Performance Evaluation of a 100-TeraOps Parallel System

A. T. Chronopoulos and Y. Gong
Wayne State University
email:chronos@cs.wayne.edu

H. Grebel and S. G. Ziavras
New Jersey Institute of Technology
Newark, New Jersey 07102

Abstract

This work deals with the performance evaluation of an MIMD distributed shared-memory parallel computer capable of achieving 100 TeraOps. This machine was the result of the one of the eight "point design " projects supported by NSF to investigate the PetaOps machines designs for the first decade of the next millennium. The machine consists of cards of eight silicon processors and their memories. The eight processors are connected via a crossbar switch. The interconnection network between the cards is based on the Generalized Hypercube (GH) topology with optical links instead of wires.

In this work, we review the salient features of the design. We then study the communication operations for this design and the efficient mapping of linear algebra operations and algorithms. Finally, we evaluate the performance of these operations. Our performance evaluation demonstrates that the machine delivers overall a high performance and thus it is a very promising future design.

Key Words: Generalized Hypercube, 100 TeraOps, Performance Evaluation, Optical Interconnection Network .

1 Introduction

High-performance computing is expected to continue providing leadership in successes in science, engineering and technology in the next millennium. If the current pace of developments in semiconductor and optical technologies is maintained it is predicted that computer systems with a sustained rate of one PetaOps (i.e. 10^{15} Operations per second) will be available in the first decade of the next century.

NSF-sponsored 'point design' studies demonstrated the feasibility of systems attaining 100 TeraOps (i.e. 10^{12} Operations per second) by the year 2005 using computer and network components rates as projected by the Semiconductor Industry Association (SIA), [4], [10], [11], [13], [16], [21], [19], [23].

Many past and current massively parallel computers are based on meshes or k-ary n-cubes (e.g. *Cray T3E, Intel Paragon, Tera* and the designs in [6], [14]). We consider the generalized hypercube GH(d,k), where k = size of a single dimension and d = dimension of GH . Figure 2.1 illustrates the 2-D GH(d,k), with $d = 2$ and $k = 4$. Unlike the mesh or k-ary n-cubes, the generalized hypercubes have all their nodes, in each dimension, fully interconnected. The diameter of a 2-D GH(d,k) is only 2 and the bisection width is $k^3/4$. The word *nodes* will refer to the GH nodes, throughout this paper. However, the number of interconnection links in each dimension increases linearly with the number of nodes k [2].

Many research projects have been devoted to parallel processing using optical components [1], [9], [15], [17]. Recent studies in optical engineering have shown that it is feasible to implement interconnections with wavelength selectivity for channel allocation [3], [18], [20], [22].

Our 'point design' is a distributed memory system based on the GH(2,k). We assume that the system has k=36 nodes in each dimension. We deal with the problem of the high number of interconnection links in each dimension of the 2-D GH by using optical in-

terconnections [23]. The nodes in each dimension are interconnected via an optical network with wavelength selectivity for each source/target node selection. Lasers are used to convert the electronic signals to light. Each node consists of eight MIMD processing elements (PEs) equipped with a cache and a local memory. The PEs of a node are interconnected via a full crossbar switch [23].

The basic communication functions of GHs have been studied in [8]. The objective of this paper is to review the salient features of our system design (see [23]) and to present a study of the mapping of basic operations and linear algebra algorithms and their performance. Although, the focus of study of the operations/algorithms is for the point design the obtained results (with minor modifications) would apply to any GH based system.

The following notations will be used throughout the paper.

- P = total number of nodes in GH
- P_n = number of PEs per node
- T_p = Total parallel time
- T_{comp} = Parallel computation time
- T_{com} = Parallel communication time
- T_{lm} = node local memory fetch/store time
- N = Dimension of vectors
- t_{op} = Computation time for a 64-bit multiplication/addition/division
- t_w = Communication time for a 64-bit word between two adjacent nodes of GH
- t_{lw} = time for local memory fetch/store of one word on a node.

