor/switch ratio for various Omega networks, this ratio is $K/\log_K N$. If we provide two switches (one to be used in the forward network and the other in the reverse network) for every processor in the system, that means we can build $K/\log_K N$ Omega networks in parallel, thus increasing the total throughput between processor and memory. We will also provide K memory modules per processor, reducing the probability of conflicts in memory accesses, and compensating for the difference in speed between processors and memories. The next paragraphs will show how to build systems using these components.

A set of K memory modules is grouped together into a *supermodule*. A supermodule has K inputs and K

outputs, and any input can access any memory module, and any memory module can access any output. This can be accomplished by using $K \times K$ crossbar switches between the inputs and memory modules, and between the memory modules and outputs, as shown in Figure 3. In the case of a system with $P = M$, each memory module is usually divided into multiple banks (which may be capable of performing accesses simultaneously, in order to compensate for the long memory cycle time), which would roughly correspond in size to the independent modules in the case of multiple networks (the total memory size should be approximately the same in both cases).

Let us first consider the case of one memory port per processor. The system then consists of N processors, $m = K/\log_K N$ Omega networks of the type $\Omega = (N, K)$, and $M = NK$ memory modules. Each Omega network has $\frac{N}{K}\log_K N$ switches, and therefore the total number of switches in the system is $2N$ (N for the forward networks, and N for the reverse networks). The system is built as follows:

1. We first build m Omega-networks of type (N, K), and label them $\Omega_0, \Omega_1, \ldots, \Omega_{m-1}$ (this is done both for the forward and reverse networks).
2. To the outputs of the forward networks and inputs of the reverse networks we connect the supermodules. There are N supermodules (KN total memory modules), and m networks (of each type) with N connections per network. Output i of forward network j is connected to input j of supermodule i, and output k of supermodule l is connected to input l of network k, for $0 \leq i, k \leq N-1$, and $0 \leq j, l \leq m-1$.
3. Using a $1 : m$ demultiplexer, we connect processor P_i to the the i^{th} input of all m forward networks $\Omega_0, \Omega_1, \Omega_{m-1}$, and using a $m : 1$ multiplexer we connect the i^{th} output of all m reverse networks to P_i.

See Figure 4 for an example of how two Omega networks are used in parallel to connect 16 processors to 64 memory modules.

Each of the supermodules is connected to the m networks (forward and reverse). Notice that m inputs and outputs of each supermodule are used ($1 \leq m \leq K$). We partition the modules inside a supermodule by exclusively assigning module i to network $i \bmod m$. This guarantees that there will be no conflicts in the crossbar switches inside a supermodule.

For a processor to access a memory module, it must now first select the appropriate network, and then the message must be routed through the network. Since we have essentially increased the bandwidth of the connection between processor and memory by using m Omega networks in parallel, the chances for a conflict are smaller than using a single Omega network. Figures 10 to 15 compare the performance (in terms of network bandwidth and delay) of multiple Omega networks to that of a single network. Again, the formulas used to obtain these figures are derived in Section 4.

The building block shown in Figure 5 can be used to build systems with K, K^2, ..., K^K, processors. The ratio of memories and switches to processors is kept fixed for all the system sizes, and in this sense the system is very scalable. The *connections* between components, though, is not fixed, and different system sizes require different wirings (which is not a scalability feature). The performance of the interconnection system does not scale perfectly, since the number of parallel networks decreases as the number of processors increases.

Table 2 shows some characteristics of systems with 8, 16, 32, 64, 256, 1024, 4096, 65536 and 1048576 processors with a maximum of 4 stages in the network, built using switches of size 8×8, 16×16 and 32×32 (the systems with more practical interest have 8 to 4096 processors).

3.3 Processors with Multiple Memory Ports

Some high-performance processors need more than a single port to memory, with 4 being a more reasonable number. We can accommodate such processors by using a $4 \times m$ crossbar between the processors and the networks, as shown in Figure 6. This crossbar we added is now a source for contentions, and affects the performance of the system. Another option is to treat each physical processor as 4 *virtual* processors, giving to each of the memory ports of the processors an input in *all* parallel networks, as shown in Figure 7. Figures 16 and 17 compare the performance of these two approaches, using expressions derived in the next section.

4 Performance Evaluation

The basic expressions used in this paper for the performance of a crossbar switch can be found in [7]. The performance model used here for multistage interconnection networks is very similar to that of Patel [8]. More complete analytical performance models can be found in [4, 9].

A basic assumption for the derivation of our performance models is that the processors generate uniformly distributed random requests for memory modules. That means that in a request a processor requests a given memory module with probability $1/M$. Furthermore, all processors generate requests at the same rate, λ ($\lambda \leq 1.0$ requests/cycle).

4.1 Performance of a Crossbar Switch

Consider a $n \times m$ crossbar switch. In each cycle it receives requests from its n inputs, and routes as many requests as possible to its m outputs, requests that cannot be routed to the desired output in a given cycle can be dropped or (more realistically) stored in a FIFO queue for that input, so that they can retry latter (for our analysis we are assuming that retries are performed, but we ignore the effect of the retries in increasing the request rate). Define r_i as the *input* bandwidth for input i, *i.e.* the average number of requests received at input i in each cycle. Define b_i as the *output* bandwidth for output i, *i.e.* the average number of requests reaching output i in each cycle. If $r = r_0 = r_1 = \ldots = r_{n-1}$ and each input requests all the outputs with same probability, then $b_0 = b_1 = \ldots b_{m-1} = b$, and

$$b = \left[1 - \left(1 - \frac{r}{m}\right)^n\right] \tag{1}$$

The total bandwidth of the crossbar is mb (b can also be defined as the *normalized* bandwidth, since it is the total bandwidth divided by the number of outputs).

