ELSEVIER

Simulation Practice and Theory 7 (1999) 251–278

SIMULATION PRACTICE AND THEORY

www.elsevier.nl/locate/simpra

Computer simulation of PEC network [1]

Xin Liao [a], Xian-He Sun [b,*]

[a] *Solaris Clustering Group, Sun Microsystems Inc., Menlo Park, CA 94025, USA*
[b] *Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803-4020, USA*

Received 11 November 1997; received in revised form 1 January 1999

Abstract

Interconnection network is a decisive component of parallel and distributed computer systems. With the merits of simplicity and efficiency, 2-dimensional (2-D) mesh has been a popular choice of large MIMD interconnection networks. Mesh, however, has its known weaknesses in scalability and connectivity. *Packed Exponential Connections* (PEC) is a newly proposed network which is designed to improve the scalability and connectivity of 2-D mesh while maintaining its merits. In this study, the performance of PEC over mesh network is carefully examined through computer simulation. Characteristics of PEC networks are revealed. A novel routing scheme is proposed and used in $\mathcal{PROTEUS}$ environment to simulate the performance of 2-D PEC network. Simulation and analytical results show that for many applications where non-local communications are required, PEC network provides superior performance to that of mesh. Based on simulation results, structural modification is also suggested to further enhance the performance of PEC network. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Parallel architecture; Simulation; PEC network; Routing scheme; Performance analysis

1. Introduction

Performance study is an important aspect of parallel system research. It can be used to select the best architecture platform for a class of applications, to choose the best algorithm for solving a particular application on a given hardware platform, predict the performance of large application instances, to identify application and architecture bottlenecks suggesting application restructuring and architectural en-

* Corresponding author. E-mail: sun@csc.lsu.edu.

[1] This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-1672, by National Science Fundation under NSF award ASC-9720215, by Louisiana Education Quality Support Fund, and by LSU under 1998 COR award.

PII: S0928-4869(99)00005-1

hancements, and to gain insight on the interaction between an application and an architecture predicting the performance of other application-architecture pairs.

There are three commonly used techniques in performance evaluation. They are *experimentation, theoretical and analytical modeling* and *software simulation*. Parallel and distributed computer systems are quite complex. Accurate modeling of a parallel system is very difficult to accomplish. On the other hand, building a real machine for performance experimentation is expensive at best and unrealistic for most situations. For parallel computing, simulation has played an important role at all levels of the design and analysis of multiprocessor systems, ranging from architectures to runtime systems and from algorithms to applications.

Packed Exponential Connections (PEC) is a newly proposed network which has gained intensive attention recently [8,9,11]. $\mathcal{PROTEUS}$ [2,3] is a simulation system which can provide fast and accurate simulation for parallel algorithms. In this research, the performance of PEC network is studied via simulation on $\mathcal{PROTEUS}$ system. The characteristic of PEC network is examined. A practical routing scheme, $R-Route$, is proposed, analyzed, and simulated solving two applications on PEC and mesh network. $\mathcal{PROTEUS}$ software system is appropriately enhanced to carry the simulation of the PEC network. Simulation results of PEC and mesh network are then compared and analyzed. Simulation results confirm analytical results: PEC network is a promising network; it provides a superior performance to mesh for applications requiring global communications, especially when system ensemble size and problem size are large. Based on simulated performance, modification of PEC network is also proposed to further enhance the performance.

This paper is organized as follows. The structure and functionality of $\mathcal{PROTEUS}$ is introduced in Section 2. A detailed description of PEC network is presented in Section 3. Properties of 1-D and 2-D PEC network are analyzed and the similarity and difference between PEC and mesh are also described. In Section 4 a 1-D PEC routing scheme, *R-Route*, is introduced and analyzed. Based on it, a 2-D PEC routing algorithm is proposed and implemented in $\mathcal{PROTEUS}$ environment. In Section 5 two binary-exchange scheme algorithms, namely all-to-all broadcast and Fast Fourier Transform (FFT), which are used in the simulation study, are introduced. Section 6 presents the simulation results of PEC and mesh network. Comparison and analysis of the performance are also provided. Finally, Section 7 gives the conclusion and summary.

2. The $\mathcal{PROTEUS}$ simulator

$\mathcal{PROTEUS}$ is a parallel system simulator developed at MIT [2,3]. In $\mathcal{PROTEUS}$ environment, each simulated parallel processing node consists of a processor, a network interface, a cache, and a memory. The processor is a sequential processor extended with instructions for network access and cache coherence. The network interface connects the processor with the interconnection network. The cache, which is optional, can be used to handle cache coherence and to provide remote memory access. The memory at each node is divided into two modes, a *shared*

mode and a *private* mode. The *shared* mode maps to the global address space, which can be accessed from other processor nodes via interconnection network. The *private* section is not accessible from the interconnection network. If the memory of each processor is all in *shared* mode and memory access time is uniform, the structure is a COMA Model [4]. If we consider the effect of interconnection network, it is a NUMA model. If all the memories of processors are in *private* mode and no global address space is used, then it is the message-passing model.

To simplify the replacement and adjustment to different applications, $\mathcal{PROTEUS}$ is designed with a modular structure. It includes operating system modules, shared memory modules, cache modules, data collection modules, and network modules. The operating system module provides simulated kernel interface on parallel environment, such as thread scheduling and management, memory management, and interrupt and trap handling. It also provides InterProcessor Interrupts (IPI) and handlers, which serves as message handler facility in message-passing architecture. Shared-memory module provides access to local shared memory, handles memory management, and provides atomic synchronization operations such as test-and-set. Data collection module is responsible for information collection and display. Network module simulates the movement of data within the interconnection network.

In general, a parallel-system simulator may face potential drawbacks on two fronts – speed and accuracy. To achieve accuracy, $\mathcal{PROTEUS}$ applies execution-driven mode, which ensures network contention ordering of program events and permits accurate simulation of contention and process interaction. To achieve high speed, $\mathcal{PROTEUS}$ avoids interpreting user application code whenever possible, thus removing the overhead of interpretation for most instructions. It is designed to make the entire system run in a single address space. $\mathcal{PROTEUS}$ also provides users high flexibility in choosing or customizing the level of accuracy and gives users the control of tradeoff between accuracy and speed. These and other features make it faster and more accurate than most existing simulators.

In addition to performance, a primary asset of $\mathcal{PROTEUS}$ is its support for monitoring and debugging. $\mathcal{PROTEUS}$ provides an internal debugging mode called ''snapshot'', which allows users to examine the status of threads, processors, locks and memory. $\mathcal{PROTEUS}$ also provides *repeatability*, so that the user can re-run the simulations to find out bugs. $\mathcal{PROTEUS}$ provides an integrated subsystem for system configuration, data collection and result display. Its graphic capabilities make it simple in system configuration and in performance evaluation of algorithms and architectures. Like other simulators, however, limitations exist. Though $\mathcal{PROTEUS}$ is a high-level fast simulator compared to its counterparts, it still takes long time to simulate comparatively complex algorithms. One reason for the long running time is that $\mathcal{PROTEUS}$ is a sequential program. Its parallel version is not available. The other limitation of $\mathcal{PROTEUS}$ is that it does not provide options for recording simulation results. The simulator automatically records much information about the execution. When massively parallel processing (number of processors > 500) is simulated, information recording will become very resource consuming.

3. Packed exponential connections network

Many factors contribute to the design of an interconnection network. In addition to speed and connectivity, simplicity and scalability are two other important concerns in the network design [10]. Some networks such as complete connection network may provide high analytical performance. But, difficulties in VLSI fabrication and hardware scalability render them impractical when building actual parallel computers. A simple and effective interconnection network is the mesh. Mesh, however, lacks support for long distance connections which makes it very inefficient when system ensemble size is large and non-local communication is needed.

PEC network is a new interconnection network that solves the problems of scale and connectivity by augmenting a two dimensional mesh with additional long connections [5]. It has very promising characteristics for supporting large scale MIMD parallel processing.

3.1. 1-Dimension PEC network

Lin and Prasanna [8] have studied the properties of 1-D PEC network. The following definitions are based on those in Refs. [5,8]. When the number of nodes is $N = 2^n$ indexed from $0, 1, \ldots, N-1$, we have the following definitions.

Definition 1. A node with index $i = q2^h + 2h - 1$ has PEC value h, where h is a positive integer, and q is a non-negative integer. We say $PEC(i) = h$.

Note that the PEC value of a node i is the least significant position of i where "1" appears. The rightmost bit position is counted as position 1.

Fig. 1 shows a 1-D PEC network of size 16. Note that the subgraph obtained by deleting node 0 has a hierarchical structure. This is also denoted as *SPEC network*.