The following types of mapping will be used to map vectors or matrices onto the 2-D GH: (a) For dense matrix computations, we map the matrices via 2-D block checkerboard

partitioned mapping (BCPM) [7], [12]. (b) For other vector computations except those involving grid mappings we use block striped partitioned mapping (BSPM). In this case the contiguous vector blocks are first mapped to a logical linear array, which is mapped onto the 2-D GH in row major snake-like fashion [12]. (c) For the 3-D grids, we apply 3-D BCPM (see [12] for the 2-D grids BCPM).

In section 2, we present the generalized hypercube topology and evaluate the basic communication functions and operations. In sections 3-4, we discuss the mapping onto the GH(2,k) of basic linear algebra vector/matrix operations and linear systems solvers and evaluate their performance. In section 5, we present the performance related features of the architecture. In section 6, we present the performance rates for the various operations and linear algebra algorithms.

2 Basics of communication primitives/operations on the GH

The GH topology architecture has been studied in [2], [8]. Due to its high degree of node connectivity, it offers a viable alternative to the shared memory and other distributed memory architectures. In this section, we summarize all the major communication primitives and operation primitives for the GH. These primitives are basic communication kernels used in implementing parallel computations on the GH.

Study has been conducted by using spanning tree approach for four fundamental communication primitives, under store-and-forward (SF) communication model with bidirectional all-port assumption [8]. We use the same model for communication and we ignore the startup cost in our estimated timings, for simplicity of expression.

Assuming that each source node transmits a message of size M words, then the number of communication time steps (from [8]) is summarized in Table 2.1:

Primitive	Time steps
One-to-all Broadcasting	$t_w[M+d-1]$
All-to-all Broadcasting	$t_w \frac{M(k^d-1)}{d(k-1)}$
One-to-all-Scattering	$t_w \frac{M(k^d-1)}{d(k-1)}$
All-to-all-Scattering	$t_w M k^{d-1}$

Table 2.1. The Communication time steps obtained for the GH Network.

We next discuss the following important operation primitives for the GH: (a) reduction, (b) prefix sums, (c) circular shift and (d) irregular gather/scatter. These operations are frequently used in the implementation of parallel algorithms.

We will describe the operations (a)-(d) on a vector of size N (with $N > P_n P$). We assume the vector $\vec{x}$ has been distributed to the PEs, so that each PE is assigned a subvector of size $\frac{N}{P_n P}$. Table 2.2 summarizes the complexity for each case (a)-(d), which we discuss in detail next.

Operation	T_{comp}	T_{com}	T_{lw}
Prefix-sums	$(\frac{N}{P_n P}-1)t_{op}$	$2t_w$	$2(\frac{N}{P_n P})t_{lw}$
Reduction	$(\frac{N}{P_n P}-1)t_{op}$	$2t_w$	$\frac{N}{P_n P}t_{lw}$
Circular q-shift	N/A	$\frac{2N}{P}t_w$	N/A
Irregular gather/scatter case (i)	N/A	$k\frac{N}{P}t_w$	$2Nt_{lw}$
Irregular gather/scatter case (ii)	N/A	$\frac{N(k^2-1)}{2P(k-1)}t_w$	$2Nt_{lw}$

Table 2.2. Operation primitives performance complexity.

(a) **Reduction:** We first apply the operation involved in the reduction locally on the GH node. This requires $\frac{N}{P_n P} - 1$ operations. Then we apply GH row-wise node exchange and reduction, followed by column-wise exchange and reduction.

(b) **Prefix-sums:** Initially at each GH node, the partial prefix sums of subvectors (each of size $\frac{N}{P_n P}$) are computed. The entire sum of each the subvector assigned to each GH nodes must be sent to other GH nodes to complete their prefix sums, by applying GH neighbor node exchanges row-wise then column-wise. During the node row-wise exchange, we also compute the full row sum in every node of the row. Then the full row sums are exchanged

column-wise. Finally, we add the full sums received from lower row node indices than the current nodes to the partial sum of the current node. T_{comp} is the same as in (a). Unlike (a), in (b) we must store locally on each PE $\frac{N}{P_n P}$ results.

