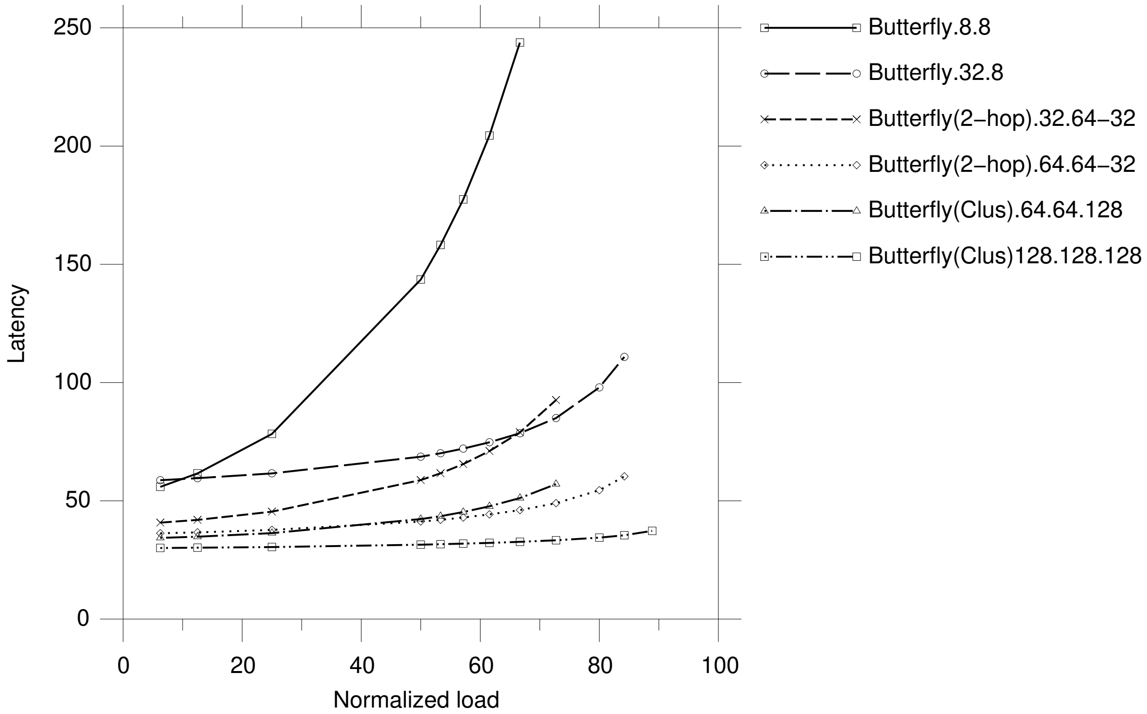

Figure 6.5: Open-Network Model; M=32 modules, C=4 modules per cluster

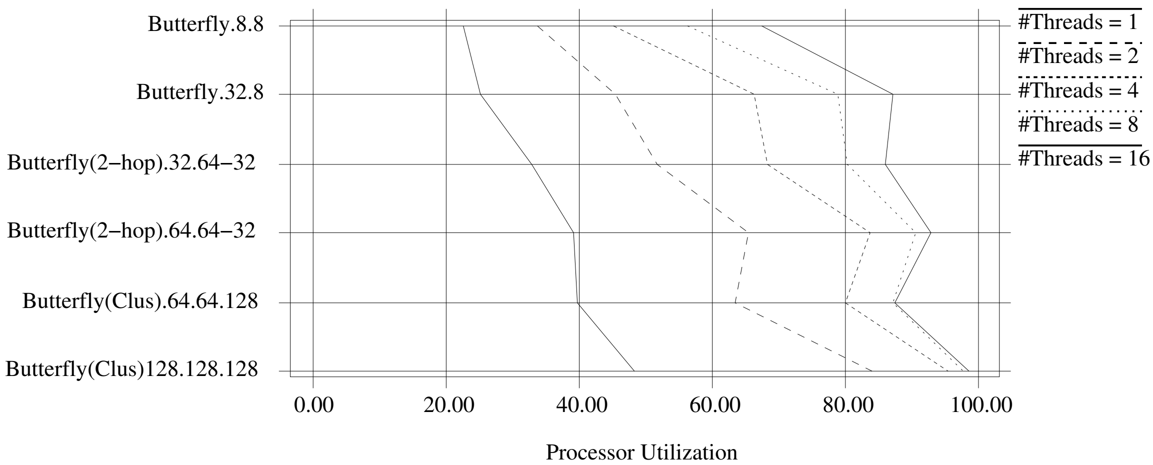

Figure 6.6: Periodic Access Model, access interval = 16 cycles

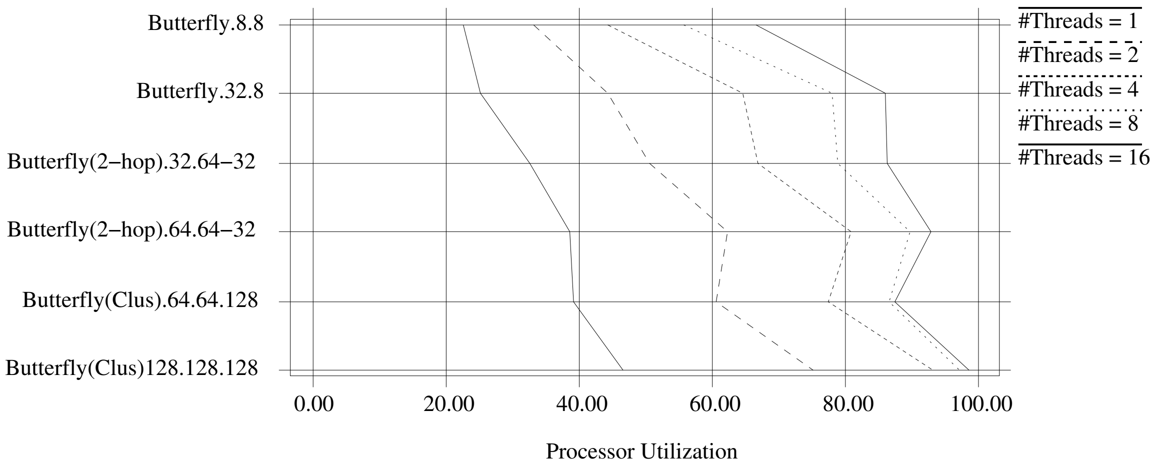

Figure 6.7: Geometric Access Model, access interval = 16 cycles

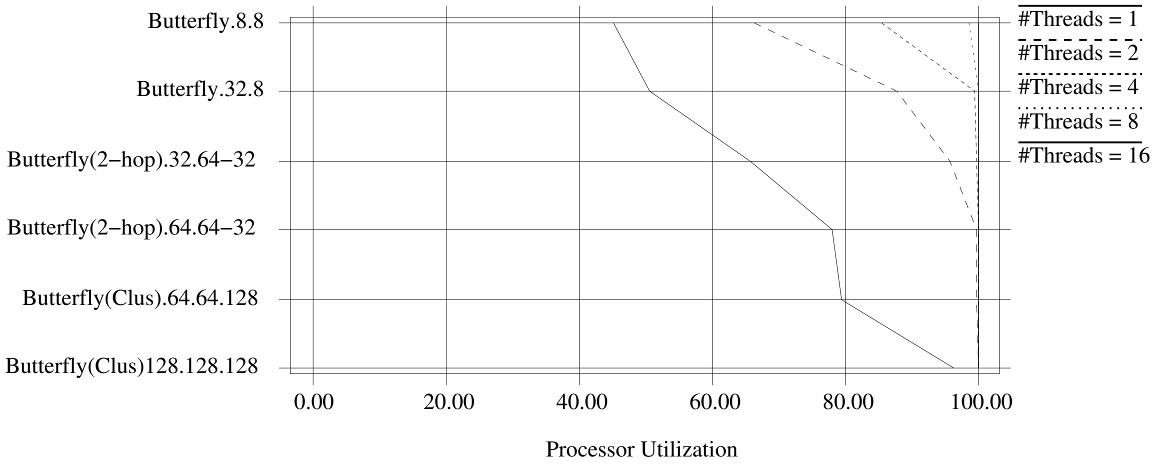

Figure 6.8: Periodic Access Model, access interval = 32 cycles

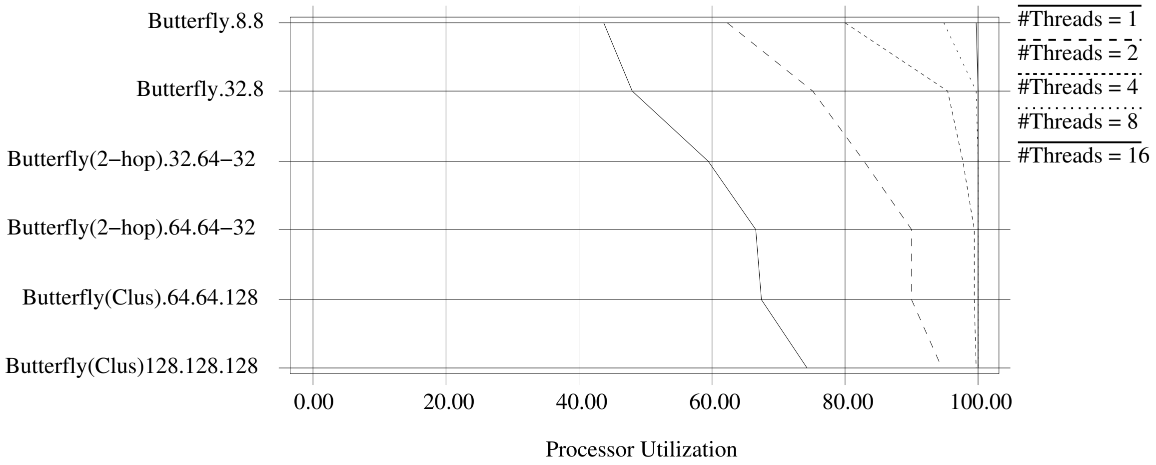

Figure 6.9: Geometric Access Model, access interval = 32 cycles

Another interesting factor seen from the open-network results, is that *Butterfly(2-hop).64.64.32* and *Butterfly(Clus).128.128.128* have almost flat latency curves even up to the point where the load is as high as 75 to 80% of the peak. This indicates that the high bandwidth available at the lower levels of the packaging hierarchy eliminates almost all the network contention for the channels at the lower levels of the hierarchy.

6.3 Summary

In this chapter, we considered a packaging model that was more complex than the ones considered in the two previous chapters. The model was based on a three level hierarchy and was motivated by the need to model short and long-distance connections. Under this model, we compared the cost of several networks with equal peak performance. Since the cost vector under this model was more complex than the two previous models, we only examined the trade-off between two of the dimensions of the cost vector, namely the short-distance pin-count and the number of long-distance bundles. We saw that we could reduce the number of long-distance bundles by using a larger number of short-distance wires. We illustrated this using three different interconnection networks that represented three separate positions in this trade-off. When we looked at the performance on a finer scale, we saw that as we reduce the number of long-distance bundles, we got better performance because reducing the number of long-distance connections increased the widths of the long-distance channels, which, in turn, improved their utilization.

Chapter 7

Scalability

Computer manufacturers usually want to build machines over a wide range in price and performance to cater to the needs of different customers with varying budgets and computing requirements. Such a design goal was proposed as early as the 1960s for the IBM System/360 [SBN82]. The IBM System/360 achieved a performance range of fifty to one by employing a wide choice of components, engineering techniques and technologies. The designers of the 360 family invested considerable effort in the design of separate machines for each point on the price-performance curve.

A similar design goal of providing a wide range in price and performance becomes even more important in the case of parallel machines, since it seems natural to buy more processors when a more powerful machine is needed. Further, in the case of parallel machines achieving this design goal seems easier (than in the case of System/360), since performance can be increased by increasing the number of processors. Therefore, parallel machine manufacturers wish to design machines that can be built for a wide range of sizes to enable customers to purchase a machine appropriate to their needs. Besides, customers also frequently discover that their computing needs outgrow the capacity of the machine they currently have [Bel92], and often desire to upgrade their machine by increasing the number of processors. Therefore, manufacturers often want the ability to incrementally upgrade parallel machines to larger sizes.

Since designing parallel machines usually requires substantial investment in time, effort, and money, it is desirable to reduce the complexity associated with the design process. While designing machines over a range of sizes, manufacturers prefer that many aspects of the design carry over from one machine size to another. It is often preferable to build the entire range of machines from some small set of building blocks. The effort invested in designing the building blocks, then is amortized over the whole family of machines. In contrast, designing each machine of the family separately, in a manner similar to the System/360 would imply substantially higher design costs and time.