The efficiency of a $m \times n$ crossbar switch can be defined as

$$e = \frac{mb}{nr} = \frac{m}{rn}\left[1 - \left(1 - \frac{r}{m}\right)^n\right] \tag{2}$$

The efficiency e is the probability that a particular request from an input will be satisfied in a given cycle, therefore, each request experiences, on the average, a delay of

$$d = \frac{1}{e} \tag{3}$$

cycles in the crossbar.

If the crossbar is of type $K \times K$ the above expressions for bandwidth and efficiency become

$$b = \left[1 - \left(1 - \frac{r}{K}\right)^K\right] \tag{4}$$

$$e = \frac{1}{r}\left[1 - \left(1 - \frac{r}{K}\right)^K\right] \quad (5)$$

4.2 Performance of Omega Networks

After these initial considerations, we can proceed with the modeling for single and multiple Omega networks. We will only analyze the performance of the forward network (processors to memories), since we believe this is enough for a comparison between single and multiple networks. We also ignore the effects of the initial demultiplexer (connected to the processor), since this is not necessarily an active device. It may appear that for the case of multiple networks there is an additional switch hidden inside the supermodule, but in fact, this switch adds roughly the same delay as the circuitry in a single memory module with multiple banks.

Consider an Omega network $\Omega = (N, K)$. It has, as discussed before, $n = \log_K N$ stages. The stages are numbered $n-1, n-2, \ldots 0$. Let r_l be the input bandwidth for the switches in stage l, and b_l be the output bandwidth for the same switches. Then the input bandwidth for the whole network is:

$$r = r_{n-1} = \lambda \quad (6)$$

and the output bandwidth for the whole network is:

$$b = b_0 \quad (7)$$

and, since the outputs of stage l are the inputs for stage $l-1$, we have:

$$r_{l-1} = b_l \quad (8)$$

Define the function $F(x, K)$ as

$$F(x, K) = \left[1 - \left(1 - \frac{x}{K}\right)^K\right] \quad (9)$$

and let

$$F^0(x, K) = x \quad (10)$$

$$F^i(x, \cdot) = F(F^{i-1}(x, K), K) \quad i \geq 1 \quad (11)$$

then

$$b_l = F(r_l, K) = F^{n-l}(r, K) = F^{n-l}(\lambda, K) \quad (12)$$

and

$$b = b_0 = F^n(r, K) = F^n(\lambda, K) \quad (13)$$

where $n = \log_K N$, r is the (normalized) input bandwidth of the network, and b is the (normalized) output bandwidth.

The total delay in the network is the sum of the delays in each stage. The delay in stage l is

$$d_l = \frac{1}{e_l} = \frac{r_l}{b_l} = \frac{F^{n-l-1}(\lambda, K)}{F^{n-l}(\lambda, K)} \quad (14)$$

and the total delay in the network is

$$d = \sum_{l=0}^{l=n-1} \frac{F^{n-l-1}(\lambda, K)}{F^{n-l}(\lambda, K)} \quad (15)$$

4.3 Performance of Multiple Networks

In the case of m parallel networks ($m = K/\log_K N$), we essentially divide the requests from each processor into m equal parts, one for each network. Therefore, for each network:

$$r = \frac{\lambda}{m} \quad (16)$$

$$b = F^n\left(\frac{\lambda}{m}, K\right) \quad (17)$$

$$d = \sum_{l=0}^{l=n-1} \frac{F^{n-l-1}\left(\frac{\lambda}{m}, K\right)}{F^{n-l}\left(\frac{\lambda}{m}, K\right)} \quad (18)$$

The m parallel networks have m times more outputs than a single Omega network, therefore to compare the bandwidths of both approaches we introduce a new parameter, B, the normalized *network* bandwidth:

$$B = b, \quad \text{for a single network} \quad (19)$$

$$B = mb, \quad \text{for multiple networks} \quad (20)$$

B is the bandwidth the network provides for each processor, and therefore it is a fair measure of performance. Note that $0 \leq B \leq 1$.

Figures 10 to 12 compare the bandwidth of single and multiple Omega networks for switches of size 8×8, 16×16 and 32×32, respectively, for $\lambda = (1.00, 0.50)$ and various numbers of processors. We observe that for all cases the bandwidth of multiple Omega networks is better than that of a single network, however the difference is larger for heavier traffic. This indicates that the use of multiple Omega networks is justified for systems with high performance processors, that have higher memory bandwidth requirements. The difference in performance is also higher for systems with larger switches, this is expected since for the same number of network stages, larger switches mean more networks in parallel (see Table 2). Very important is the fact that the normalized bandwidth is much less sensitive to the number of processors, in the case of multiple networks. This is a good scalability and stability feature.

Figures 13 to 15 compare the network delay of single and multiple Omega network for switches of size 8×8, 16×16 and 32×32, respectively, for $\lambda = (1.00, 0.50)$ and various numbers of processors. We observe that the delay through multiple networks is less than through a single network, a direct consequence of the increased bandwidth. We also observe that the delay through multiple networks is less sensitive to the traffic intensity than the delay through a single network. Again, as it was for the bandwidth, this is a desirable property, since it makes the whole system less sensitive to this parameter, relieving, at least in part, the worries of the user.

