ECE669: Parallel Computer Architecture
Fall 2004 **Handout #4**

Homework # 4 Due: Novemeber 3

Scalability and Interconnection Networks

Ex 1: Scalability

Q 1: Using the definition of scalability presented in class, compute the scalability of 1-D, 2-D, 3-D, and hypercube networks for the relaxation phase of a 3-D Jacobi relaxation algorithm. As discussed in class, assume that a word of data can be transferred between adjacent nodes of the network in one cycle. Assume that there are no bandwidth constraints.

Ex 2: Interconnection Networks

In this exercise we will study interconnection networks. The study of interconnection networks is important because the overall performance of a multiprocessor is often critically hinged on the effectiveness of its network. Appendix C will lead you through the mechanics of running network simulations.

For your first experiment, you will verify the accuracy of the analytical network model discussed in class.

Q 2: Run a simulation with 64 processors interconnected in a 2 dimensional mesh using a request rate of 0.04 (messages per second) and a message size of 4 (flits). Run the simulations with 2000 packets. The simulation yields the effective network channel utilization ρ and the effective message latency T. When certain simulation parameters are not specified in the problem set, use the default values from Appendix C.

You will now compare these values with the corresponding numbers obtained through an analytical network model. Since the simulation you just completed does not incorporate feedback from the network (that is, it does not modify the input request rate depending on the network latency), you should use the network model in Appendix A to compute the network latency. Note that, like the simulator, the model assumes unidirectional channels. (The simulator can handle other situations as well, but you will not need these for your homework.)

The next experiment is designed to give you a feel for the shape of network latency curves.

Q 3: Obtain the network utilization and message latency through simulations for a 2 dimensional 64 processor network for the following request rates and message sizes.

- Request rates vary from 0.01 to 0.06 in increments of 0.01.
- Message sizes vary from 2 to 10 in increments of 2.

You will find the `foreach` feature of Unix `csh` very helpful for running these simulations. Here is an example of its use:

```
Unix Prompt> foreach i (1 2 3)
? echo $i
? end
1
2
3
Unix Prompt> foreach i ( 1 2 )
? foreach j (this-one that-one)
? echo $i $j
? end
? end
1 this-one
1 that-one
2 this-one
2 that-one
```

Note that `foreach` statements may be nested. Use the man page for more information.

Plot latency versus request rate curves to get a better feel for the numbers. Compare with the predictions of the model (assuming no feedback from the network). Which of the above combinations of request rates and message sizes are valid given the maximum allowable channel utilization of 1?

How do you interpret the results from the network simulations when the maximum allowable channel utilization is exceeded in the model? Or, how do you expect the network simulator deals with offered request rates that exceed the maximum achievable limit. (Hint: think of what will happen at the input queues into the network.)

Q 4: If you are allowed to choose between doubling your message size and doubling your request rate, given a current request rate of 0.02 and a message size of 4, which would you prefer? Use the network performance table that you just computed in determining the answer to this question.

Are there any other considerations that might influence the above choice?

Q 5: Consider a 1-dimensional network with end-around connections (that is, a ring). Let the channels be unidirectional. Let there be k nodes in this network. If messages from a given node are sent with equal probability to any other node (including itself), derive the average number of hops traveled by a message.

Direct networks have the advantage of locality. That is, a message traveling between physically adjacent nodes uses less network bandwidth and suffers lower latency than a message that must travel between nodes separated by a larger distance. Let us now investigate how the exploitation of locality translates to improved performance in mesh

networks.

In our simulator the locality parameter adjusts the number of processors that can communicate with each other. For example, a locality parameter of 0.5 simulates a workload in which each processor sends messages only to a subcube consisting of half the total number of processors in the machine. This subcube has a corner at the source node, and destination processors for each message are chosen randomly from among this subset of processors.

For example, let us consider an N-processor torus in which nodes are represented by their x and y coordinates. Given a locality fraction l, destination nodes for messages originating from source node (i, j) are randomly chosen from the set of nodes with coordinates $(x|\ i \leq x \leq i + \sqrt{lN} - 1, \quad y|\ j \leq y \leq j + \sqrt{lN} - 1)$.

Q 6: Measure network latency for the locality parameter varying from 0.2 through 1.0 in increments of 0.2. Use a request rate of 0.001 and message size 4 for your simulations.

Next, measure network latency with the same range in the locality parameter, but with a much higher request rate of 0.04.

You will notice that the effect of communication locality is more pronounced at the higher load. Can you come up with some insight explaining this behavior?

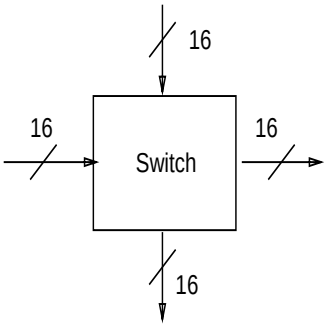
Q 7: How might you modify the analytical model for k-ary n-cubes to account for communication locality? Hint: the distance in a dimension parameter k_d is related to the locality parameter l.