However, it is conceivable that when each member of the family is designed individually, each member may provide better performance because we may be able to perform many design optimizations that would not be possible in the building block approach. Since the building blocks have to be usable all the machines of the family, we may be faced with more stringent constraints in the design of the building blocks themselves. It is also conceivable that the additional constraints on the building blocks may lead to an increased usage of packaging resources such as pins, wires etc. In other words, cutting design costs may result in an increase in hardware (or packaging) costs or a reduction in performance. In this chapter we explore this trade-off in greater detail.

The ability to build parallel machines over a wide range of sizes is usually referred to as *scalability*. However *scalability* is currently an intuitive notion that has no formal definition [Hil90]. First, in Section 7.1 we formalize the general intuition regarding *scalability* and provide a definition of the term. Our definition permits us to distinguish machine families as being scalable or unscalable. We then classify scalable machine families into a number of broad categories based on the relative design costs associated with them(Section 7.2). This classification is based on the size of the building blocks that are common across the members of the family. We illustrate this classification by extending a few of our earlier 1024 processor designs to families corresponding to each of the scalability categories (Section 7.3). A comparison across the families illustrates the trade-off mentioned earlier(Section 7.4).

7.1 What is scalability?

The main reason for building a family of machines over a range of sizes is to provide a range in cost and performance. Larger machines should provide correspondingly higher performance for higher cost. Therefore we define scalability as follows:

Definition 7.1 *A network family is (P, C)-scalable in the range $[N_0, N_1]$, if for each machine of size $N_0 \leq N \leq N_1$, the performance of the machine exceeds $P(N)$ and the hardware* *cost is less than $C(N)$.*

Note that the definition is qualified by the functions $P(N)$, $C(N)$, and the constants N_0 and N_1. The functions $P(N)$ and $C(N)$ are the desired performance and cost respectively. Ideally, network performance should increase linearly in network size[1]. Therefore, we define $P(N) = constant{*}N$. We would also like the cost to be linear in the size of the network, however a linear increase in performance usually implies a super-linear increase in cost.

The reason for the super-linear increase in cost is as follows. As the network size increases, the average distance traveled by a message usually increases. In order to maintain a linear increase in peak throughput, as the network size increases, the total number of wires in the network should increase in proportion to the product of the network size and the average distance traveled by messages. This implies that, in order to provide a linear increase in peak throughput, the number of wires in the network must increase in a super-linear fashion. Therefore, unless $C(N)$ is a super-linear function, we will end up classifying all networks as unscalable. Typically we might choose $C(N) = constant{*}N \log N$.

Given $P(N)$ and $C(N)$, each network family will have a (possibly null) range of scalability — $[N_0, N_1]$. Figure 7.1 shows the ranges for a few common network topologies[2]. The networks have different ranges of scalability depending on where their performance and cost curves cross the bounding functions $P(N)$ and $C(N)$. Rings have a constant bisection bandwidth and a cost that is linear in network size. 2D meshes also have linear cost, but their bisection bandwidth is proportional to $\sqrt{N}$. For both these networks, the range of scalability is limited by performance. On the other hand, both the bisection bandwidth and cost of the complete graphs are proportional to N^2, and the range of scalability is limited by cost.