4.4 Performance for Processors with Multiple Memory Ports

The above expressions for network bandwidth and delay are also valid for the case of processors with multiple memory ports, if we use the virtual processor approach and define λ as the request rate *per port* (We obviously have to scale the x-axis in Figures 10 to 15 by the number of ports per processor). If we use the crossbar approach, we then have to take into account the effect of the crossbar at the beginning of the network. For a processor with 4 memory ports, this is a $4 \times m$ crossbar, and its efficiency is given by (again, λ is the request rate per processor memory port)

$$e_x = \frac{m}{4\lambda}\left[1 - \left(1 - \frac{\lambda}{m}\right)^4\right] \tag{21}$$

and the delay through this crossbar is

$$d_x = \frac{1}{e_x} \tag{22}$$

The normalized bandwidth through this crossbar, which is equal to the input rate of the parallel Omega networks, is

$$r = b_x = \frac{1}{\lambda}\left[1 - \left(1 - \frac{\lambda}{m}\right)^4\right] \tag{23}$$

and, using the previous expressions, the total network normalized bandwidth is

$$b = F^n(r, K) \tag{24}$$

$$B = mb \tag{25}$$

and the total delay through the network is

$$d = d_x + \sum_{l=0}^{l=n-1} \frac{F^{n-l-1}(r, K)}{F^{n-l}(r, K)} \tag{26}$$

Figures 16 and 17 compare the network bandwidth and delay (respectively) for the virtual processor and crossbar approaches. The virtual processor approach uses 4 network inputs per processor, and therefore, using the networks we discussed, systems with 2, 4, 8, 16, 64, 256, 1024, 16384 and 262144 processors can be built. The crossbar approach does not change the number of processors in the systems that can be built, *i.e.* 8, 16, 32, 64, 256, 1024, 4096, 65536, 1048576. Notice that systems with 8, 16, 64, 256 and 1024 processors can be built using both approaches.

In general we would expect the virtual processor approach to deliver better performance, since it provides more replication, and avoids an extra crossbar stage. However, for these particular systems that we are considering, the crossbar approach delivers better bandwidth for systems with 1024 processors, and better delay for systems with 16 and 1024 processors. This is explained as follows: for systems with a 1024 processors, the virtual processor approach uses 2 Omega networks of type (4096,8) in parallel, which have 4 stages of switches, while the crossbar approach uses 16 Omega networks of type (1024,32), which have only 2 stages. This more than compensates for the extra crossbar stage and the replication in the virtual processor approach. Similar considerations are also valid for the systems with 16 processors.

5 Related Work

Tang and Mendez [16] have identified the efficiency of data transfer between processor and memory as a limiting factor in the performance of vector computers, and they concluded that the number of memory modules in a system should be proportional to the product of memory access ports and the memory cycle time (in units of processor cycle time).

Szymanski and Fang [15] have analyzed several configurations of switches and banyan networks, and compared the performance of a single network to that of multiple parallel networks (with equivalent total cost) and concluded that single networks perform better for small systems and light loads, while parallel networks are better for larger systems or heavier traffic.

Smith et al. [13] discuss the need for supercomputers to support scalability, and recognize that the dominant scalability problem is the support of shared memory for multiple processors and memory ports. One of the solutions they propose is the extension of multistage interconnection networks such as used in the Cray Y-MP, that in this particular case connects 8 processor to 256 memory banks. Smith and Taylor [14] do a performance analysis of such networks, which have a *fanout*, since they connect a number of processors to a larger number of memory modules. This fanout can be either *wide* (fanout at the beginning of the network, processor side) or *narrow* (fanout at the end of the network, memory side). The authors' results show that wide fanout gives better performance than narrow fanout, and that an equal number of memories and (total) processors ports is of little value for supercomputer design.

Shing and Ni [11] address the problem of memory and interconnection network contention by essentially time-multiplexing physical resources (network switches and memory modules). Each user of a physical resource has a designated time slot in which it can use the resource.

Robbins and Robbins [10] solution to increase the efficiency of shared memory systems involves no change in the interconnection network, but only in the memory system. Each physical bank, in a system such as the Cray Y-MP, is replaced by a logical bank consisting of a number of physical banks. The authors claim that such change allows a Cray Y-MP - like system to scale up to 64 processors.

Franklin and Dhar [2] present some considerations on physical constraints and modularity issues in the design of a large (2048×2048) interconnection network.

Andrews, Beckmann and Poulsen [1] have developed some networks that provide efficient cache coherence schemes for systems with hundreds and thousands of processors. Since the use of caches reduces the memory bandwidth required by processors, this is another solution for the problem of providing enough memory bandwidth.

Hsu and Yew [3] propose the use of hierarchical (clustered) systems to reduce the bandwidth requirements of interconnection network. and show that this is particularly important in face of packaging constraints.

6 Conclusion

We have shown how multiple Omega networks can be used to build a shared memory multiprocessor that has a higher bandwidth between processors and memories and is also highly modular, and scalable from systems with a few processors to thousands of processors. The use of multiple Omega networks allows us to accommodate even the needs of high-performance processors with multiple memory ports.

Current high-performance RISC microprocessors run at speeds of 50, 100 and even 200 MHz (DEC Alpha), some already have multiple memory ports (Motorola MC88000 and Texas Instrument TMS320C40). Each of these processors requires enormous memory bandwidth, and if we are going to build shared memory multiprocessors with hundreds or thousands of these processors (to surmount the Teraflop performance barrier, while keeping the *flat memory* programming model), high-performance interconnection between processors and memories will be a key issue for the success of this enterprise. We believe that the use of multiple Omega networks operating in parallel is a viable alternative to the construction of such systems, since it provides the necessary bandwidth, keeps the delay through the network within acceptable values, and has the scalability properties that are essential for the design of massively parallel systems.

As future work, we plan to generalize the topics in this paper to networks built with homogeneous elements of p processors, s switches and m memory modules, considering the use of *incomplete* networks and investigating the optimization of networks in the cost-performance space. We also plan to study the system and application level performance of shared-memory multiprocessors that utilize multiple Omega networks.