Definition 2. A PEC network is a graph $G = \langle V, E \rangle$, where $V = \{0, 1, \ldots, 2^{n-1}\}$ and E is a set of links, $E = \{(i,j) | (i = j \pm 1)$ or $(i = j \pm 2^h$, and $PEC(i) = PEC(j) = h)\}$, where i and $j \in V$.

In general, a node i with $PEC(i) = h$ has at most four links which connect it to nodes $i+1, i-1, i+2^h$, and $i-2^h$. Fig. 2 shows an alternative representation of 1-dimension PEC network of size 32.

3.2. 2-D PEC network

In a rectangular mesh network, each node has four nearest neighbor connections (E,W,S,N). PEC adds four additional connections that are part of a second 2-D mesh of a different scale. The nearest neighbor mesh will be called $PEC(0)$. A mesh that connects every second node is a 2-D mesh of $PEC(1)$. In general, a 2-D mesh of $PEC(k)$ will connect the processors 2^k neighbors away. 2-D PEC has several proper-

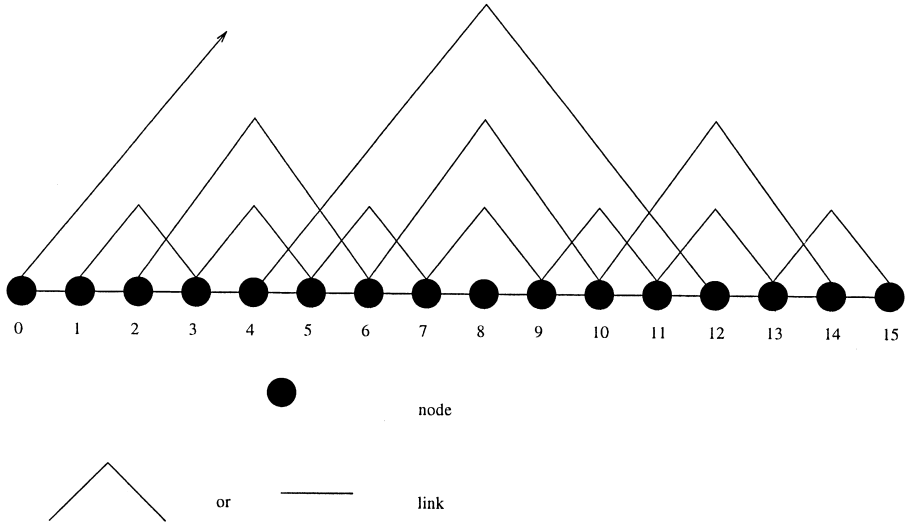


Fig. 1. A 16-node 1-D PEC network.

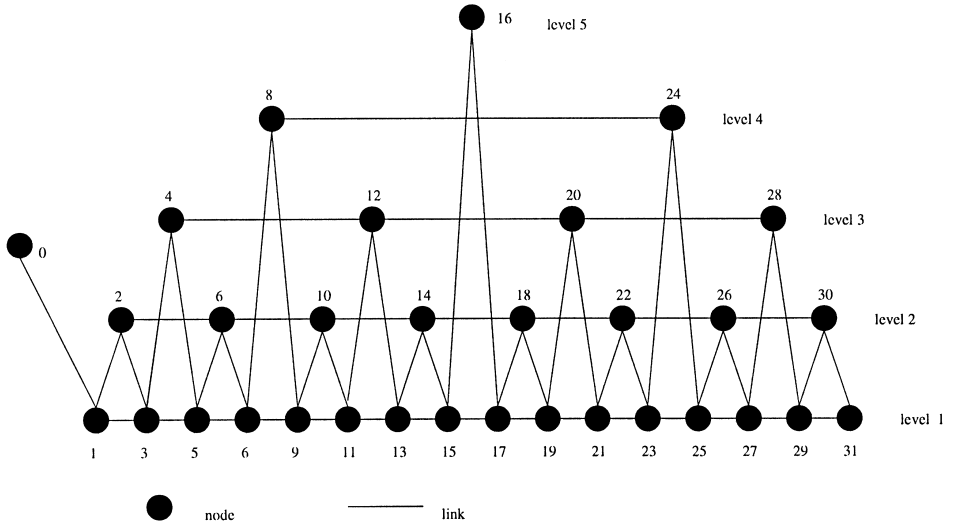


Fig. 2. An alternative representation of a 32-node PEC network.

ties which make it an attractive alternative to other networks. Fig. 3 shows a 16×16 2-D PEC network [5]. Definitions 3 and 4 give two primary concepts of 2-D PEC. Proposition 1 presents the single most important property of 2-D PEC connectivity [9,11].

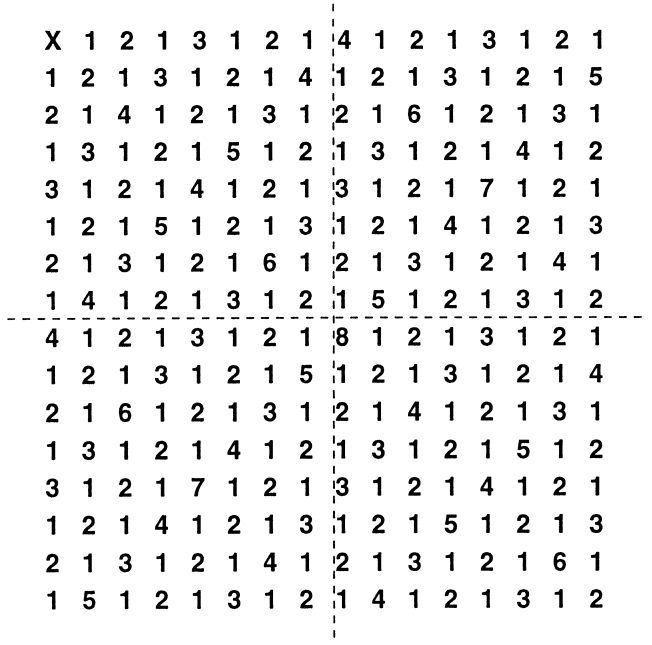


Fig. 3. An example of 16×16 2-D PEC.

Definition 3. A processor farm groups the nodes in one area of the processor array, and uses these processors together to implement a function, or a set of functions within the context of a larger application.

Definition 4. A tile to be a rectangular farm extending 2^j processors in one direction and 2^k in the other. A tile is properly aligned if and only if it's upper left processor is offset from PEC origin by $n2^j$ in the first direction and $m2^k$ in the other (n and m are positive integers).

Fig. 4 shows the aligned tile. With the definition of *farm* and *tile*.

Proposition 1. *In a 2-D PEC network, every 2^j by 2^k properly aligned tile has exactly one connection of length 2^{j+k} and exactly one that is longer.*

4. Simulation design

The heart of parallel-system simulation design is the design of routing schemes. The introduction of a 1-D routing scheme, extension of the new routing to 2-D PEC, and incorporation of the new scheme into $\mathscr{PROTEUS}$ simulation environment are presented in the following four sections, respectively.

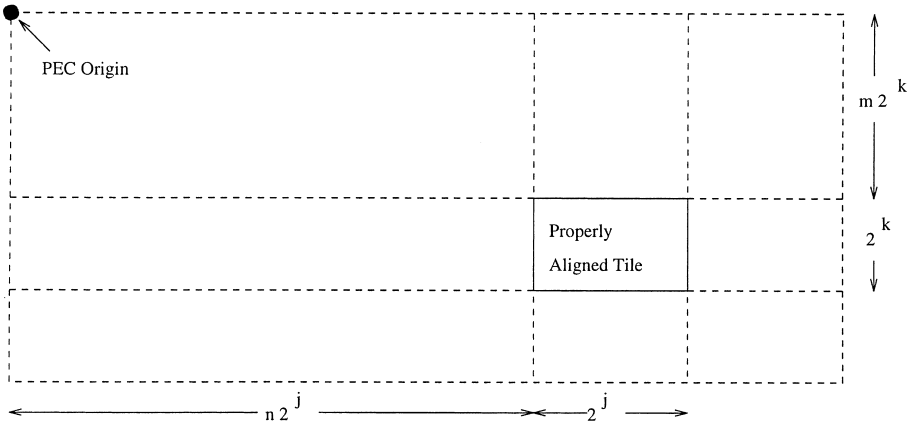


Fig. 4. Properly aligned tiles of 2-D PEC.

4.1. A routing scheme on 1-D PEC

Lin and Prasanna [8] have proposed the *G-Route* which can set up a path from node 1 to node i in the PEC graph with $2^n - 1$ nodes, where $1 \leqslant i \leqslant 2^n - 1$. *G-Route* reaches the theoretical lower bound of routing on 1-D PEC network. However, it has its drawbacks. First, instead of providing a general routing scheme from node i to node j, *G-Route* only gives out the route scheme from node 1 to node j. Secondly, *G-Route* requires the user to present the optimal characteristic set, H_n, which is not easy to acquire. When the PEC network is not a *perfect* PEC network, that is when the network size is not 2^n but $m2^n$ (for example, 24, 48, etc.), determining an optimal characteristic set is extremely difficult, and therefore the applicability of *G-Route* is very limited in practice [2].