If a network family has a range of scalability $[N_0, \infty]$, we call it *asymptotically scalable*. Most practical networks are not asymptotically scalable; instead networks have a bounded range of scalability. Although asymptotic scalability is interesting from a theoretical point of view, practical considerations limit the range of machine sizes that are interesting. It is unnecessary to consider building machines that cost so much, or consume

[1] In this dissertation, we measure performance in terms of the bandwidth of the network under randomly distributed communication traffic and we estimate the network performance using its peak throughput. We saw that for such traffic the bisection bandwidth is a good estimator of network performance

[2] assuming a fixed channel width

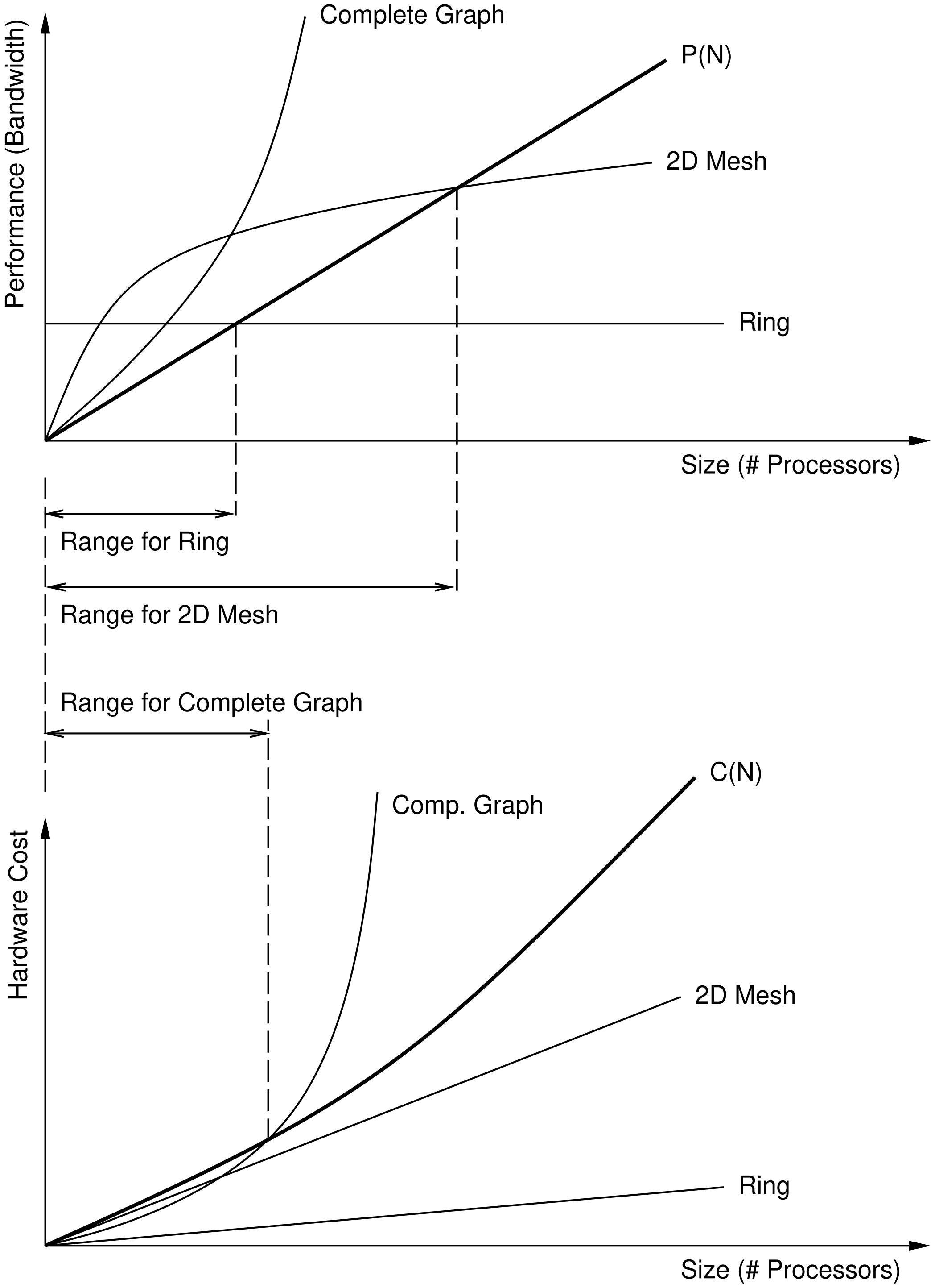

Figure 7.1: Ranges of Scalability

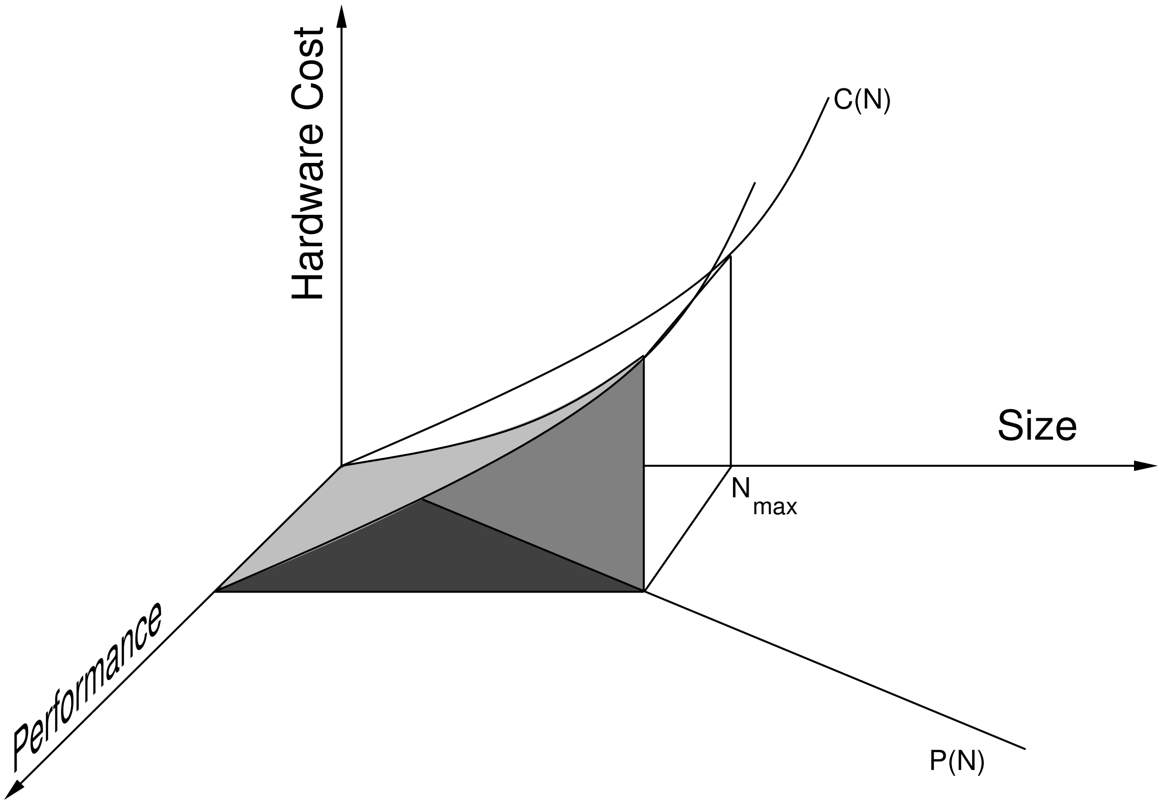

Figure 7.2: Scalability space

so much power, or occupy so much space, that nobody would be able to afford them. For parallel computers built using state-of-the-art microprocessors, we believe that the upper limit is in the neighborhood of 1000–2000 processors[3]. Accordingly, we are mainly interested in networks with a range of scalability $[N_0, N_{max}]$, where N_{max} is defined based on technological and economic considerations. N_0 can be taken to be 0 without loss of generality.

To summarize, we define a network family to be *scalable* if all the members of the family fall within the shaded region shown in figure 7.2.

[3]The largest available configurations of many of the current parallel machines that use reasonably powerful 32-bit processors, such as the CM5, KSR, Intel Delta, etc, are of the order of a 1000 processors [CKP+92]. The reason for this limit is primarily cost; the average cost these machines is in the range of \$10,000 to \$30,000 per processor [BBDS92], and not many customers are willing to spend more than \$30-50 Million on a parallel machine.

7.2 Design costs vs. Hardware costs

Our definition of scalability does not specifically exclude families in which each member of the family has a completely different architecture design. For example, it is possible to define a family of machines which use busses for up to, say, 32 processors, 2D mesh from 32 to 128 processors, and maybe a butterfly for sizes larger than 128 processors. This artificial family illustrates a deficiency in our definition, namely that it ignores design costs.