Acknowledgement

The author wishes to thank Carl Beckmann for some valuable comments.

References

[1] John B. Andrews, Carl J. Beckmann, and David K. Poulsen. Notification and Multicast Networks for Synchronization and Coherence. *Journal of Parallel and Disrtributed Computing*, (15), 1992.

[2] Mark A. Franklin and Sanjay Dhar. On Designing Inteconnection Networks for Multiprocessors. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 208–215, St. Charles, Illinois, 1986.

[3] William T. Hsu and Pen-Chung Yew. The Performance of Hierarchical Systems with Wiring Constraints. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages I:9–16, St. Charles, Illinois, 1991.

[4] Young Man Kim and Kyungsook Y. Lee. Performance Analysis of Buffered Banyan Network under Nonuniform Traffic. In *Proceedings of the 1989 International Conference on Supercomputing*, pages 452–460, Crete, Greece, June 5–9, 1989.

[5] J. Konicek et al. The Organization of the Cedar System. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages I:49–56, St. Charles, Illinois, 1991.

[6] Duncan H. Lawrie. Access and Alignment of Data in an Array Processor. *IEEE Transactions*

on Computers, C-24(12):1145–1155. December 1975.

[7] T. N. Mudge and B. A. Makrucki. Probabilistic Analysis of a Crossbar Switch. In *Proceedings of the 9th Annual International Symposium on Computer Architecture*, pages 311–319, Austin, Texas, April 26–29, 1982.

[8] Janak H. Patel. Performance of Processor - Memory Interconnections for Multiprocessors. *IEEE Transactions on Computers*, C-30(10):771–780, October 1981.

[9] Mahib Rahman and David G. Meyer. General Analytic Models for the Performance Analysis of Unique and Redundant Path Interconnection Networks. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages I:584–591, St. Charles, Illinois, 1991.

[10] K. A. Robbins and S. Robbins. Bus Conflicts for Logical Memory Banks on a Cray Y-MP type Processor System. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages I:21–24. St. Charles, Illinois, 1991.

[11] Honda Shing and Lionel M. Ni. A Conflict-Free Memory Design for Multiprocessors. In *Proceedings of Supercomputing'91*, pages 46–55, Albuquerque, New Mexico, November 18–22, 1991.

[12] Howard Jay Siegel and William Tsun yuk Hsu. *Computer Architecture - Concepts and Systems*, chapter Interconnection Networks. North-Holland, 1988.

[13] J. E. Smith, W.-C. Hsu, and C. Hsiung. Future General Purpose Supercomputer Architectures. In *Proceedings of Supercomputing'90*, pages 796–804, New York, New York, November 12–16, 1990.

[14] J. E. Smith and W. R. Taylor. Accurate Modeling of Interconnection Networks in Vector Supercomputers. In *Proceedings of the 1991 International Conference on Supercomputing*, pages 264–273. Cologne, Germany, June 17–21, 1991.

[15] Ted Szymanski and Chien Fang. Design and Analysis of Buffered Crossbars and Banyans with Cut-Through Switching. In *Proceedings of Supercomputing'90*, pages 264–273, New York, New York, November 12–16, 1990.

[16] Peixiong Tang and Raul H. Mendez. Memory Conflicts and Machine Performance. In *Proceedings of Supercomputing'89*, pages 826–831, Reno, Nevada, November 13–17, 1989.

[17] Chuan-Lin Wu and Tse-Yun Feng. The Universality of the Shuffle-Exchange Network. *IEEE Transactions on Computers*, C-30(5):324–332, May 1981.

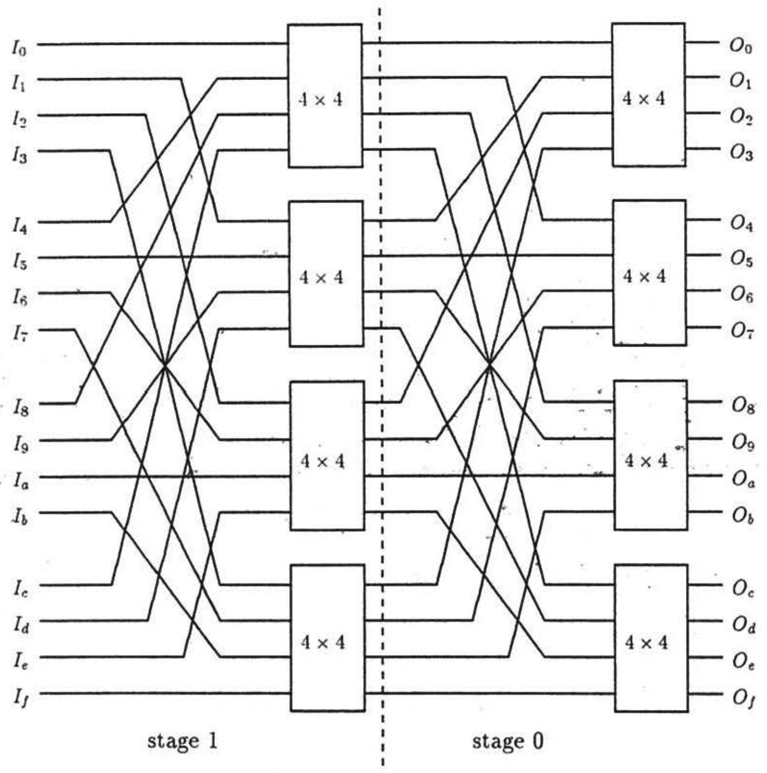

Figure 1: An Omega network of the type $\Omega = (16, 4)$. This network has 16 inputs, 16 outputs, and 2 stages of 4-way shuffle and 4×4 switches.