The next experiment involves making a set of tradeoffs that commonly arise when designing networks for real systems.

Q 8: Design a k-ary n-cube interconnection network with the highest performance (lowest latency) given the following constraint: each network node must fit on a chip with a maximum of 64 pins allocated to the wires between nodes. Neglect the wires between the network chip and the processor. You are free to use either the simulator or the model in this analysis. (Ignore issues of wire length.)

Note that the simulator does not simulate past four dimensional networks, so you might consider using the model for this part, or argue that simulating past four is unnecessary. Also note that when the simulator cannot determine an integer dimension value from the specified values of N and k, it increases N so that an integer value for the dimension can be found.

An example chip might look as follows:

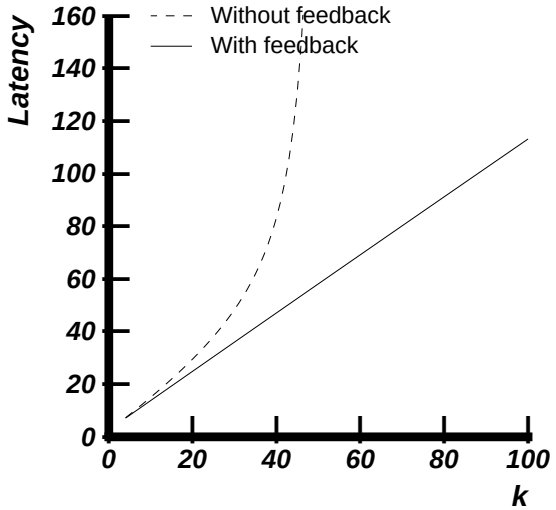


Assume the following workload parameters:

- The number of processors N is 64.
- The request rate m is 0.04.
- The message size is 160 bits. (Note that the message length used by the simulator is specified in flits. For example, for 160-bit messages, $B = 10$ when the channel widths are chosen to be 16 as in the above figure.
- The locality parameter l is 0.5.

Q 9: In this exercise you will study network performance when the processor cannot support unlimited outstanding requests. Appendix B presents a network model in which the processor is allowed a single outstanding request.

The network latency using the model with feedback and the model without feedback are plotted in the following figure.



The curve for the model appears largely linear as k increases when the request rate is controlled by network feedback. Using the model, show that when the network becomes

very large, ($k >> 1$), network latency is linearly related to k_d. How does processor utilization relate to k_d?

A Modeling the Performance of k-ary n-cube Interconnection Networks

The network latency can be derived by plugging the network request and message size into the following network model (more details in class). The model shown here assumes unidirectional channels.

We will use the following notation:

T	network latency
m	message rate
B	message size
ρ	network channel utilization
k	network radix
k_d	distance traveled in a dimension ($\frac{k-1}{2}$)
n	network dimension
M	memory latency (assume $M = 0$)
N	number of processors
U	processor utilization

For a k-ary n-cube the network latency can be derived as shown below. (Assume $M = 0$ in all your computations.)

$$Channel\ Utilization\ \rho = mBk_d$$

$$Network\ latency\ T \approx \left[1 + \frac{\rho B}{(1-\rho)} \frac{(k_d - 1)}{k_d^2} \left(1 + \frac{1}{n}\right)\right] nk_d + M + B$$

The above expression is fairly accurate for values of $k_d > 2$. For smaller values of k_d, the following expression is more accurate.

$$Network\ latency\ T \approx \left[1 + \frac{\rho B}{(1-\rho)} \frac{(k_d - \frac{1}{2})}{(k_d + \frac{1}{2})^2} \left(1 + \frac{1}{n}\right)\right] nk_d + M + B$$

B Computing Processor Utilization

The network latency computation in Appendix A assumed that the processor issued requests at the rate m, irrespective of network latency. This model is valid when the processor can do other computation while waiting for the network to satisfy a request. For such

processor architectures, the processor utilization is usually close to one, that is $U = 1$. The processor utilization can be thought of as the probability the processor is doing useful work in any given cycle. Clearly, $0 \leq U \leq 1$.

Alternatively, a processor can wait for the completion of a request before proceeding. When a processor does not allow computation while a network request is outstanding, we must compute processor utilization and network latency slightly differently.

Essentially, because the processor cannot issue requests while it is waiting for the network, the probability of a request on a given cycle (or request rate) is reduced. Note that m is the probability of a network request only during a cycle that the processor is busy. Consequently, the effective request rate into the network must be reduced by the fraction of processor idle time. The set of equations computing processor utilization can now be corrected as follows to reflect the modified request rate of mU:

$$Channel\ Utilization\ \rho = UmBk_d$$

$$Network\ latency\ T = \left[1 + \frac{\rho B}{(1-\rho)} \frac{(k_d - 1)}{k_d^2} \left(1 + \frac{1}{n}\right)\right] nk_d + M + B$$

$$Processor\ utilization\ U = \frac{1}{1 + mT}$$

However, we now have a problem. There is a cyclic dependency in the set of equations above. Such a set of equations arise in feedback systems. Notice that the network exerts backpressure on the processor causing it to reduce its effective request rate.