(c) **Circular q-shift:** We must apply circular $qmod(P)$ shift to subvectors of $\vec{\mathbf{x}}$ (of size $\frac{N}{P}$) in the 2-D mesh. In the wrap-around 2-D mesh topology, circular q-shift requires all data to be shifted by $qmod\sqrt{P}$ along rows and to be shifted by $\lfloor q/\sqrt{P} \rfloor$ along columns. There is a correction shift by "1" after the row shift along the first column [12]. In the case of GH, we can combine the two column operations above in one time step because of full connectivities along column nodes. Since a row/column shift cost equals one time step, the communication cost is $\frac{2N}{P} t_w$.

(d) **Irregular gather/scatter:** This algorithm can be illustrated in the following loops:

```
do i = 1, N
  idx(i) = rand(N);
do i = 1, N
  b(idx(i)) = a(i);
```

We have several different cases depending on the distribution of idx(i) indices to GH nodes. We focus on the two distinct worst cases in terms of remote PE communication. The remaining cases fall in between these two. *Case(i)*: the indices idx(i) are uniformly distributed on the GH nodes. This setting requires one all-to-all scattering with $M = \frac{N}{P}$ from any node to the rest of the nodes, which yields $T_{com} = k\frac{N}{P} t_w$. *Case (ii)*: All idx(i) indices are mapped to only one GH node. Then this node must apply a one-to-all scattering with $M = N$ to all the nodes, thus $T_{com} = \frac{N(k^2-1)}{2(k-1)} t_w$.

3 Basic Linear Algebra Operations

In linear algebra, several basic operations are frequently used as components in the design of large scale complex parallel computing problems. We can categorize them in two groups, $vector\ operations$ and $matrix\ operations$. In the following sections, we study how to parallelize these operation on the GH.

3.1 Vector Operations

We can summarize the basic vector operation definitions in Table 3.1 and the parallel time in Table 3.2. We then explain the complexity in each case.

Operation	**Definition**
Inner product	$\vec{\mathbf{s}} \leftarrow \vec{\mathbf{x}}^{\mathbf{T}} \cdot \vec{\mathbf{y}}$
Saxpy	$\vec{\mathbf{y}} \leftarrow \mathbf{a}\vec{\mathbf{x}} + \vec{\mathbf{y}}$
Outer product	$\mathbf{A} \leftarrow \vec{\mathbf{x}} \cdot \vec{\mathbf{y}}^{\mathbf{T}}$

Table 3.1. Vector operations

Operation	T_{comp}	T_{com}	T_{lm}
Inner product	$\frac{2N}{P_n P} + 2(\sqrt{P}-1) + (P_n - 1))t_{op}$	$2t_w$	$\frac{2N}{P_n P}t_{lm}$
Saxpy	$\frac{2N}{P_n P}t_{op}$	N/A	$\frac{3N}{P_n P}t_{lm}$
Outer product	$\frac{N^2}{P_n P}t_{op}$	$\frac{N}{\sqrt{P}}t_w$	$\frac{N^2}{P_n P}t_{lm}$

Table 3.2. Vector operations performance complexity.

(a) **Inner product:** We assume that the result is needed by all PEs. The '+' and '$\times$' on $\frac{N}{P_n P}$ entries are performed by each PE. The node summation takes $P_n - 1$ operations and the summation of all nodes is a reduction.

(b) **Saxpy operation:** It does not require any communication. Scalar multiplication and vector summation are done at each local processor. The computation time is $\frac{2N}{P_n P}t_{op}$.