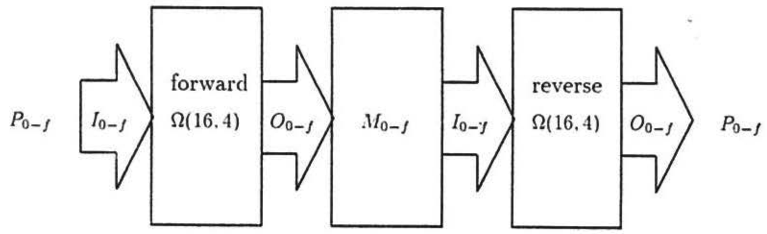

Figure 2: Typical use of 2 unidirectional Omega networks to build a shared-memory multiprocessor. In this particular case, 2 of the networks of Figure 1 are used to build a system with 16 processors and 16 memory modules

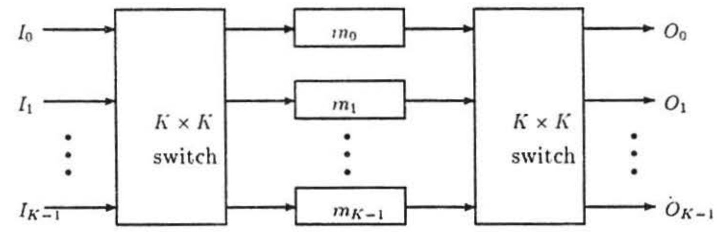

Figure 3: A supermodule contains K independent memory modules. Any of the K inputs can reach any memory module, which in turn can reach any of the K outputs. The way the supermodule is used there is never a conflict in any of the switches.

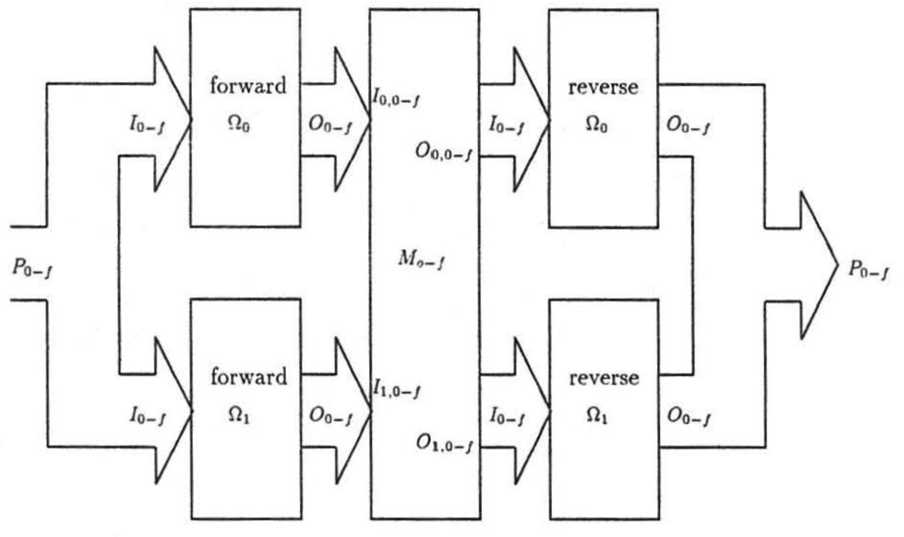

Figure 4: An illustration of the use of multiple Omega networks to both increase the bandwidth between processors and memory modules, and improve the ratio of memory modules to processors. In this case, two of the networks of Figure 2 are used to connect 16 processors to 64 memory modules (16 supermodules). M_{0-f} represents the 16 supermodules. $I_{0,0-f}$ and $O_{0,0-f}$ represent input and output 0 for the 16 supermodules, respectively. $I_{1,0-f}$ and $O_{1,0-f}$ represent input and output 1 for the same supermodules.

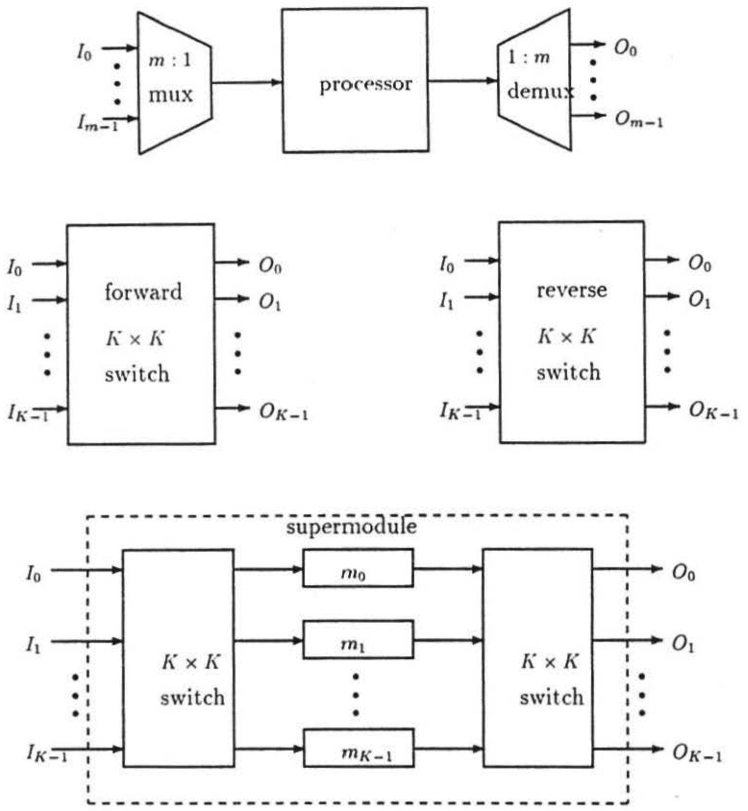

Figure 5: Building block for systems with K, K^2 ..., K^K processors ($K = (8, 16, 32)$). The parameter m varies with the size of the system, therefore the multiplexer and demultiplexer used with the processor must be programmable. Also, there must be K physical memory modules contained in the supermodule. The I's and O's are local inputs and outputs, respectively (they are not interconnected).