On 1-dimension PEC, a routing path from node i to node j can be determined recursively when we know at each stage the longest link it takes. *R-Route* routing scheme proposed in this study is based on this property. Assuming i and j are the source and destination node on 1-dimension PEC, respectively, and assuming the longest link between i and j is the link that goes through nodes with PEC level equals k, where $k \geqslant 0$, then *R-Route* can be generated using the following steps:

1. Put all the links through level $t = k$ nodes between node i, j into $PATH$, and mark remaining links between node i, j unresolved.
2. Search for links through level $t = t - 1$ nodes from the unresolved links that do not overlap with any link in $PATH$. If there is any, put them into $PATH$.
3. Repeat Step 2 until both the nodes i and j are reached by $PATH$.
4. $PATH$ is the *R-Route* routing path.

[2] Although we can combine routing i to 1 and 1 to j together to make an i to j routing scheme, it will make node 1 the bottleneck in communication.

An example of *R-Route* path is given in Fig. 5. There are two major differences between *G-Route* and *R-Route*: firstly, there is a trace-back operation in *G-Route* but not in *R-Route*; secondly, a characteristic set is needed in *G-Route* but not in *R-Route*.

Considering the routing from node 1 to node $N-1$ on a *perfect* 1-dimension PEC which has size 2^n, we can find that the longest link in the routing path divides the PEC into two symmetric sub-networks, where each has size 2^{n-2}. Obtaining one longest link on each of the subnetwork recursively, we get the links with distance $2^{n-4}, 2^{n-6}, \ldots, 2^0$ (see Fig. 6). Using the two 0-*layer* links in PEC to connect the links between different levels, we can generate the following recursive characteristics:

$$\mathrm{hop}(H_i) = \begin{cases} 2(\mathrm{hop}(H_{i-2}) + 1) + 1 & \text{if } i > 4, \\ 3 & \text{if } i = 3, \\ 1 & \text{if } i = 2. \end{cases}$$

Using a general method to simplify the recursive function [1], we can obtain the complexity of *R-Route* routing scheme.

Theorem 1. *Using R-Route routing scheme, the up-bound steps of 1-dimension PEC is* $O(\sqrt{N})$, *where N is the number of nodes on the 1-dimension PEC network.*

Proof. We consider the longest path in 1-D PEC, from node 1 to node $N-1$. From the above analysis we can get the following recursive function:

$$n_i = 2(n_{i-2} + 1) + 1,$$

we have:

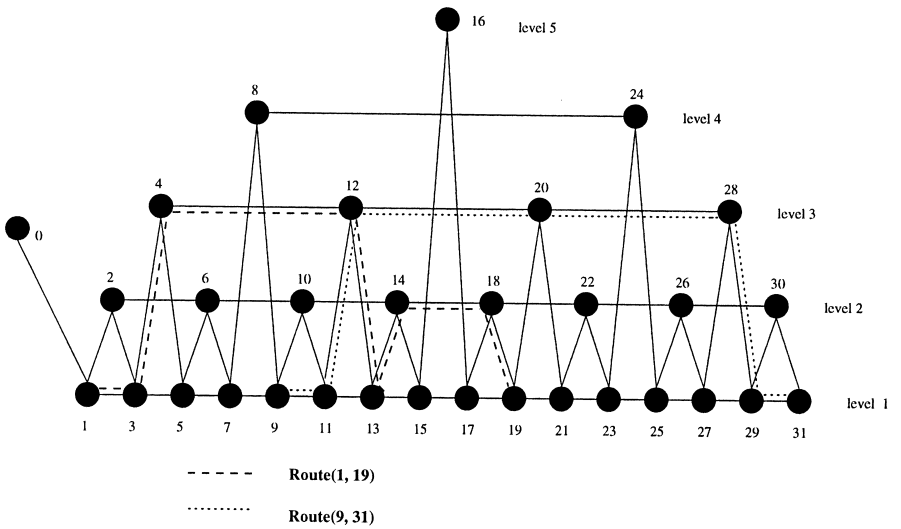


Fig. 5. An *R-Route* path for (i, j) pair of nodes.

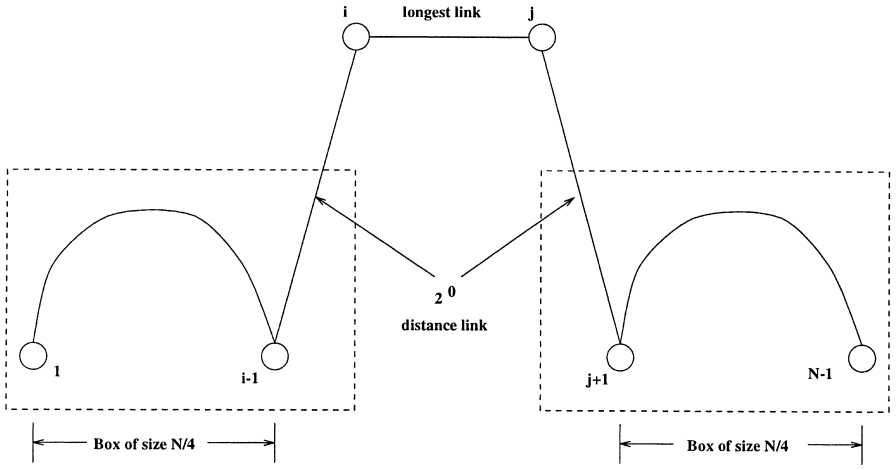


Fig. 6. *R-Route* scheme analysis.

$$
\begin{aligned}
n_i &= 2(n_{i-2}+1)+1 \\
&= 4(n_{i-4})+2\cdot 3+1 \\
&= 8(n_{i-6})+4\cdot 3+2\cdot 3+1 \\
&\vdots \\
&= 2^{i/2-1}+3\cdot 2^{i/2-1} \\
&= 2\cdot 2^{i/2}.
\end{aligned}
$$

We have reached the conclusion. □

Fig. 7 gives the implementation of *R-Route*. It consists of three steps.

1. Record the PEC value of all nodes between "from" and "to".
2. Sort the PEC values and find the highest PEC value among the PEC values where more than one node hold (find the furtherest jump).
3. Put the nodes into PATH, and apply the algorithm recursively to the un-solved part of the path.

Fig. 7 also lists the maximum steps to finish the *R-Route* from node 1 to node $N-1$.

The most salient feature of *R-Route* is it does not require the knowledge of a *characteristic set* [8]. The other advantage is it can be applied to an *imperfect* PEC network.

4.2. 2-D PEC routine scheme

PEC network is a relatively new network. Optimal routing schemes on PEC networks are not available, especially for 2-dimension or higher PEC networks. By theoretical analysis, an optimal route between *farms* in a PEC network could give routing distances similar to those found in a binary tree [9]. Using high level connections for optimal routing schemes is still a subject of current research [5]. There is no

```
int     pec_tab[SIZE_OF_PEC]; /*save PEC value*/
struct     record {
  int        count;
  node       site[SIZE_OF_PEC];
} recordOccur[HIGHEST_PEC_VALUE]

R-Route(from, to) /*assume "from" < "to"*/
begin /*save PEC values*/
      for(i = from; i ≤ to; i++)
      begin
            for(j = 0; j ≤Highest_PEC_Level; j++)
            begin
                  if ( j=pec_tab[i]) then
                  { recordOccur[j].count++;
                    insert(i, recordOccur[j].site);
                  }
            end
      end
      for(i =SIZE_OF_PEC; i > 0;i++)
      begin /*find the longest jump path*/
            if (recordOccur[i].count > 1) then
            { save(recordOccur[i].site, PATH);
              k = i; break;
            }
      end
      if (k = 0) route to 'node 0'; /*trivial case*/
      R-Route (from, lowest_node(PATH)) /*recursive*/
      R-Route (highest_node(PATH), to) /*recursive*/
end{R-Route}
```

Fig. 7. *R-Route* algorithm.

near optimal routing scheme available on 2-D PEC. Fig. 8 gives an intuitive sense of the complexity to utilize high level PEC connections on 2-D PEC. The 2-D PEC routing algorithm has the following three step structure:

1. Check the distance on both X and Y.
2. Choose the dimension with a longer distance first for 1-D route.
3. Move forward, repeat Steps (1) and (2) until distance is 0 (reach destiny).