(c) **Outer product:** This operation actually generates a matrix. We use block checkerboard partitioning (BCP) mapping onto the 2-D GH [12]. Let subvectors of $\vec{\mathbf{x}}$/$\vec{\mathbf{y}}$ with $\frac{N}{\sqrt{P}}$ elements

each be distributed among the first row/column nodes on each node. The subvectors $\vec{\mathbf{x}}$/$\vec{\mathbf{y}}$ are broadcast simultaneously in GH node rows/columns, respectively.

3.2 Matrix Operations

We use BCP mapping onto the 2-D GH for all the dense matrix operations ((a)-(c) next). We study the (a) matrix transpose, (b) matrix $\times$ vector multiplication, and (c) matrix $\times$ matrix multiplication and (d) structured sparse matrix $\times$ vector operation ((i) block tridiagonal (BT) and (ii) Finite difference method (FDM) grid. We show the complexity for (a) - (d) in Table 3.3. and we next explain the algorithm mapping and complexity in each case.

Operation	T_{comp}	T_{com}	T_{lm}
Matrix Transpose	N/A	$\left(\frac{(N^2/P)}{2(\sqrt{P}-1)} + 2\right) t_w$	$\frac{N^2}{P_n P} t_{lw}$
Dense Matrix $\times$ Vector	$\left(\frac{2N^2}{P_n P} + \frac{N}{P_n\sqrt{P}} - 1\right) t_{op}$	$\frac{2N}{\sqrt{P}} t_w$	$\frac{2N^2}{P_n P} t_{lw}$
BT Sparse Mat. $\times$ Vect.	$\frac{13N}{P_n P} t_{op}$	$\sqrt{N} t_w$	$\frac{9N}{P_n P} t_{lw}$
FDM grid Sparse Mat. $\times$ Vect.	$\frac{13N}{P_n P} t_{op}$	$(\sqrt{\frac{N}{P}} + 1) t_w$	$\frac{9N}{P_n P}$
Matrix $\times$ Matrix	$\frac{2N^3}{P_n P} t_{op}$	$\frac{N^2}{P} t_w$	$\frac{3N^2}{P} t_{lw}$

Table 3.3. Matrix operations performance complexity.

(a) **Matrix Transpose**: We combine the BCPM and BSPM algorithms [12] in order to get an efficient matrix transposition for the 2-D GH. Figures 3.1-3.2 illustrate this technique. All nodes, except the diagonal node, have $\sqrt{P} - 1$ distinct paths along rows as well as $\sqrt{P} - 1$ paths along columns. It takes a maximum of 3 time steps for any path to reach the destination node. For instance, source node P_{12} sends parts of it submatrix to P_0, P_4,P_8,P_{13},P_{14}, and P_{15} in the first step of the three steps. All other nodes work similarly without link conflicts. Then

$$T_{com} \quad = \quad \frac{3M}{2(\sqrt{P} - 1)} t_w$$

where $M = \frac{N^2}{P}$.

(b) **Dense Matrix $\times$ Vector Multiplication**: We assume BCPM for the matrix **A** and we assume that the vector $\vec{x}$ is distributed to the top row of GH nodes and the result vector $\vec{y}$ is stored in the rightmost column of GH. The subvector of $\vec{x}$ is broadcast in GH columns. After the local multiplication, a single-node-accumulation is used along rows to form the summation. So $T_{com} = \frac{2N}{\sqrt{P}} t_w$.

The computation per node involves $\frac{N}{\sqrt{P}}$ x $\frac{N}{\sqrt{P}}$ multiplications, $\frac{N}{\sqrt{P}}$ x $\frac{N}{\sqrt{P}} - 1$ additions, and $\frac{N}{\sqrt{P}} - 1$ additions in the accumulation phase. Since each node has P_n PEs, the computation time is:

$$T_{comp} \approx \left(\frac{2N^2}{P_n P} + \frac{N}{P_n \sqrt{P}} - 1 \right) t_{op}$$

(c) **Structured sparse matrix $\times$ vector multiplication**: We discuss only important cases, which result from the discretization of partial differential equations (PDE) models on bounded 3-D domains.