While it is relatively easy to compare the performance of two network families, and it is possible to use models developed in this thesis to characterize hardware (or packaging) costs, modeling design costs is an extremely complex task. While it is possible to associate physical resources such as pins and area with hardware costs, design costs tend to be intangible and hard to quantify. Instead of trying to develop a model to characterize design costs we will characterize design costs in terms of the fraction of the machine that remains common across the members of the family. An artificial family such as the one described above will have a high design cost since there is very little in common between all the members of the family.

Recall that we model packaging technology as consisting of a hierarchy of l levels of packaging modules, where each level-i module consists of some number of level-$(i-1)$ modules which in turn consist of level-$(i-2)$ modules and so on. The packaging hierarchy gives us a natural way of classifying network families according to design costs. We consider a network family which uses the same level-i modules in all members of the family to have lower design costs than one in which only level-$(i-1)$ modules are common to all members, but the level-i modules are different. In the latter family the building blocks are smaller, and the design of all the levels above level-$(i-1)$ have to be done separately for each member.

When all the members of a network family use the same level-i modules, we say that the family is *scalable by level-i reconfiguration* since changing the machine size requires reconfiguring the interconnections between level-i modules. If a customer wants to upgrade a machine, the machine is dismantled down to its level-i modules, additional level-i modules (which include the additional processors) are brought in and all the level-i modules are then interconnected to obtain the new machine.

Scalability by level-0 reconfiguration means that all machines in a family use the same level-0 modules, which we earlier defined to be processors, memory units or routing nodes. We can consider networks in which even the level-0 modules change from one member to another. Such families have nothing in common across their members, except perhaps, some high level design concept. We call such families *scalable by concept*. Each member of the family uses distinct building blocks that may all be designed based on the same concept. Upgrading such machines to larger sizes essentially means the customer replaces a smaller machine by a larger one. This form of upgrade is similar to upgrading work-stations or other uniprocessor machines.

At the other extreme are families in which the larger networks actually contain the smaller networks as a sub-portion. We say that such families are *scalable by inclusion*. Note that scalability by inclusion is not the same as scalability by level-$(l-1)$ reconfiguration though the top level of the hierarchy is level-$(l-1)$. In scalability by level-$(l-1)$ reconfiguration, the connections between the top level modules can be changed from one family to another, but this is not permitted by scalability by inclusion. Upgrading machines that are scalable by inclusion is extremely simple. The customer purchases additional processors and network hardware, and simply connects them to the existing machine.

Clearly a machine that is scalable by inclusion is also trivially scalable by level-$(l-1)$ reconfiguration, which in turn is trivially scalable by level-$(l-2)$ reconfiguration and so on. Therefore these categories classify interconnection networks into mutually inclusive categories as shown in figure 7.3. As we go towards the center of the figure, design costs decrease owing to the additional constraints on network design. However, as we pointed out earlier, these constraints may increase packaging costs or reduce performance. We shall examine these trade-offs in the following sections.

7.3 Scaling of previous network designs

In Chapters 4, 5 and 6, we considered network designs for 1024 processor machines under different models of packaging costs. The designs that we evaluated were specific to a machine size of 1024. We now consider extending the designs to a family of networks of different sizes and analyze the variation in packaging costs as the network designs conform to the various categories of scalability described above. Here, we consider networks under packaging model 1, but a similar analysis can be performed for the other packaging models. We consider a range of scalability of 1–1024 processors, and a peak communication bandwidth of 8 bits per processor, per cycle, under randomly distributed communication.

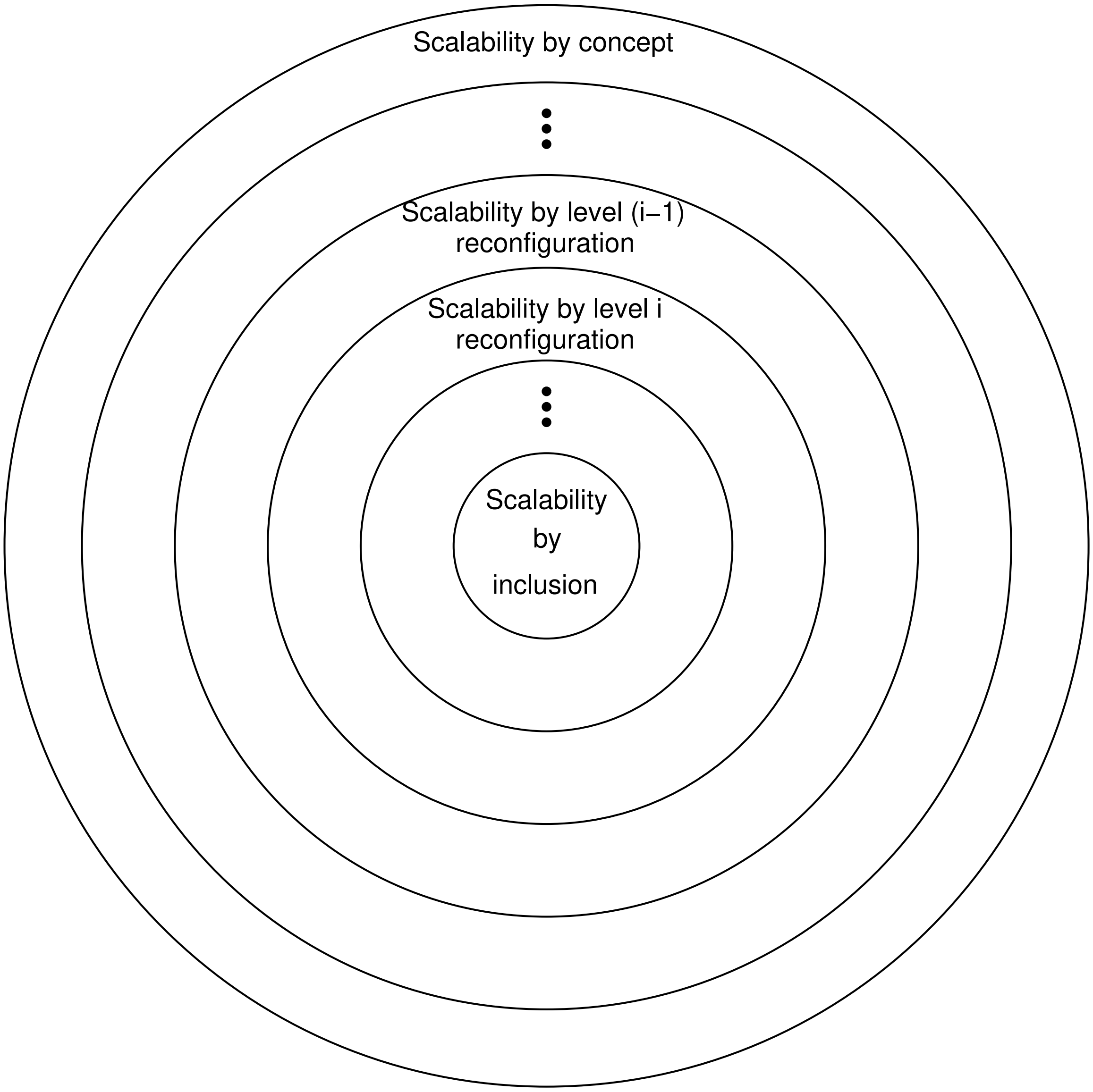

Figure 7.3: Classifying networks in terms of scalability

The packaging technology used under model 1 consisted of a two level hierarchy where processors were placed within *modules*, and were interconnected using two levels of wiring, namely inter-module connections and intra-module connections. The cost function under this model had two components, the number of modules and the pin-count per module. We also had a restriction on the maximum number of processors we could place within a module. In Chapter 4, we considered modules which could hold 16, 32 or 64 processors, but in this chapter we restrict ourselves to 32 processors per module. In this section we show how we can build families of networks for the following network sizes: N=32, 64, 128, 256, 512 and 1024. We shall illustrate three separate families, corresponding to the three categories described above.