In our simulation, we assume that the X–Y routing will be accomplished with each *farm*. That means a data value on a given processing node will be sent to a target

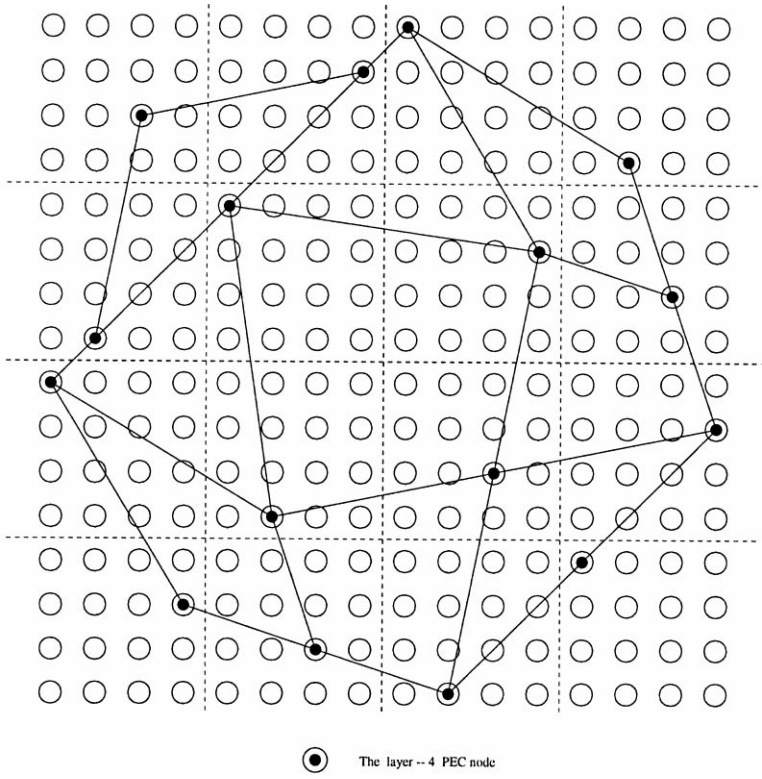


Fig. 8. Layer-4 connection on 16×16 PEC.

node by the following horizontal and vertical paths (or vice versa) which utilize either nearest neighbor or 1-D routing contributions.

Fig. 9 gives the 2-D routing program. To avoid generating high contention in a particular direction, the program chooses the direction with the longest distance as the first routing direction. Based on Theorem 1, we can find that the complexity of the extended *R-Route* on 2-D PEC is $O(N^{1/4})$, where N is the number of nodes in the network. Note that the complexity result matches the analytical results of Wong et al. (see Section 3 and [11]).

4.3. Embedding the routing scheme into *PROTEUS*

The network modules in *PROTEUS* [2] provide the functionality to simulate data movement in an interconnection network. Instructions that affect remote nodes are implemented using simulator *request*. The network module uses three types of requests. The first is *send* request, which signifies that the processor is ready to send a packet to the network interface. The second type of request is *route* request, which determines the next node for a packet, computes the arrival time of the packet, and calculate the contention involved on each message. The third type is *receive* re-

```
routetab_mk(int from, int to, int *mark)
begin
      decide distance on X-Y direction
      while( distance ≠ 0)
      begin
            if ( distance_X > distance_Y)
            begin
                  chose X-direction do 1-D route.
                  index moved forward.
            end
            else
            begin
                  chose Y-direction do 1-D route first.
                  index moved forward.
            end
      end{ while}
      generate route-table file.
end
```

Fig. 9. 2-D PEC routing algorithms.

quest, which occurs when the packet reaches the target node. The receive request either interrupts the processor or notifies the cache depending on the packet. In *PROTEUS*, only the network module generates route and receive requests; all other modules generate only send requests.

The route request decides the next node to which the packet should be forwarded. For example in a k-ary n-cube the route request determines, based on the target node, which output link to use, the incoming link, etc. New release of *PROTEUS* provides a *router* function from which users have an interface to define the special interconnection. The function resides in the program `net.exact.c`. The format of the function is:

```
default_router_(int from, int to, int prev, int, curr, int *line)
```

from is the number of source node, to is the target node, prev is the last position, curr is the current position, and *line is the list of dimensions passed. The return value of this function is the next node position corresponding to curr.

default_router_() is called each time when there is a single data movement from one node to a neighbor node. To facilitate the simulation speed, the PEC routing scheme could be generated beforehand and stored in a 2-D file – -routetab. The contents of routetab is generated by a separate program using *R-Route* method. So it avoids the tedious procedure to do recursive searching at each step of the PEC routing. As a contrast, we also implement mesh routing scheme in the default_router_().

Because mesh routing is direct and easy to execute, its routing function is put inside default_router_() [2,6].

The number of messages passed and the total contention involved on each node are the two parameters used in our performance evaluation. *PROTEUS* does not provide a means of cataloging the number of messages passed on each node. For contention delay, though *PROTEUS* provides a graphic display, it is very brief and inaccurate. To record the statistic, a variable and a macro are set up in file net.exact.c. Global variable HOPS is used to record the number of messages passed on each node. Macro NET_CONTENTION(p, t, b) is used to register all the contention produced on each node.

5. Application and analytical results

Two binary-exchange scheme algorithms, namely all-to-all Broadcast and FFT, have been implemented to test the performance of PEC network. A generic binary-exchange computation can be defined as the one that repeatedly uses data values that are a power of two apart. Given an initial set of n data values, $a_0, a_1, \ldots, a_{n-1}$, we consider patterns of processing that involve pairs of the form $a'_i = a_i \otimes a_{i+2}$ and $a'_{i+2^k} = a_i \otimes a_{i+2^k}$ for $i = j \ldots j + 2^k - 1, j = 0 \ldots, 2^{k+1}, 2 \cdot 2^{k+1}, 3 \cdot 2^{k+1}, \ldots,$ where

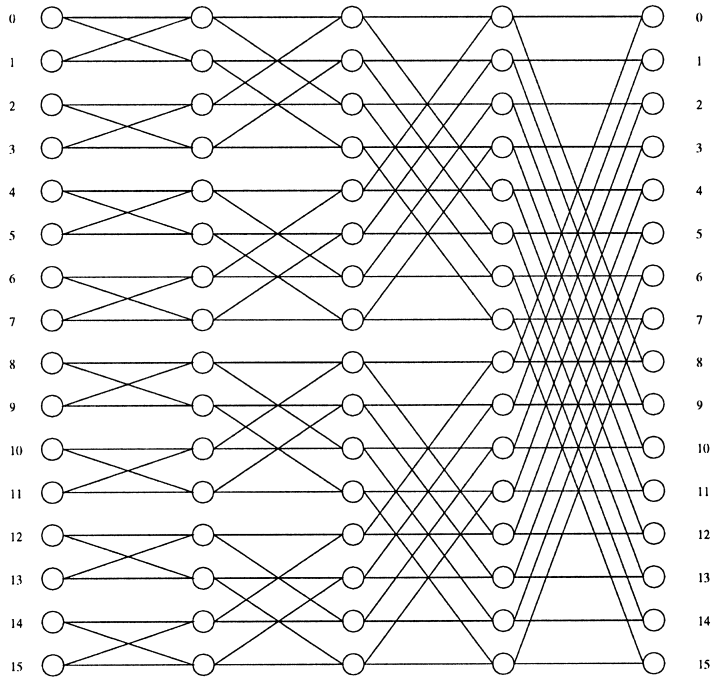


Fig. 10. Binary-exchange scheme.

the operator $x \otimes y$ denotes any arithmetic, comparator, or set operation. The range of k can be either from 0 up to $(\log_2 n) - 1$, or vice versa [11].

The main feature of binary-exchange scheme is the communication pattern required for completion. As we have mentioned above, if a_i and a_{i+1} are close to each other, then a_i and a_{i+2^k} will be considered far from each other (see Fig. 10). If the computer system has a shared single memory, then the time required to access either their nearest neighbor or remote nodes is the same. However, this is not true for distributed memory machines. On distributed systems, latency of local and remote data access is different, and the performance of a binary-exchange algorithm varies with network support. In addition, binary-exchange is a widely used computation pattern of scientific and non-scientific applications. They are good candidates for testing the performance of PEC networks.

5.1. All-to-all broadcast

In one-to-all broadcast, a single value that resides on one of the PEC nodes is to be copied to all other nodes. In all-to-all broadcast (also called total-data-exchange), a single value resides on each of the PEC nodes is to be copied to all other nodes. One way to accomplish all-to-all broadcast is to conduct one-to-all broadcast on each node concurrently and send the value using $n + 1$ phases as discussed in Section 3. The all-to-all broadcast algorithm, which is implemented on each node, is listed in Fig. 11. Since each node will hold all the N data at the end, at least N memory locations have to be allocated on each node before data transfer, where N is the total number of nodes in the PEC network.