(i)**Block Tridiagonal:** Let **A** be a block tridiagonal matrix. We consider the important case where the diagonal blocks are pentadiagonal matrices and the off_diagonal blocks are diagonal matrices. These matrices result from the PDEs solved numerical via FDM. We assume the dimension of each block is $N^{1/3} \times N^{1/3}$ and $P \leq N^{1/3}$. This matrix can also be viewed as a special seven diagonal matrix. We apply BSPM. Each node is assigned $\frac{N}{P}$ rows. Each node needs only $N^{2/3}$ elements of $\vec{x}$ from its linear array two immediate neighbors. Both neighbor exchanges can be done at the same time. Thus, $T_{com} = N^{2/3} t_w$. There are seven non-zero elements per row and so six additions and seven multiplications are performed per row. Therefore, the computation time is $\frac{13N}{P_n P} t_{op}$.

(ii)**FDM grids:** We map PDE domain of $N^{1/3} \times N^{1/3} \times N^{1/3}$ grid points with regular grid point ordering to the 2-D GH nodes (see [12] for the 2-D FDM grid case). There are N grid points in total and each node is assigned $\frac{N}{P}$ grid points. Each node need only exchange $6\frac{N}{P}^{1/3}$ boundary points with all six adjacent nodes. All data exchanges on six boundaries

can occur simultaneously. Thus we have:

$$T_{com} = (\frac{N}{P}^{1/3} + 1)t_w \quad (1)$$

The computation time for finite difference grids is the same as in (a).

(d) **Matrix $\times$ Matrix Multiplication**: We consider matrices $\mathbf{A}$/$\mathbf{B}$ mapping via the BCP onto the 2-D GH . We use Cannon's algorithm [12]. An initial alignment of submatrices $\mathbf{A}$ and $\mathbf{B}$ is applied, which takes time = $\frac{N^2}{P}t_w$. After the initial alignment, one block submatrix multiplication is performed. Then $\sqrt{P}-1$ $\mathbf{A}$/$\mathbf{B}$ submatrix shifts are applied each one being followed by one block submatrix multiplication. Each submatrix takes time = $\frac{N^2}{P}t_w$. Thus:

$$T_{com} = \sqrt{P}(\frac{N^2}{P})t_w = \frac{N^2}{\sqrt{P}}t_w$$

4 Systems of linear equations

In this section, we discuss the Gaussian elimination, the Jacobi relaxation and the conjugate gradient algorithms.

4.1 Gaussian Elimination

We use the cyclic BCPM, see Figure 4.1. Our goal is to minimize the nodes idling. From Figure 4.1, we can observe that the amount of communication does not remain constant for all iterations. The communication consists of broadcasting row A[k,j] along GH node-columns and column A[i,k] along GH node-rows, in the GH node handling the part of matrix where Gaussian elimination is still active. Each such node must transmit a subvector of row entries followed by a subvector of column entries. For the first $\sqrt{P}$ iterations, at most $\frac{N}{\sqrt{P}}$ elements are broadcast row-wise/column-wise. For the next $\sqrt{P}$ iterations, at most $\frac{N}{\sqrt{P}} - 1$ entries are broadcast; and so on. So, we have: communication time is:

$$T_{com} \leq 2\sqrt{P}\left((\frac{N}{\sqrt{P}}) + (\frac{N}{\sqrt{P}} - 1) + \cdots + 1\right)t_w$$

$$= 2t_w\sqrt{P}\sum_{i=0}^{\frac{N}{\sqrt{P}}-1}(\frac{N}{\sqrt{P}}-i)$$
$$= (\frac{N^2}{\sqrt{P}}+N)t_w \quad (2)$$