7.3.1 Scaling by concept

Under scaling by concept, we permit each member of a network family to be designed independent of the others. This approach permits us to design each member of the family so that it achieves the maximum performance for a given cost. Recall that under model 1, we showed that maximizing the performance for a fixed cost required the inter-module network to be a complete graph. We also saw that we could partition a butterfly network on N inputs into $\sqrt{N}$ modules, to get a complete graph between modules.

We could take the same approach to designing each member of the family of networks, but in this approach, since each module has $\sqrt{N}$ processors, networks smaller than 1024 processors use fewer processors per module than permitted. This results in a larger number of modules than necessary and increases the cost. So we fix the number of modules $M = N/32$ and build a complete graph over these modules.

A natural way of building a complete graph on M modules, is to partition a butterfly network on M^2 inputs. However, the resulting network will only have M inputs per module, but N/M processors per module. Therefore, we connect N/M^2 processors to each input using a binary tree. Memory units are connected in a similar manner. Since the per processor network bandwidth remains constant, all the networks will have approximately the same number of pins per module. However, these pins will be partitioned among fewer channels in the smaller machines, resulting in wider inter-module channels in the smaller machines. Figure 7.4 shows the internals of a module for a network with $N = 256$.

In this family of networks, the channel widths and the amount of routing hardware

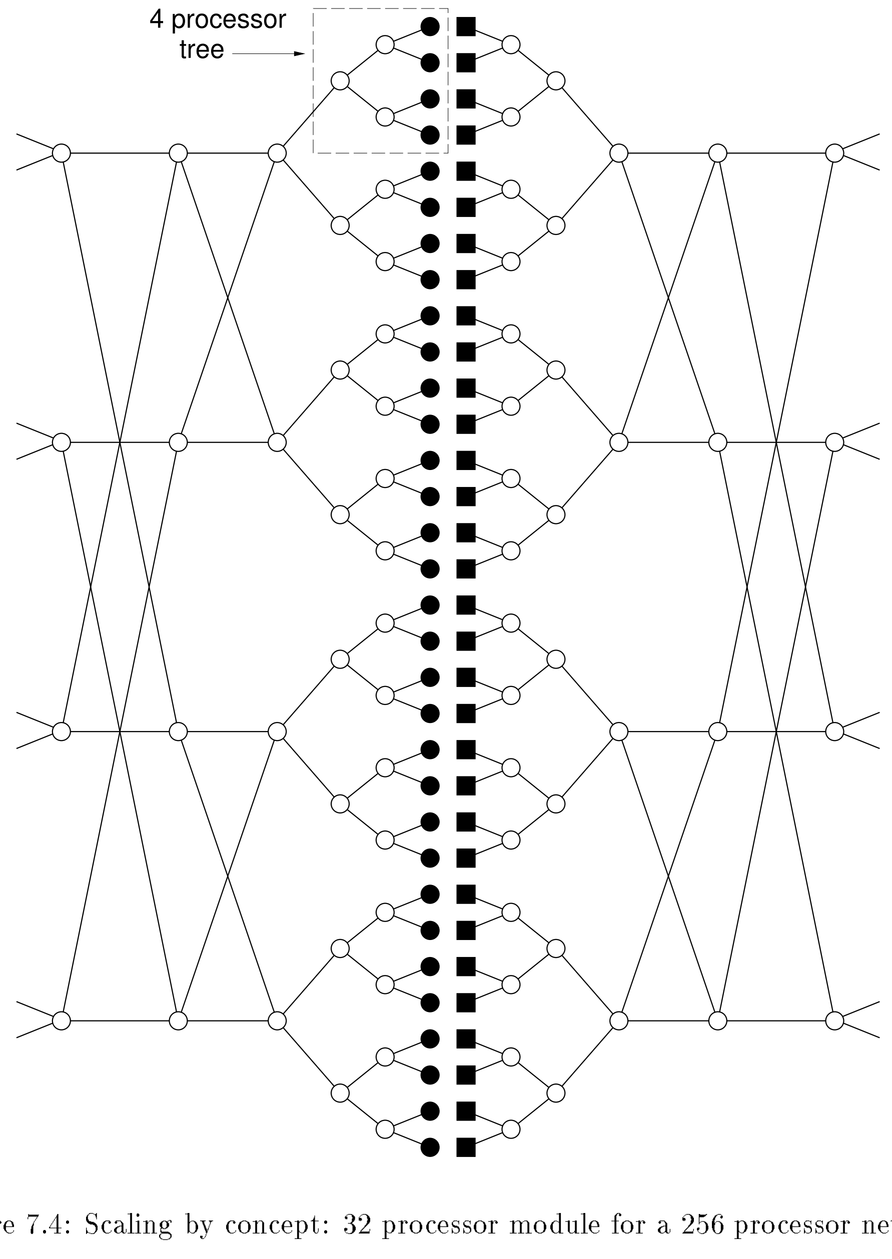

Figure 7.4: Scaling by concept: 32 processor module for a 256 processor network

per module change as we scale the machine. Hence this mode of scaling is considered to be *scaling by concept*. We can also extend this construction to machine sizes larger than 1024 until the point where we have sufficient pins per module to build a complete graph between modules. Machine sizes beyond that are not feasible in this family.

7.3.2 Scaling by level-1 reconfiguration

For scaling by level-1 reconfiguration, we fix the network within a level-1 module, and use the same module to build machines of different sizes. Fixing the contents of a module also fixes the total number of wires leaving the module, and the way the wires are partitioned into channels. We can divide the total number of wires leaving a module into 32 channels corresponding to a machine size of 1024 processors. Machine sizes smaller than 1024 processors, require wider channels between the modules. Since we cannot change the widths of the channels, as we did in the previous family, we use channel dilation to achieve the required number of wires between any pair of modules. If there are N processors, we need $M = N/32$ modules. The connection between any pair of modules consists of $32/M$ channels. This gives us the required peak bandwidth needed to meet the performance specification. In this network, when we double the number of processors, we have to re-organize connections between modules by reducing the dilation of the inter-module channels by a factor of 2 to accommodate the new modules that are to be included.

Each network in this family requires approximately the same number of pins as the previous family. Since packaging cost was assumed to be equal to the pin-count under model 1, this network family has approximately the same packaging cost as the previous one. However, this family requires more routing hardware per module. The additional hardware contributes to higher packaging costs, but this difference is ignored by our packaging model.

7.3.3 Scaling by inclusion

In the two previous families, we used a butterfly network to interconnect the processors. This network consisted of $\log N$ switching stages and connected the processors at one end to the memories at the other end. Since processors and memories are logically paired with each other, we can think of this network as folding a butterfly over itself so that each input is paired with an output. If we have to double the number of processors, we have to double the number of rows of the butterfly and increase the number of columns by 1 and then fold the network over itself. Since the additional column of switching nodes falls between the processors and memories, this network does not lend itself to scaling by inclusion.

One way of achieving scaling by inclusion, is to double the number of routing stages to $2\log N$ to generate a Benes network which is shown in figure 7.5. The network is shown with processors on the left and memories on the right, but it is actually folded over itself so that the processors and memories are placed adjacent to each other. A Benes network is equivalent to a fat-tree network in which the total bandwidth at each level remains constant. Scaling the network in this design requires adding additional modules consisting of only routing nodes. An example of scaling the network from 16 to 32 processors is shown in figure 7.6.

Two of the currently available commercial machines, the Meiko CS2 and the CM5[4] use the folded Benes network. Another network seen in commercial machines that exhibits the property of scaling by inclusion is the Hypercube. In both networks, the average number of hops made a message grows logarithmically in N, the machine size, and in both networks the total number of wires grows as $N \log N$. We already noted that we need a super-linear increase in the number of wires to achieve a linear increase in throughput. The folded benes achieves this super-linear increase in wires by adding routing nodes, where as the hypercube increases the number of wires per node. For scaling by inclusion, we can not actually increase the number of wires per node, so each node in the hypercube should already have enough wires for the largest machine we plan to build. The hypercube, then, wastes these wires in the smaller configurations since they remain unused.