5.2. Fast fourier transforms

FFT plays an important role in many scientific applications, including time series and wave analysis, solutions to linear partial differential equations and image filtering. The computation pattern of sequential FFT algorithm is a reversed binary-exchange (called butterfly) scheme. Three different data layout exist for parallelizing sequential FFT algorithms: cyclic layout, block layout, and hybrid layout. Cyclic layout assigns the $(i + j \times N)$th row of the butterfly to the ith processor where N is the number of nodes in the network and j is a non-negative integer. Block layout places the first m/N rows of the butterfly on the first node, next m/N rows to the second node, and so on, where m is the problem size. Hybrid layout starts with cyclic layout and switches to block layout in the middle of the computation [7]. Both cyclic layout and block layout need one local computing, at the end and at the beginning, respectively, and compute at each stage of the butterfly. The hybrid layout consists of two local computings and one global data transpose. Due to the transpose communication, hybrid data layout based algorithms are also called *transpose algorithms* [7]. Transpose algorithms provide a better performance than that of the other two approaches when the problem size is large. Our implementation is based on the hybrid approach, and the data transpose is accomplished by using all-to-all personalized broadcast. All-to-all personal-

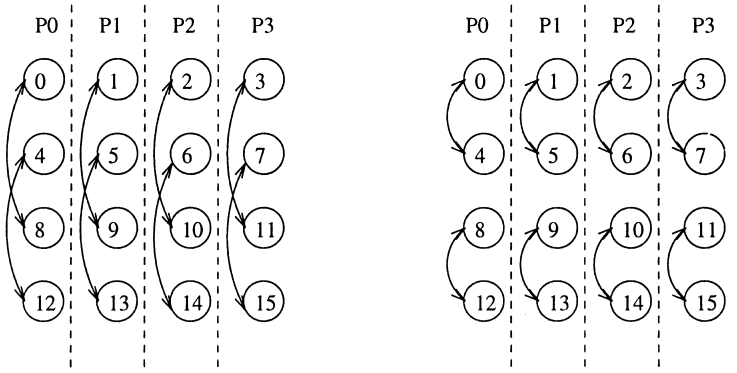
```
one_to_all(int phase, int MyNum, int *array)
{
    step = 2**phase;
    temp = (int *) malloc(N*sizeof(int));
    if ((MyNum % rootP) % (step*2) < step)
      exchange_mem(phase, MyNum + step, array, 0, temp);
    else
      exchange_mem(phase, MyNum - step, array, 0, temp);
    barrier(record, NO_OF_PROCESSORS); /* set sync barrier */
    trigger(phase, MyNum, 0, temp); /*update mem at same time */
    barrier(record, NO_OF_PROCESSORS);
    if (MyNum/rootP)%(step*2) < step);
      exchange_mem(phase, MyNum + step*rootP, array, 1, temp);
    else
      exchange_mem(phase, MyNum - step*rootP, array, 1, temp);
    barrier(record, NO_OF_PROCESSORS);
    trigger(phase, Mynum, 1, temp);
    barrier(record, NO_OF_PROCESSORS);
}
exchange_mem(int phase, int node, int *array, int *temp)
{
    ready_count = N/P*(2**(phase*2)) /*N: No.of elmts, P: No.of Nodes */
    if(level == 0)
      for(i=0;i<ready_count;i++)
         temp[i] = @(x+N*node+i);
    else
      for(i=0;i<ready_count*2;i++)
         temp[i] = @(x+N*node+i);
}
```

Fig. 11. All-to-all broadcasting algorithm.

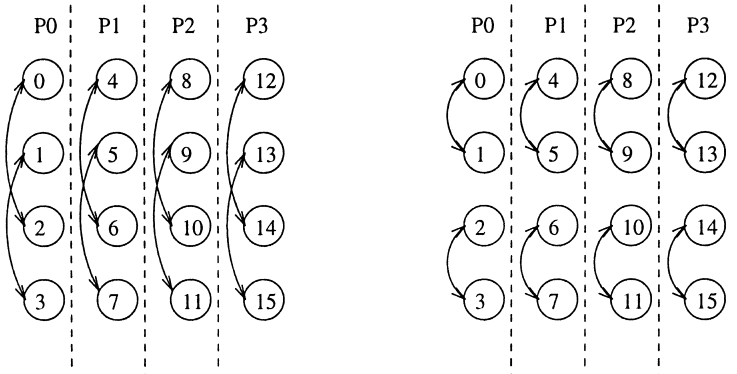
ized broadcast has the same communication pattern as the all-to-all broadcast, the binary-exchange pattern, while with an increased data size in communication.

Fig. 12 gives the steps of 2-dimensional transpose on 16 node FFT on four processors. Transpose algorithm has the property that, if the $\sqrt{n} \times \sqrt{n}$ array of data is transposed after computing the $\sqrt{n}$ column FFTs, then the remaining part of the problem is to compute the $\sqrt{n}$-point column-wise FFTs of the transpose data. The transpose algorithm applies this property to compute the FFT in parallel by using a column-wise striped partitioning to distribute the $\sqrt{n} \times \sqrt{n}$ array of data among the processors [7].

Notice that in Fig. 12, in the first and third phases of the algorithm, each processor computes $\sqrt{n}/N$ FFTs of size $\sqrt{n}$ each, where N is the number of processors. The second phase transposes the $\sqrt{n} \times \sqrt{n}$ data block, which is striped column-wise among N processors.



(a). Steps in Phase 1 of the transpose (before transpose)



(b). Steps in Phase 3 of the transpose (after transpose)

Fig. 12. Transpose steps.

6. Simulation results and analysis

Simulations have been conducted on $\mathscr{PROTEUS}$ [2] environment to simulate performance of 2-D PEC and mesh network with 64 nodes and 256 nodes. Since data transfer is the main concern of performance evaluation, two different statistic graphs are presented below for network contention and sum of dataflow hops which occurred on each processing node, respectively.

For all the graphs, we assume m to be the total number of data being accessed, Y-axle shows the respective parameter it measured while X-axle shows the number of processors. The system parameters used in $\mathscr{PROTEUS}$ are:

Network based	Direct network
Bidirectional	No caches
k-ary radix	8(16)
Dimension	2

Number of Pro	64(256)
Switch delay	6
Wired delay	8
Bits in memory size	19
Memory modules	64(256)
Memory access time	7
Exact simulation	Yes
Sync. send/receive	No

6.1. Simulation result

Two sets of graphs are presented to demonstrate the dataflow hops and contention on each processor respectively. Dataflow hops on each node are the total number of messages which have been sent out by the node, including both messages generated and messages passed. This parameter gives a general idea of the "intensity" of usage for each node. Contention on each node is the sum of time taken by all messages waiting to be sent out on the node. This parameter gives a general idea of the contention involved in each node. Two different data sizes and two different ensemble sizes are used for testing. The simulated results on PEC and on mesh are presented side-by-side for easy comparison.

6.2. Performance analysis

Figs. 13–24 exhibit the node-by-node, detailed performance of PEC and mesh network. They provide a means to pine out the performance bottlenecks. For comparison of overall performance, three additional pairs of graphs are given in this section to depict the sum of dataflow hops and sum of contentions occurred during the

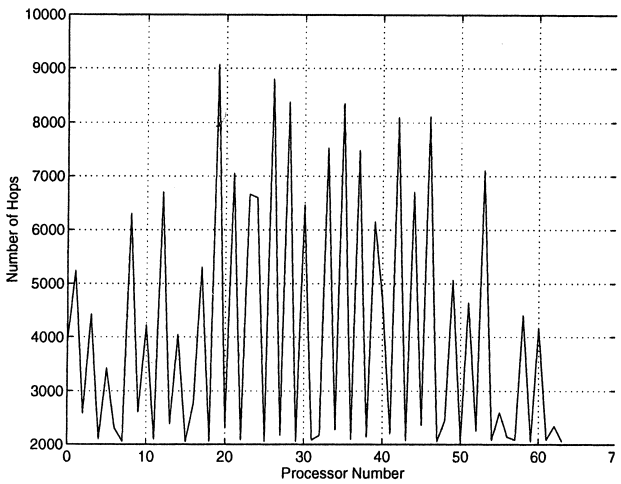


Fig. 13. 2-D PEC network dataflow hops of each processor all-to-all broadcast, 64 nodes, $m = 2^{10}$.

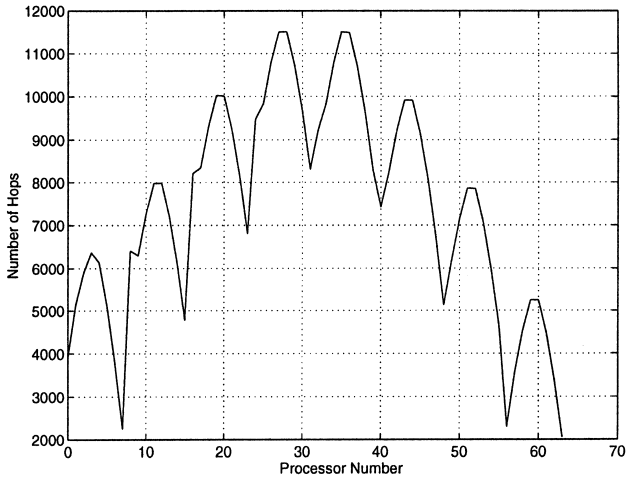


Fig. 14. 2-D mesh dataflow hops of each processor all-to-all broadcast, 64 nodes, $m = 2^{10}$.