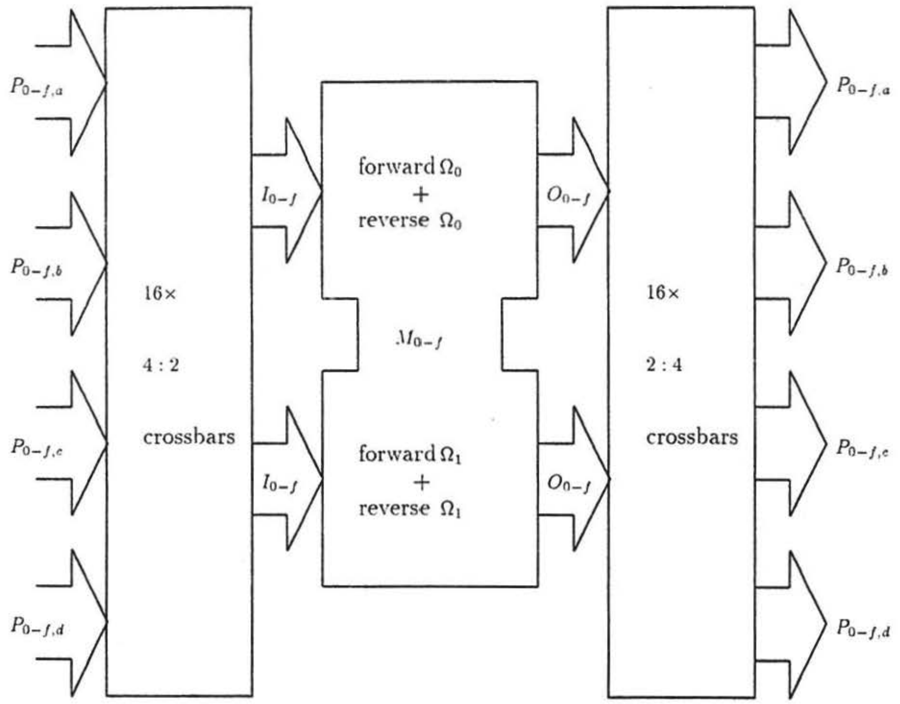

Figure 6: One of the options for processors with multiple ports accommodates the same number of processors as in the one-port case. It uses an $p : m$ crossbar switch for each processor to go from p processor ports to the m ports per processor that the network provide. In this example, $p = 4$ and $m = 2$.

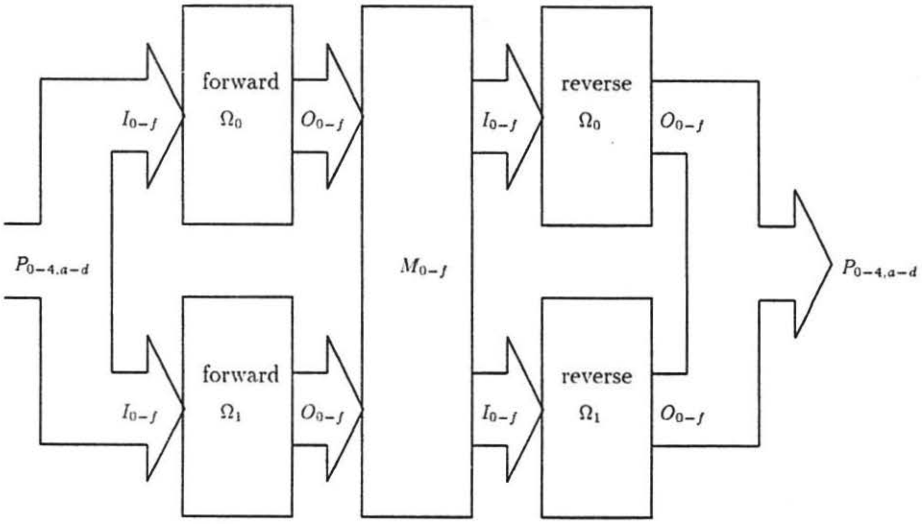

Figure 7: Another option for processors with multiple ports to memory is to treat each *port* as a virtual processor. In this specific example, 4 processors with 4 ports each behave as 16 *virtual* processors, and are connected to 64 memory modules, using the same ensemble as in Figure 4.

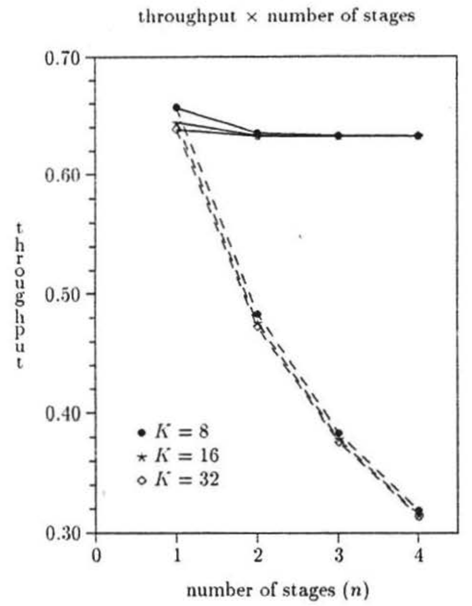

Figure 8: A comparison of throughputs of crossbar switches (solid lines) and Omega networks (dashed lines) built with $K \times K$ switches ($K = 8, 16, 32$). An Omega network (N, K), with $n = \log_K N$ stages is compared to a crossbar switch of size $K^n \times K^n$ (same number of inputs and outputs).