In the family of folded benes networks, the average number of hops that messages have to make, to get from their sources to destinations, is twice the number of hops in the butterfly network. To meet the same peak bandwidth specification, networks in this family require approximately twice the number of routing nodes and twice the number of wires as the butterfly network. We also have restrictions in partitioning this network because we do not want to modify the existing network when we scale the network to a larger size. We can put 32 processors per module as before, and include five columns switching stages within each module for each of the two butterfly networks. For larger machines we need more routing stages and these have to be placed in additional modules which are connected

[4]In the case of the CM5 the bandwidth does not remain constant, but reduces by factor of 4 in the first two levels

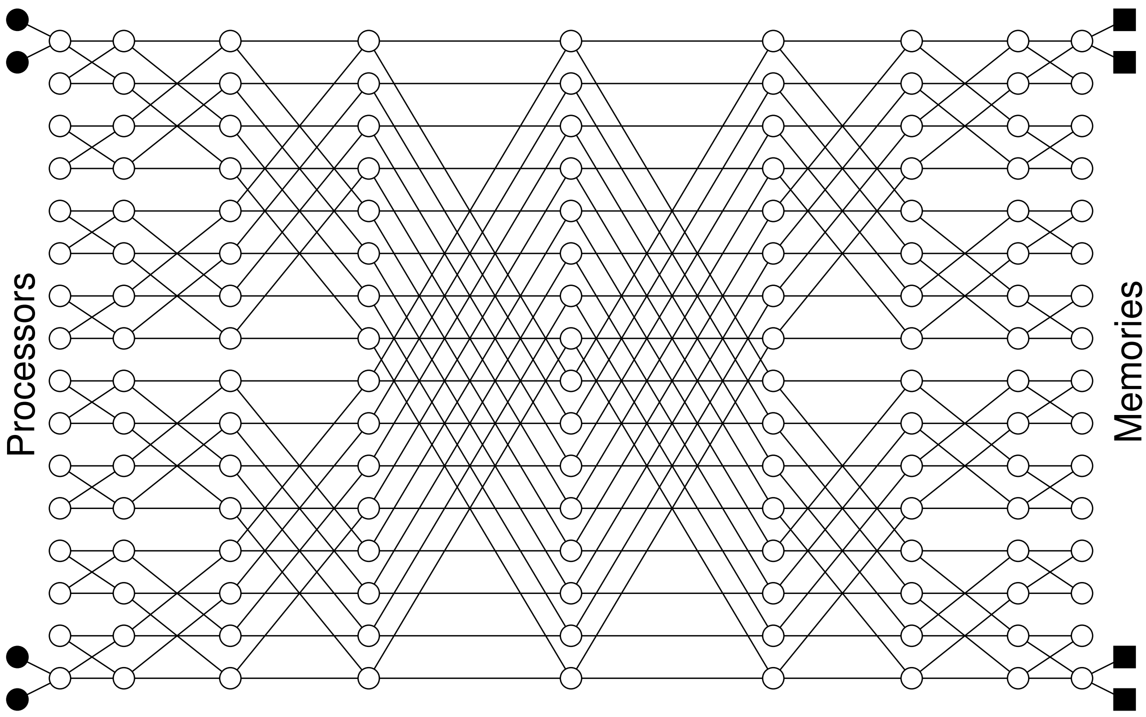

Figure 7.5: Benes network

to the modules that contain processors.

Since the partitioning of the network into modules is different from that of the two previous families, it is difficult to compare the packaging costs of this family with that of the two previous families. However, since we can see that this family uses approximately twice as many wires and routing nodes as the two previous families, we can expect the packaging costs of this family to be around twice that of the previous families.

7.4 Performance Comparison

In this section, we compare the performance of the three families of networks described above. For fairness, we hold the packaging costs of the three families the same while comparing the performance. The differences in performance illustrate the trade-offs between network performance and design cost.

The family of networks that are scalable by concept and the family of networks that are scalable by level-1 reconfiguration have the same packaging cost as per our packaging model. Therefore a direct performance comparison between the two families is justified.

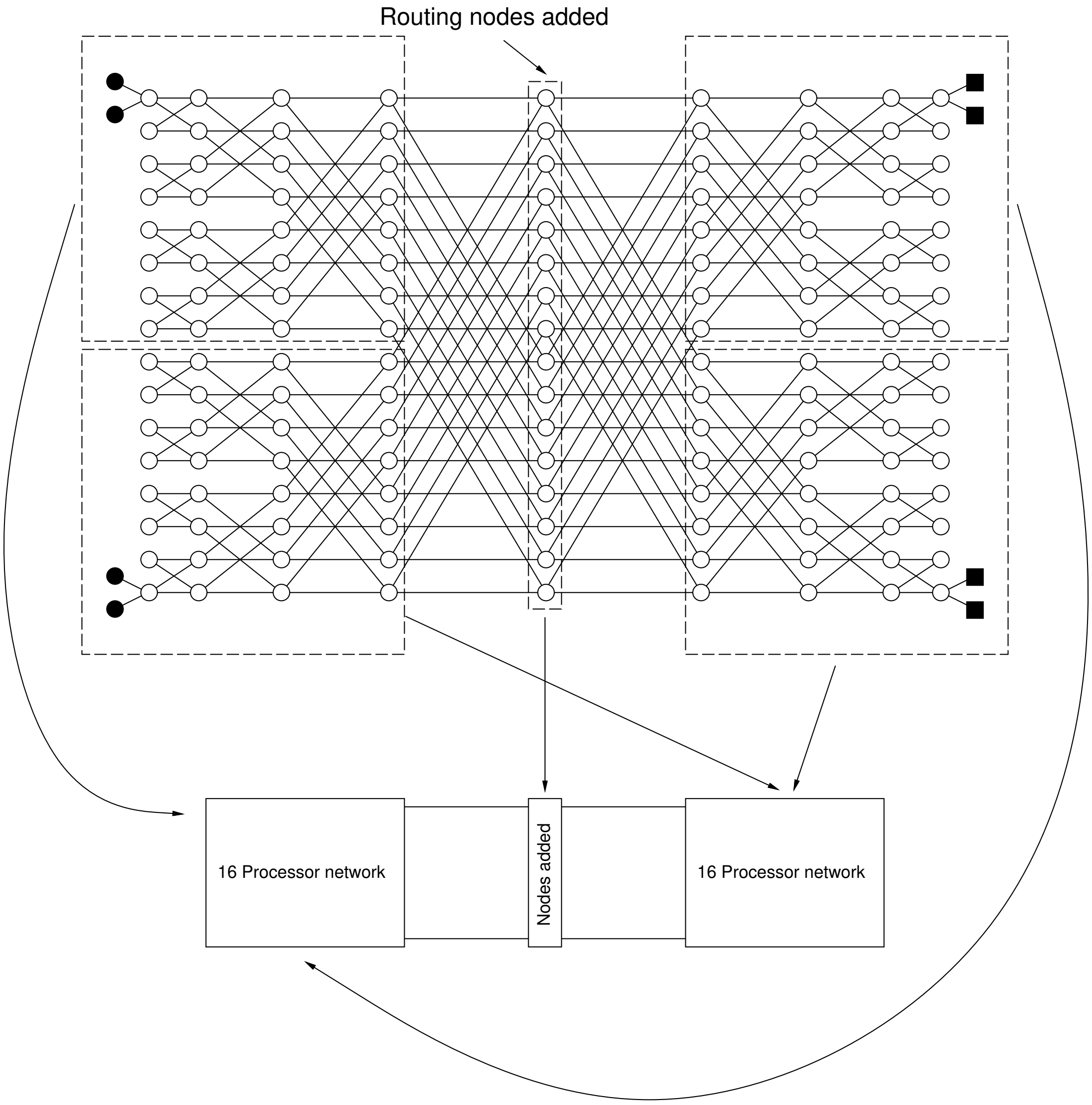

Figure 7.6: Scaling a benes network

However, the packaging cost of the third family (fat-trees or benes networks) is approximately twice that of the other two families. To normalize packaging costs to the same level as the other two families, we have to reduce the bisection width of this network by a factor of two. This will lead to a drop in the performance by approximately a factor of two, making the latter family substantially worse in comparison to the other two families. This suggests that we lose approximately a factor of two in performance for the lower design cost and higher scaling flexibility provided by scaling by inclusion.