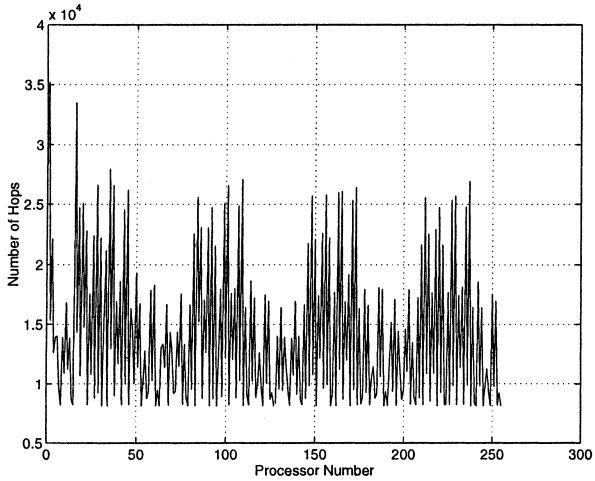


Fig. 15. 2-D PEC network dataflow hops of each processor all-to-all broadcast, 256 nodes, $m = 2^{14}$.

simulation. In the figures from 19 to 21, solid line "——" represent PEC results, while dashed line "- - -" represents mesh results. *X*-axle shows *m* where 2^m is the total number of data accessed. *Y*-axle shows the respective metric value.

6.2.1. All-to-all broadcast

As discussed in Section 5, all-to-all broadcast transfers local data of each node to all nodes on the network. From Fig. 13 to Fig. 16 and from Fig. 17 to Fig. 20, we observe the following:

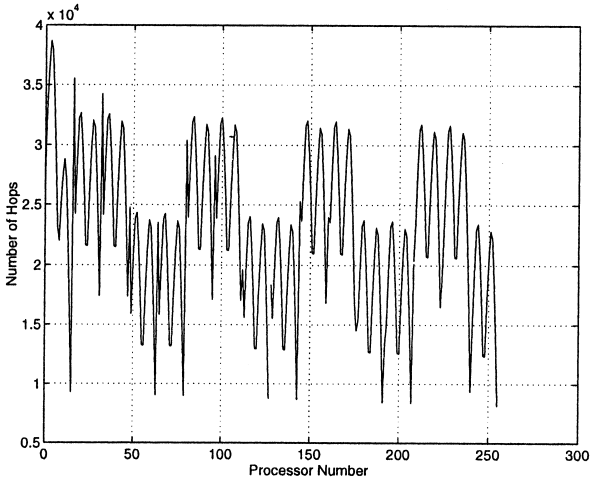


Fig. 16. 2-D mesh dataflow hops of each processor all-to-all broadcast, 256 nodes, $m = 2^{14}$.

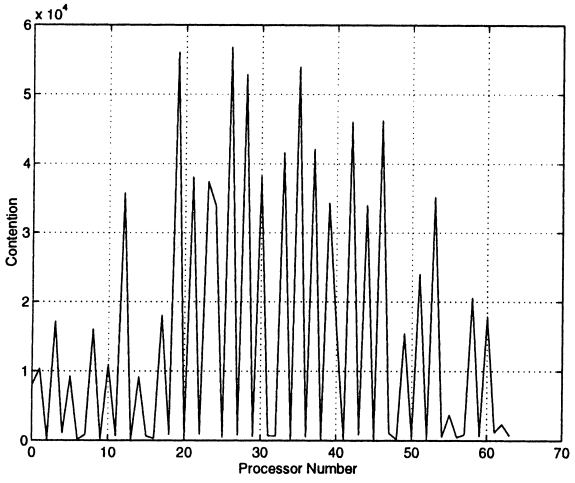


Fig. 17. 2-D PEC network contention on each processor (64 nodes, $m = 2^{10}$) all-to-all broadcast, 64 nodes, $m = 2^{10}$.

- On the dataflow figures, the shape of PEC is much steeper than that of mesh. The reason for this is that on a PEC network, because *R-Route* utilizes the longest path on each X and Y direction as much as possible, the chance that a message route through a higher level link on each X- and Y-dimension is higher than through a lower level link. On mesh, however, all nodes have the same four nearest neighbor connection and no long jump route exists.
- The sum of dataflow hops on all nodes of PEC and mesh network are plotted in Figs. 25 and 27. Fig. 26 shows the total dataflow hops on 64 processors. Fig. 27

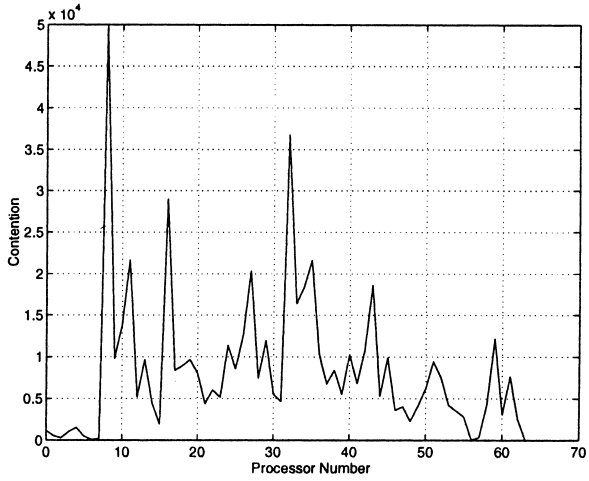


Fig. 18. 2-D mesh contention on each processor (64 nodes, $m = 2^{10}$) all-to-all broadcast, 64 nodes, $m = 2^{10}$.

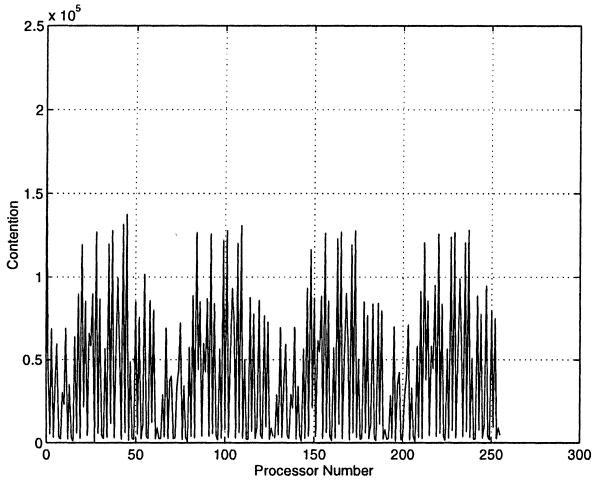


Fig. 19. 2-D PEC network contention on each processor (256 nodes, $m = 2^{14}$) all-to-all broadcast, 256 nodes, $m = 2^{14}$.

shows the total dataflow hops on 256 processors. From Figs. 25 and 27 we can see that for all-to-all broadcast PEC network encountered less dataflow than that of mesh. What is more, the relative performance difference of the two networks increases with ensemble size and problem size. It shows that the PEC network is more scalable than the mesh network in the measure of dataflow hops.

- From Fig. 26, we can see that when the number of nodes is 64, the contention on PEC is higher than the contention on mesh. The reason for this is that in PEC

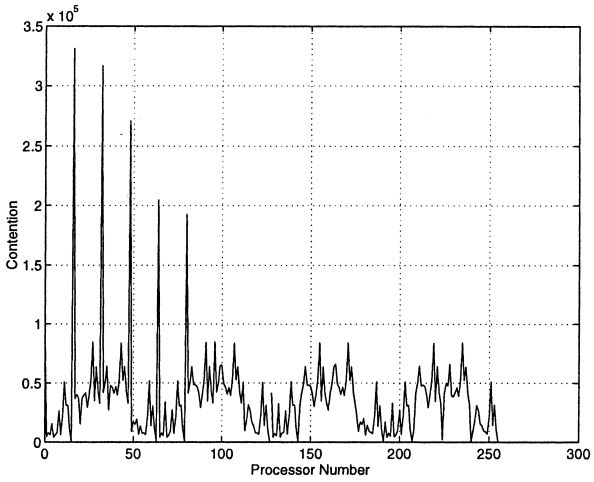


Fig. 20. 2-D mesh contention on each processor (256 nodes, $m = 2^{14}$) all-to-all broadcast, 256 nodes, $m = 2^{14}$.