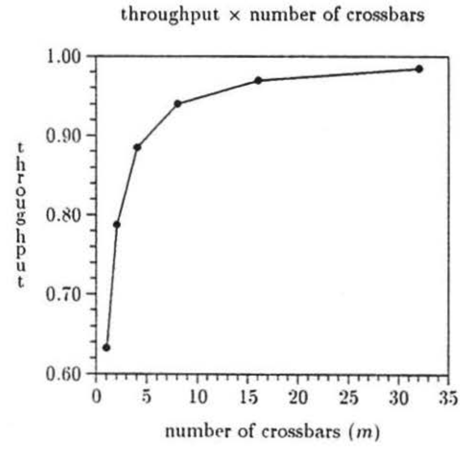

Figure 9: The limiting throughput of m $N \times N$ parallel crossbars (or one $N \times mN$ crossbar) as $N \to \infty$. Notice how the use of multiple interconnections between processors and memories increases the bandwidth of communication.

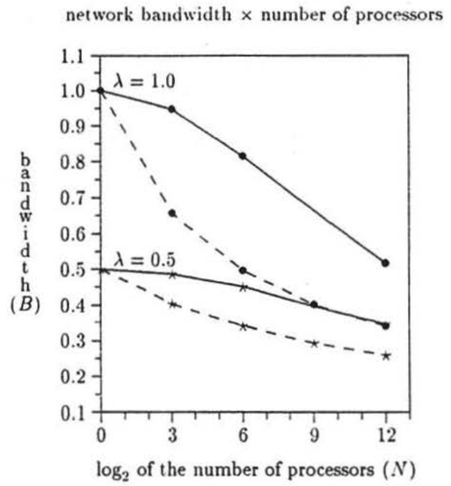

Figure 10: A comparison of normalized network bandwidths for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 8×8 switches, for different input rates (λ). The systems that can be built have 8, 64, 512, and 4096 processors for the case of a single network, 8, 64 and 4096 processors for the case of multiple networks.

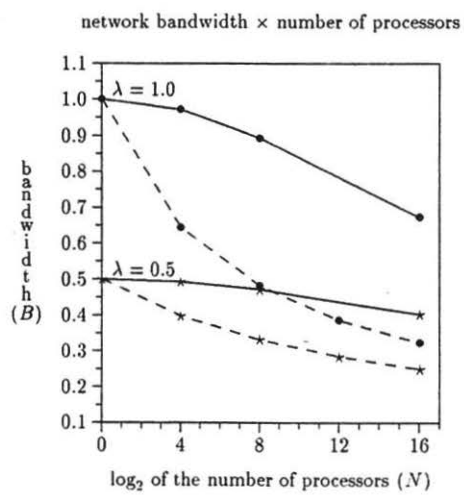

Figure 11: A comparison of normalized network bandwidths for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 16×16 switches, for different input rates (λ). The systems that can be built have 16, 256, 4096, and 65536 processors for the case of a single network, 16, 256, and 65536 processors for the case of multiple networks.

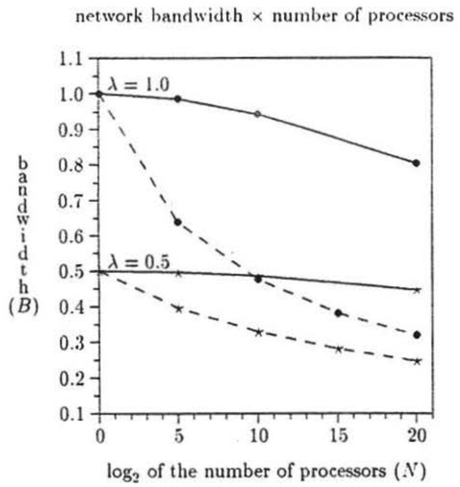

Figure 12: A comparison of normalized network bandwidths for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 32 × 32 switches, for different input rates (λ). The systems that can be built have 32, 1024, 32768 and 1048576 processors for the case of a single network, 32, 1024, and 1048576 processors for the case of multiple networks.

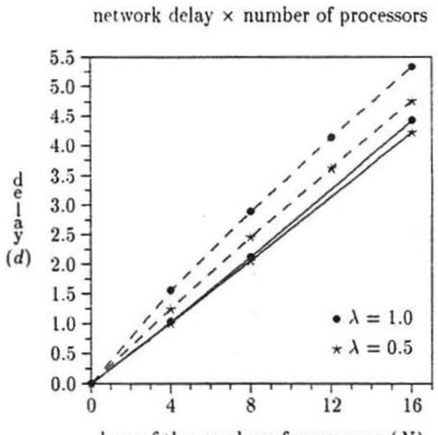

Figure 14: A comparison of input-to-output delays for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 16 × 16 switches, for different input rates (λ). The systems that can be built have 16, 256, 4096, and 65536 processors for the case of a single network, 16, 256, and 65536 processors for the case of multiple networks.