Figures 7.7 and 7.8 compares the performance of the networks that are scalable by concept to the networks that are scalable by level-1 reconfiguration. All the networks have the same peak performance (on a per processor basis), but they differ in the fraction of peak performance that is achieved under simulation. When the number of threads is small, we see that the networks that are scalable by concept provide better performance. The main reason for this difference in performance is the difference in the organization of the inter-module wires. In the first family, the wires that go between modules are organized as wide channels, while in the latter the same number of wires is organized as many narrow channels. At low levels of multithreading, the number of messages in the network is insufficient to utilize all the wires in the latter family. As the number of threads increases, the difference in performance reduces, since increasing the number threads results in a larger number of independent messages in the network. In other words, the higher design cost of scaling by concept does not buy much in terms of performance when the number of threads is high, but does so when the number of threads is low.

7.5 Summary

Scalability of parallel machines has long been an intuitive notion without a formal definition. In this chapter we first provided a formal definition of the term which gave us a means for classifying network families as being scalable or unscalable. Our definition of scalability was based on the manner in which the performance and cost increased as we increased the size of the machine. We argued for a linear increase in performance and that a linear increase in performance usually implied a super-linear increase in hardware cost.

We considered two aspects of the cost of a machine, namely hardware (or packaging) costs and design costs. Although design costs were difficult to quantify, we were able to classify scalable networks into three broad categories based on the relative design