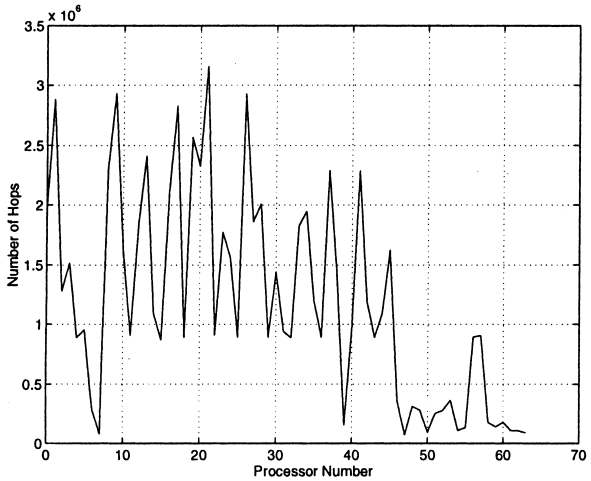


Fig. 21. 2-D PEC network dataflow hops of each processor FFT, 64 nodes, $m = 2^{18}$.

structure, all messages try to route through the highest level PEC nodes available on each X- and Y-dimension. This longest possible routing scheme generates high contention on some high level nodes of PEC. When the number of processor is 256, we find that the contention on PEC becomes lower than that of mesh. The reason for this is that, with large ensemble size, message must route through more nodes before reaching the target node on mesh; while on PEC network more high level paths become available and contention on high level links is scaled down (see

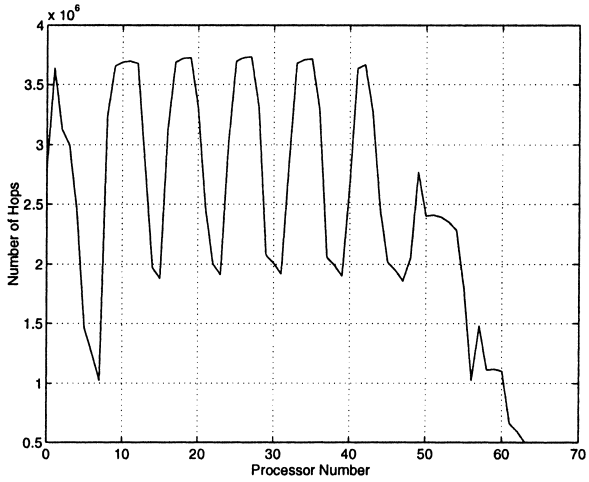


Fig. 22. 2-D mesh dataflow hops of each processor FFT, 64 nodes, $m = 2^{18}$.

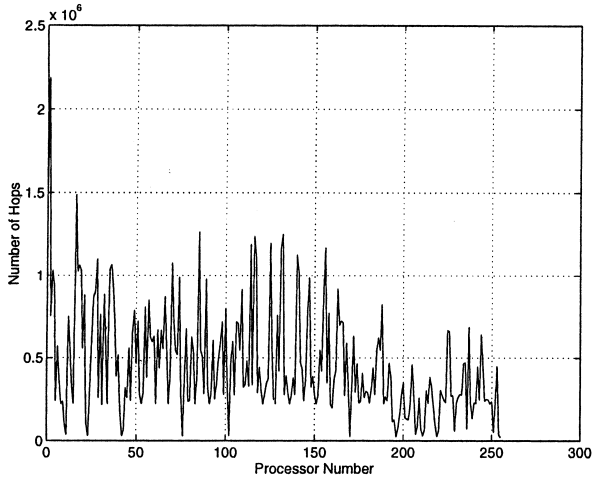


Fig. 23. 2-D PEC network dataflow hops of each processor FFT, 256 nodes, $m = 2^{18}$.

Fig. 28). The long links of PEC make it more efficient for large ensemble constructs. Contention on PEC over mesh is expected to be even better when the ensemble size scales up from 256. PEC is more scalable than mesh in the measure of contention as well.

6.2.2. *Fast fourier transform*

FFT is implemented on *PROTEUS* using *transpose* method. Through analysis of all performance figures, we have the following observations:

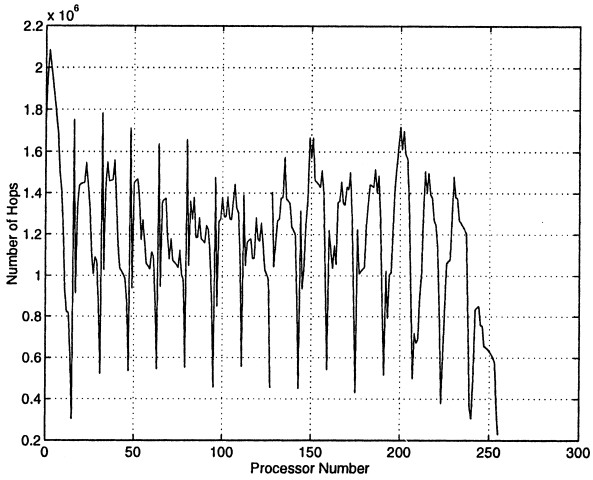


Fig. 24. 2-D mesh dataflow hops of each processor FFT, 256 nodes, $m = 2^{18}$.

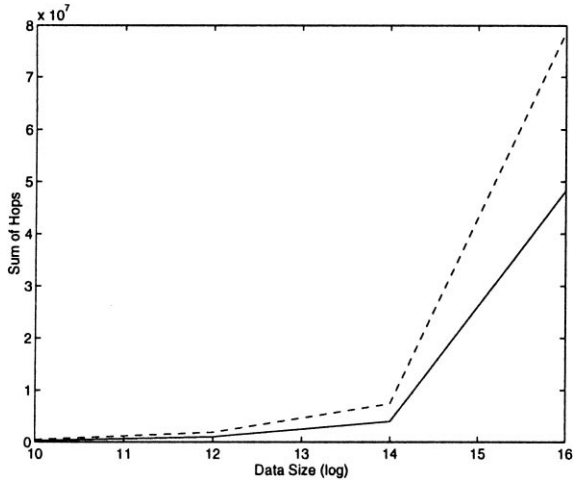


Fig. 25. Comparison of hops. Comparison of all-to-all broadcast with 64 processors.

1. The dataflow under PEC is dramatically lower than that under mesh (See Fig. 29).
2. The contention under PEC is also dramatically lower than that under mesh (See Fig. 30).
3. In PEC, the difference among dataflow hops of each processor is much smaller than that of mesh. Which means that the load balance (in terms of "intensive usage" of each node for communication) is better achieved on PEC than that of on

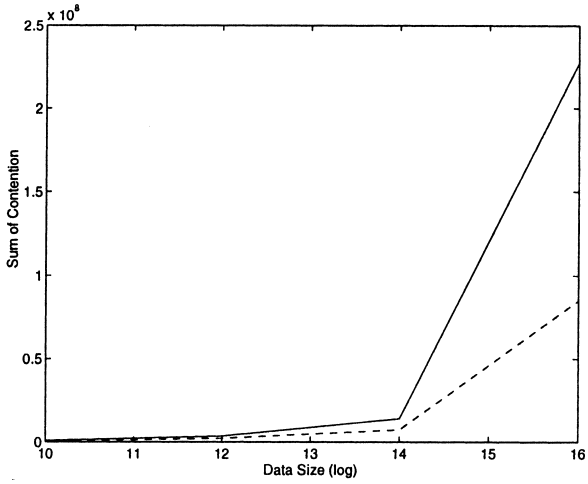


Fig. 26. Comparison of contention. Comparison of all-to-all broadcast with 64 processors.

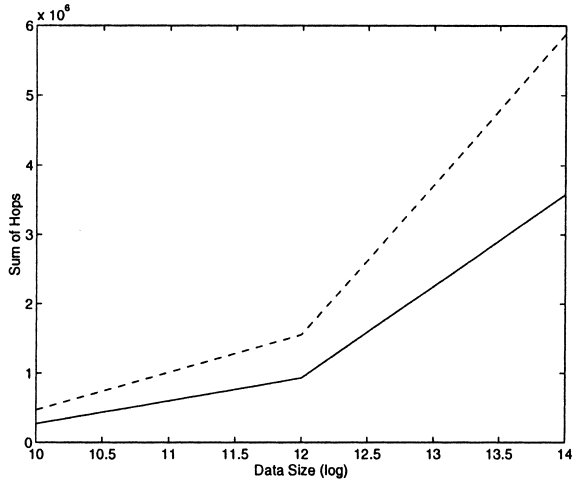


Fig. 27. Comparison of hops. Comparison of all-to-all broadcast with 256 processors.

mesh. Load balance is a very important character of interconnection network. The good load balance shows that PEC is a suitable interconnection network for FFT and many other applications.

Although communication dataflow and contention of FFT with 64 nodes are not presented, the scalability discussion of the performance of all-to-all broadcast is applicable to FFT performance as well. PEC is a more scalable network than mesh.