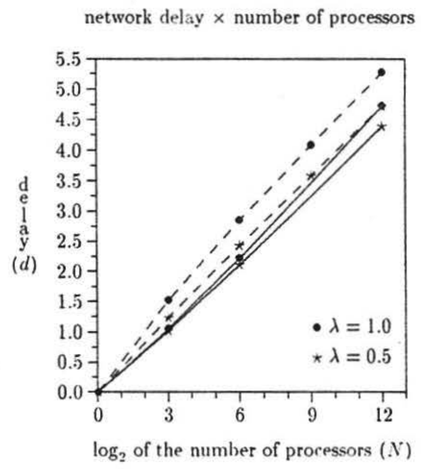

Figure 13: A comparison of input-to-output delays for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 8×8 switches, for different input rates (λ). The systems that can be built have 8, 64, 512, and 4096 processors for the case of a single network, 8, 64 and 4096 processors for the case of multiple networks.

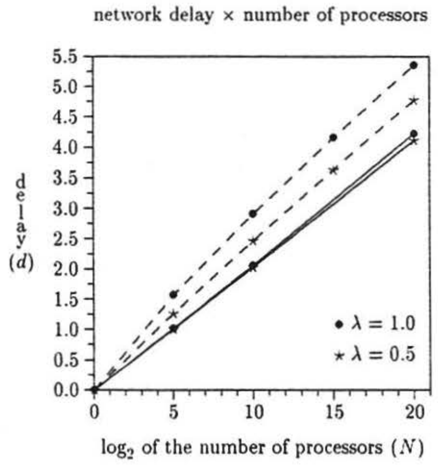

Figure 15: A comparison of input-to-output delays for single Omega networks (dashed lines) and multiple Omega networks (solid lines) built with 32 × 32 switches, for different input rates (λ). The systems that can be built have 32, 1024, 32768 and 1048576 processors for the case of a single network, 32, 1024, and 1048576 processors for the case of multiple networks.

K	$N = P = M$	$n = \log_K N$	$S = \frac{N}{K} \log_K N$	P/S
8	8	1	1	8
8	64	2	16	4
8	4096	4	2048	2
16	16	1	1	16
16	256	2	32	8
16	65536	4	16384	4
32	32	1	1	32
32	1024	2	64	16
32	1048576	4	131072	8

Table 1: Some parameters for shared-memory multiprocessors built around Omega networks. We show those *complete* networks with no more than 1M inputs/outputs that can be built with 8×8, 16×16 and 32×32 switches, and have the property that the ratio of processors to switches is an integer.

processors $P = N$	switch size $K \times K$	memory $M = KN$	number of parallel networks
8	8×8	64	8
64	8×8	512	4
4096	8×8	32768	2
16	16×16	256	16
256	16×16	4096	8
65536	16×16	1048576	4
32	32×32	1024	32
1024	32×32	32768	16
1048576	32×32	33554432	8

Table 2: Some parameters for shared-memory multiprocessors built around multiple Omega networks. All these systems have a processor-switch ratio of 1/2 (one switch to be used in the forward networks, the other in the reverse networks). The number of parallel networks used is $m = K/\log_K N$, and the ratio of memory modules to processors is K.

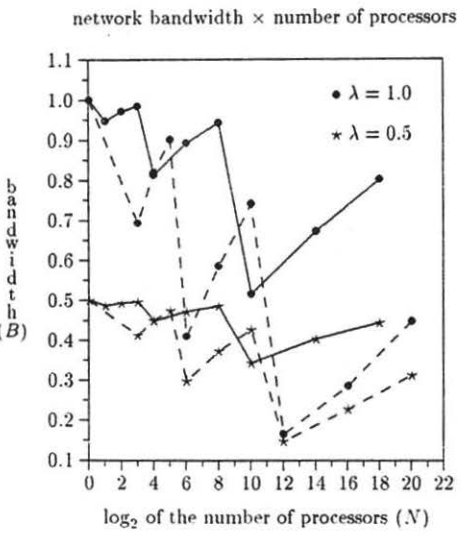

Figure 16: A comparison of normalized bandwidths for networks built for processors with 4 memory ports. The solid lines represent the values for networks that use the *virtual processor* approach (systems with 2, 4, 8, 16, 64, 256, 1024, 16384, 262144 processors), while the dashed lines are the values for networks that use the *crossbar* approach (systems with 8, 16, 32, 64, 256, 1024, 4096, 65536, 1048576 processors). We notice that for systems with 8, 16, 64 and 256 processors the virtual processor approach gives better results, while for systems with 1024 processors the crossbar approach gives better results.

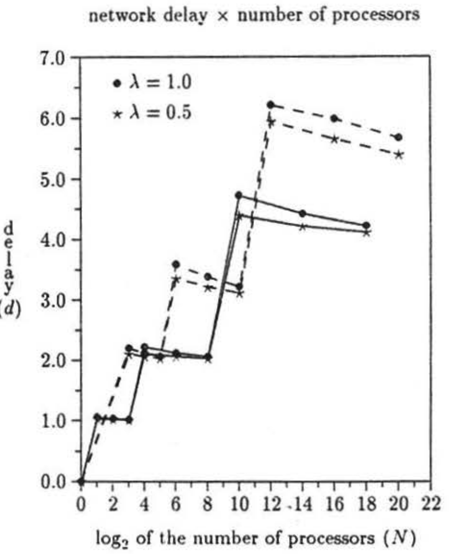

Figure 17: A comparison of input-to-output delays for networks built for processors with 4 memory ports. The solid lines represent the values for networks that use the *virtual processor* approach (systems with 2, 4, 8, 16, 64, 256, 1024, 16384, 262144 processors), while the dashed lines are the values for networks that use the *crossbar* approach (systems with 8, 16, 32, 64, 256, 1024, 4096, 65536, 1048576 processors). We notice that the virtual processor approach gives better results for systems with 8, 64 and 256 processors, while the crossbar approach gives better results for systems with 16 and 1024 processors.