The above system of equations with the two unknowns, T and U, can be solved in various ways. For example, standard iterative numerical techniques can be used to solve the above set of equations. These methods essentially assume some initial value for U and compute ρ, then compute T, and then the resulting U. The resulting U is fed back and the computation continued in a similar fashion till there is no appreciable change in the value of U from iteration to iteration. This value of U is called the *fixed point* of the set of equations.

The above set of equations is simple enough that we can solve it directly. We first substitute for U (using $U = \frac{1}{1+mT}$) in the equation for ρ. We then substitute this expression for ρ in the equation for T, which results in one equation with only one unknown T. We can then knead the above equation into the form of a standard quadratic, and solve for its roots, which gives us T. Finally, substitute in $U = \frac{1}{1+mT}$ to obtain the value of U.

The above set of equations yield the following effective network latency:

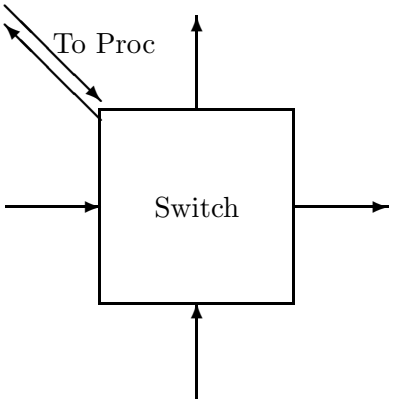
$$T = \frac{T_0}{2} + \frac{Bk_d}{4} - \frac{1}{2m} + \frac{1}{2}\sqrt{\left(T_0 - \frac{Bk_d}{2} + \frac{1}{m}\right)^2 + 2B^2(k_d - 1)(n + 1)}$$

where $T_0 = nk_d + M + B$ is the unloaded network latency. The corresponding utilization is computed as:

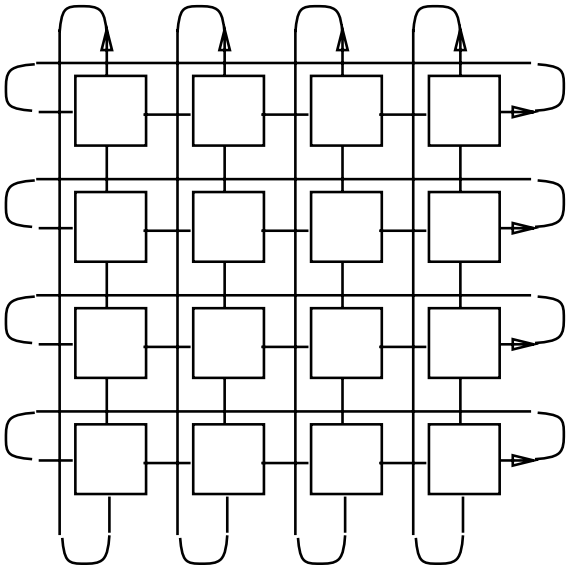
$$Processor\ utilization\ U = \frac{1}{1 + mT}$$

C Using the Network Simulator

The network simulator is called *netwrap*. It can simulate various k-ary n-cube networks. Its current configuration limits the dimension n to a maximum of 4. The simulator accepts the network parameters such as radix k, number of processors N (the number of dimensions n is calculated using $n = log_k N$), and workload parameters such as message rate, message size, and locality. It outputs network statistics such as network channel utilization and average latency. The network simulator assumes infinite buffering at the switching nodes. It assumes unidirectional channels, and the network edges are connected to form a torus. For example, a network switch in a 2-D torus would look like:



The corresponding network might look like:



The simulator is run as follows. The arguments and their typical values are described in the following table. Statistics such as network channel utilization and average latency are sent to standard output.

```
prompt% netwrap procs pkts steps req-rt msg-size rad-k det-file loc > stat-file
```

argument	description	typical value
procs	number of processors	64
pkts	TOTAL number of packets to simulate	1000
steps	number of time steps to simulate	9999999
req-rt	message probability from a node	0.05
msg-size	average message size	6
rad-k	network radix k	8
det-file	file for detailed statistics	/dev/null
loc	locality parameter (fraction of procs communicated with)	1.0
stat-file	statistics file	netwrap.out

The simulator performs a cycle-by-cycle simulation of the network till the specified number of packers are generated. Each cycle every node generates a packet with probability *req-rt* of constant size (equal to *msg-size*). The message destinations are randomly chosen from among all the nodes in the machine. However, when the locality parameter *loc* is less than one, the simulator generates a randomly chosen destination from among a smaller subcube of nodes centered at the source node. The number of processors in this subcube is smaller than the total number of processors by the fraction *loc*.

To fire up a simulation with the typical arguments shown in the above table (corresponding to a 2 dimensional torus with 64 processors) type:

```
prompt% netwrap 64 1000 9999999 0.05 6 8 /dev/null 1.0 > netwrap.out
```