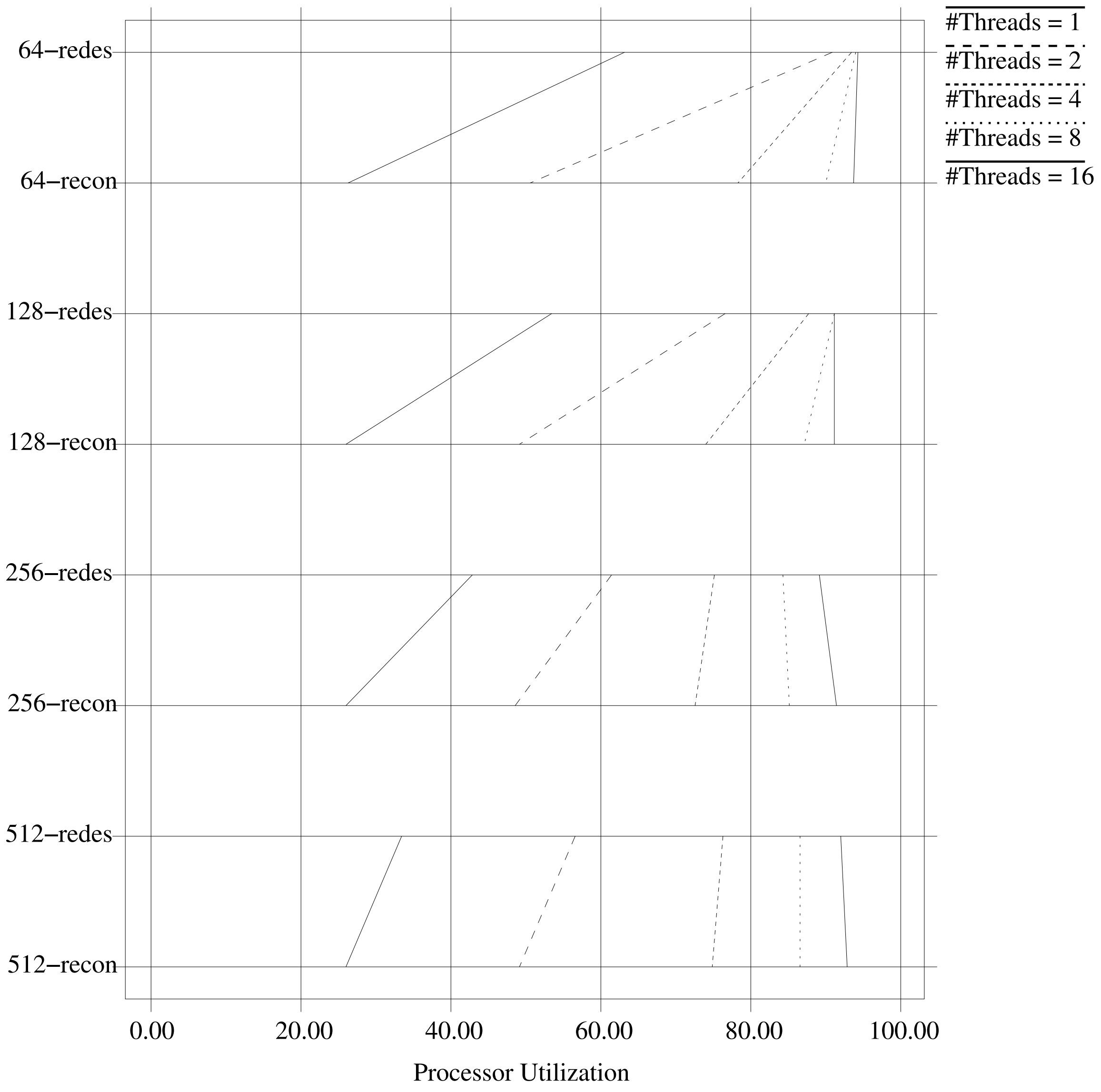

Figure 7.7: Periodic Access Model, access interval = 16 cycles

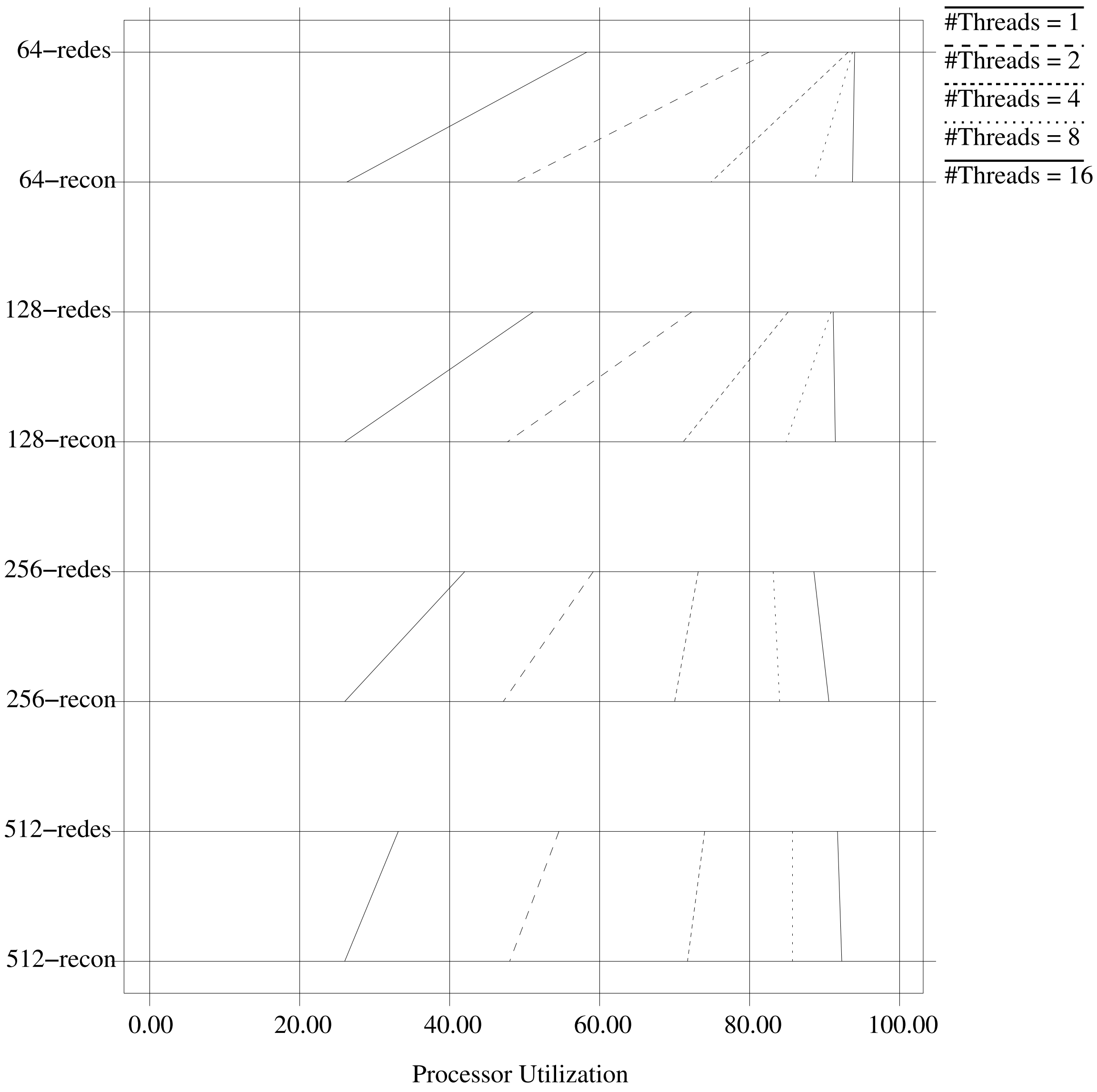

Figure 7.8: Geometric Access Model, mean access interval = 16 cycles

costs associated with them. At one extreme there were families of networks which had high design costs, but low hardware costs for a given performance, and at the other extreme were networks that had low design costs but high hardware costs. We demonstrated extensions to some of our previous designs according to each of the three categories. We compared these three families on the basis of equal hardware costs to illustrate the trade-off between performance and design costs.

Chapter 8

Fault-Tolerance

Most of the networks that we evaluated in this thesis, and found to provide the highest performance for a fixed cost, also had the property that there was a single path from any input port to any output port. While the single path property simplifies the design of the routing nodes, it also poses difficulties in the presence of faults. Even a single faulty routing node or faulty channel in the network can affect many different paths.

In this chapter we present a simple and practical fault-tolerance scheme to overcome the problems caused by network faults. Our approach to dealing with faults is to send messages using two passes over the network when the direct path is faulty. Our approach is an extension to the work by Varma and Raghavendra [VR89] and Leighton et al. [LMS92]. Our fault-tolerance scheme has the following properties:

1. It requires no modification to the underlying network. We do not add additional links or extra stages to the butterfly. The design of the routing hardware remains the same as a network that does not have any fault-tolerance properties.

2. When new faults are discovered, the machine is halted and it enters a recovery phase. During the recovery phase, we compute two pass routes for messages whose paths are affected due to faults. At the end of the recovery phase, each processor will have a routing table that specifies these routes. Since we wish to maintain complete communication capability between all the *alive* processors, we may need to disable some processors to ensure this property. Our experimental results show that the number of processors lost is minimal.

3. The performance of the network suffers some degradation. The extent to which this

happens is quantified in Section 8.4.

In Section 8.1 we give a brief description of our fault-tolerance scheme and in Section 8.2 we describe the operations that are carried out during the fault-recovery phase. Section 8.3 discusses related work. Finally Section 8.4 presents a simulation based evaluation of our fault-tolerance scheme.

8.1 Approach

We model network faults as faults in the routing nodes. No messages may be routed through faulty nodes. We assume that faults are placed randomly and that the locations of the faulty nodes are globally known.

We denote the fact that the direct path from processing element (PE) p to q is fault-free by $p \rightarrow q$. $p \not\rightarrow q$ implies that at least one node along the path is faulty. If $p \not\rightarrow q$, then we attempt to find an alternate path for messages from p to q. This path consists of two passes through the network. The message first goes from p to some other PE, say r. The message is then sent from r to q. We call r the relay for messages from p to q. The selection of relays is done off-line during the recovery phase and stored in tables at each PE. Each PE maintains a table of size N which tells if the direct path to the corresponding destination is faulty, and if so, the address of the chosen relay. Since the selection of relays is done during recovery, all messages from p to q pass through the same relay r.

The routing nodes in the network route the messages based on routing tags that specify the destination of a message. In order to use the two pass scheme for sending messages, each message carries two addresses; the address of the relay and that of the final destination. During the first pass, the address of the relay forms the routing tag. When the message reaches a PE, it checks the destination address of the message. If the destination address is different from its address, it replaces the routing tag with the destination address and sends the message along.

8.2 Fault Recovery

During the recovery phase, we compute the routing tables for all the PEs. The computation of these tables is done in two parts. First we compute the set of *alive* PEs subject to the constraint that all of them can communicate with each other. Then, we select the relays in a manner that maximizes the throughput of the network.

We describe the computation of the routing tables for the network *Butterfly.32.8* that was discussed in Chapter 4 under model 1, but the computation readily extends to other networks as well.

8.2.1 Computing the *alive* set

We first compute the N by N direct connectivity matrix D. Matrix D is a boolean matrix with entry $D(p,q) = 1$ if $p \rightarrow q$, otherwise $D(p,q) = 0$. Each row of the matrix can be easily computed using a breadth first search from the corresponding source to figure out which destinations are reachable. Note that $D(p,q)$ need not be equal to $D(q,p)$ in general. This is because the path from p to q is different from the path from q to p and it is possible that only one of the two paths is faulty.

Next we compute the boolean matrix $T = D^2 \vee D$. $T(p,q) = 1$ if $D(p,q) = 1$ or there is a two pass route from p to q. Since we require complete communication capability for all the PEs in the *alive* set, we will disable PEs until the sub-matrix of T corresponding to the *alive* set is all 1s. Disabling a PE corresponds to deleting the row and column associated with that PE from the matrix T. Obviously, we would like to disable the minimum number of PEs, but finding the minimum set is a difficult problem. T represents a directed graph on N vertices as an adjacency matrix. In order to compute a maximum *alive* set, we need to find the maximum clique in the directed graph represented by T. In general, finding the maximum clique in a graph is an NP-complete problem, but we find that when the number of faults is small, the following greedy algorithm yields large cliques and is acceptable in practice.

We first transform this directed graph into an undirected graph by replacing each pair of back-to-back edges by a single undirected edge. Unpaired edges get deleted. We sort the vertices of this undirected graph by vertex degree and eliminate vertices in increasing order of vertex degree until we are left with a complete graph. The first vertices to get eliminated are the isolated vertices. The isolated vertices correspond to the processing elements that are connected to faulty switches in the first or last columns. When the number of faults is small, we find that we are usually left with a complete graph after the isolated vertices are eliminated. When the elimination of isolated vertices does not result in a complete graph, the number of additional vertices that need to be eliminated is usually very small. It may be possible to retain these vertices in the *alive* set if we allow messages to make more than 2 passes through the network, but increasing the number of passes degrades network throughput. Our results indicate that the increase in the size of the *alive* set does not offset the throughput degradation nor the added complexity.

8.2.2 Relay selection

After computing the *alive* set we select relays for all PE pairs that cannot communicate directly. Let the set $R_{p,q} = \{r \mid p \rightarrow r \wedge r \rightarrow q\}$. Any element of $R_{p,q}$ can serve as a relay for the messages from p to q. However, the selection of relays affects the throughput of the network. For example, we do not want our relay selection algorithm to send a large number of messages through a single channel or a single PE. Our objective is to select relays so that network throughput is maximized under random communication.

Recall that the best network architectures had wider channels at the lower levels of the packaging hierarchy and narrower channels at the higher levels of the hierarchy. Therefore, the channels at the upper levels of the hierarchy are critical as far as network throughput is concerned and maximizing throughput requires balancing the load on these critical channels. For the network *Butterfly.32.8* the critical channels are those that go between the modules. The load on a critical channel x, L_x, is the expected number of messages that pass through the channel when each PE sends a message to a randomly chosen destination. Let $\hat{L} = \max_x(L_x)$. $\hat{L}$ is the worst case load and we use the function $1/\hat{L}$ as an estimator of network throughput.

In a fault-free network, since there are N PEs, the probability that any given pair of PEs will communicate is $1/N$. Each direct path between any two PEs (in *Butterfly.32.8*) will contain exactly one critical channel and the number of message paths that pass through any critical channel will be equal to N. The load on each critical channel will be 1. In the presence of faults, some messages require 2 passes through the network. These messages will contribute to the load on two critical channels, one for each pass. Also the number of alive PEs (denoted by N_A) will typically be less than N, therefore the individual communication probabilities will be $1/N_A$. Given an assignment of relays we first determine the number of message paths that pass through each critical channel x. The number of paths divided by N_A gives L_x.