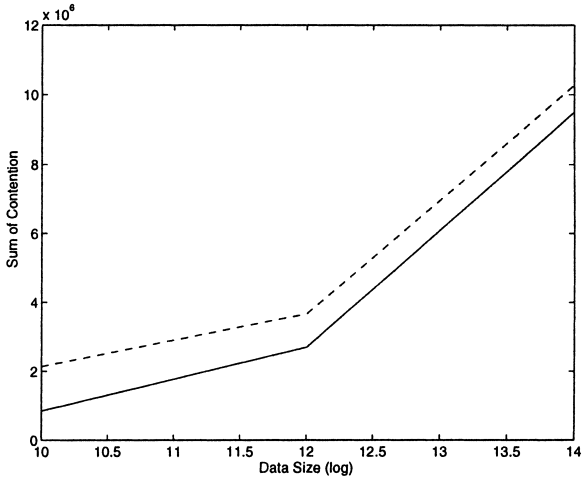


Fig. 28. Comparison of contention. Comparison of all-to-all broadcast with 256 processors.

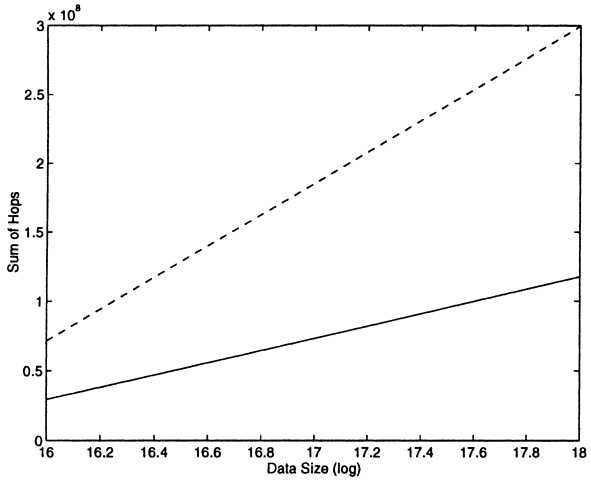


Fig. 29. Comparison of hops. Comparison of FFT with 256 processors.

6.2.3. Architecture enhancement

Combining the performance analysis of the two applications, we conclude the following findings:

- PEC network is superior than mesh, in the measure of contention and in the measure of number of messages passed per node.
- For binary-exchange scheme, the contention generated on each node of PEC is more uniform than that of on mesh, which means load balance, in terms of communication utilization, is better accomplished on PEC.

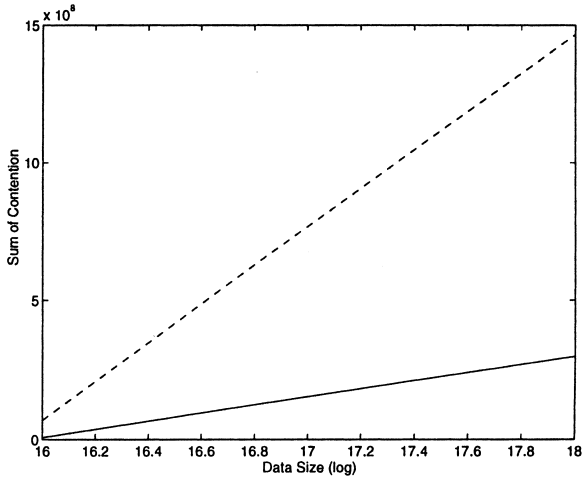


Fig. 30. Comparison of contention. Comparison of FFT with 256 processors.

- PEC is more scalable than mesh, both in the measure of contention and in the measure of dataflow per node.
- On PEC network, contention is concentrated on some high-level links along X and Y directions. Further network enhancement is possible by increasing bandwidth of these highly utilized links.

The reason of high contention on some high level links can be traced back to the routing scheme used on PEC network. To reduce traffic, 1-dimension *R-Route* is designed to use long connection (high level) links whenever possible. Therefore, the success of *R-Route* means a high utilization of long jump links when global communication is encountered. A natural approach to bail out the contention in 1-dimension PEC network is: increasing communication bandwidth with connection links' level in the network. Following the definition of fat tree, we call the PEC network with the scaled bandwidth on high level links the *fat PEC network*. On 2-D PEC network, since the routing is based on applying 1-D *R-Route* on X and Y direction separately, only links along X and Y direction contribute to the route. No horizontal or no vertical links do not participate in the routing and can be eliminated without influencing the performance. This observation suggests that links in connection level i, where 2^i is great or equal to the maximum number of nodes along X-dimension and Y-dimension of a 2-D PEC can be removed, since they do not contribute to the routing (see Fig. 8). 2-D PEC network with removed top level links is called the *truncated PEC* network. The removal of the top level connections reduces the complexity of VLSI fabrication for 2-D PEC and makes the additional links (bandwidth) for the intermediate level connections, which contribute the longest connections along each X and Y direction, possible. Combining the fat PEC concept along each X and Y direction and the truncated PEC concept on the 2-D PEC network, we get the *truncated fat PEC* network. Simulation results presented in this section show that a *truncated fat PEC* network should improve the

performance of ordinary PEC network and should provide an even better performance over the mesh topology. More simulation needs to be conducted in the future to simulate the performance of fat PEC network and truncated fat PEC networks.

7. Conclusion

PEC [5] is a relative new interconnection network for augmenting a nearest-neighbor mesh with longer distance connections. Each node of a 2-D PEC network requires a constant of eight connections, where the extra four lines connect nodes 2^h, called level h connection, distance away along the (N,W,S,E) direction of an ordinary 2-D mesh. A simulation study of 2-D PEC network is conducted in this research. Like most parallel-system simulation, this study includes multiple phases: understanding the characteristics of the network topology, developing or mastering and modifying an existing simulator, identifying algorithms and applications for benchmarking, and conducting the simulation and analysis of the simulation results. Unlike most simulation studies, since there is no routing scheme available for PEC network, a considerable effort was made in this study to develop a practical routing scheme, *R-Route*. Applications with two different global communication patterns, namely total-data-exchange and global data transpose communication pattern, are used for performance evaluation. Their performances on 2-D PEC are simulated on $\mathscr{PROTEUS}$ software environment, a widely used parallel system simulator developed at MIT. Simulation results show PEC network provides better support for communication than 2-D mesh in terms of contention and messages passed per node. In addition, PEC is more scalable and expected to deliver an even better performance over mesh when the system ensemble size and problem size increase. Truncated Fat PEC network is suggested based on our simulation findings to further improve the performance of PEC network.

The simulation results conducted in this study are subjected to the newly proposed routing scheme, *R-Route*. In contrast to a 2-D mesh, where an efficient routing scheme is readily available, routing on a 2-D PEC network is not an easy task. *R-Route* is a one dimensional routing scheme. It is not an optimal routing scheme on a 2-D PEC and, in fact, it is not an optimal routing scheme on a 1-D PEC. The development of optimal routing schemes and the searching of the lower bound of routing on a 2-D PEC are still the subject of research. With an optimal or improved routing scheme, the performance of PEC network would be further improved.

Acknowledgements

The authors are grateful to Professor David Koppelman for his help on installation and implementation of the $\mathscr{PROTEUS}$ simulation environment.

References

[1] A. Aho, J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, MA 1974.
[2] E.A. Brewer, C.N. Dellarocas, Proteus user documentation version 0.5. MIT Technical Report, MIT, Laboratory for Computer Science, Dec. 1992.
[3] E.A. Brewer, C.N. Dellarocas, A. Colbrook, W.E. Weihl, Proteus: A high-performance parallel-architectre simulator. MIT Technical Report, LCS/TR-516, MIT, Laboratory for Computer Science, 1991.
[4] K. Hang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, New York, 1993.
[5] W. Kirkman, D. Qunannman, Packed exponential connections- ahierarchy of 2-d meshes, in: Proceedings of the International Parallel Processing Symposium, 1991.
[6] D. Kopplman, Version L3.5 Proteus Changes. LSU, Aug. 1995.
[7] V. Kumar et al., Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin/Commings, Menlo Park, CA, 1994.
[8] C. Lin, V. Prasanna, Bounds on the diameter of one-dimensional per networls, J. Parallel and Distributed Computing 29 (1995) 1–16.
[9] D. Quammen, P. Wang, Bitonic sorting on 2-d pec, an algorithms study on a hierarchy of meshes netwrok, in: Proceedings of the 8th International Parallel Processing Symposium, 1994.
[10] X.-H. Sun, D. Rover, Scalability of parallel algorithm-machine combinations, IEEE Transactions on Parallel and Distributed Systems, 1994, pp. 599–613.
[11] C. Wong, C. Stokley, Q. Nguyen, D. Quammen, P. Wang, Binary-exchange algorithms on a packed exponential connected netwrok, in: Proceedings of 1995 International Conference on Parallel Processing, 1995.