The additions and multiplications involved (for a full submatrix assigned to a GH node) are in the order of $(\frac{N}{\sqrt{P}})^2$. Also, for every $\sqrt{P}$ iterations, the dimension of this submatrix is decreased by one, so we have:

$$T_{comp} \leq \frac{2\sqrt{P}}{P_n}\left[\left(\frac{N}{\sqrt{P}}\right)^2+\left(\frac{N}{\sqrt{P}}-1\right)^2+\cdots+1\right]t_{op}$$
$$= \frac{2t_{op}\sqrt{P}}{P_n}\sum_{i=0}^{\frac{N}{\sqrt{P}}-1}\left(\frac{N}{\sqrt{P}}-i\right)^2$$
$$= \frac{t_{op}}{3P_nP}(2N^3+3N^2\sqrt{P}+NP) \quad (3)$$

For the backsubstitution, only the right most column of $\sqrt{P}$ GH nodes perform the computation. (a) All the entries of a matrix row required for this operation are sent in GH rows to the last GH column, which costs $\frac{N^2}{P}$ time steps. (b)The nodes involved in the computation send the computed results column-wise to all unfinished nodes. Due to the cyclic block checkerboard mapping, this takes (N-1) time steps. steps. So we have:

$$T_{com} = (\frac{N^2}{P}+N-1)t_w \quad (4)$$

For the Cholesky algorithm, the communication cost is of the same order as the Gaussian elimination, but only half of the matrix need be stored.

4.2 Jacobi Relaxation

The 3-D grid Jacobi relaxation main computation is:

3-D Jacobi:

do m=1, nc /* number of species */

do k=1, nz

do j=1,ny

do i=1,nx

do mp=1,nc /* number of Jacobi iterations */

$$b(i,j,k,m) = u(mp,m)(b(i+1,j,k,mp) + b(i-1,j,k,mp) + b(i,j+1,k,mp) +$$

$$b(i+j-1,k,mp) + b(i,j,k+1,mp) + b(i,j,k-1,mp)) + a(i,j,k,m)$$

Enddo

The *k, j* and *i* loops can be viewed as a three dimensional cube of total size $nx \times ny \times nz$. This cube can be evenly divided into P subcubes, each of size (size $nx \times ny \times nz)/P$, and be distributed among all GH nodes. The communication cost of this algorithm only occurs at the six boundary faces of each subcube. All other elements can be obtained locally. All the $nc \times nc$ elements of $u(mp, m)$ remain constant for the *k,j* and *i* indices, so they can be stored in the node's local memory. Both a and b need to be exchanged between adjacent nodes at the boundaries. Communication is required only for mp loops. Therefore, the overall communication for the algorithm is:

$$T_{com} \quad = \quad nc \times n_b t_w \ , \tag{5}$$

where n_b is the total number of grid points on the six boundary faces of a node. The computation takes in $(5 \times nc + nc^2) \times n_g$ additions and $nc^2 \times n_g$ multiplications, where n_g is the total number of grid points for a subcube. So the total computation is:

$$T_{comp} \quad = \quad (6 \times nc + nc^2) \times n_g \ t_c \tag{6}$$

The first term of the computation refers summation of the parenthesized b-terms for the mp-indices, which can be computed at $mp = 1$ and stored in the node memory for reuse for $mp > 1$.

4.3 Conjugate Gradient

We discuss the Conjugate Gradient applied to a block tridiagonal linear system. The communication and computation costs are calculates from the costs of its basic operations (saxpy, inner product, matrix-vector multiplication). All we need is to combine the cost of the various operations. For the truncated Incomplete Cholesky (IC) preconditioner, z_k is computed by solving $Mz_k = r_k$ which can be approximated by solving:

$$(1)\ u \approx \tilde{L} r_k \quad (2)\ v \approx \tilde{D}^{-1} u \quad (3)\ z_k \approx \tilde{L}^T v\ ,$$

where $\tilde{D}$ is a diagonal matrix and $\tilde{L}$ is lower triangular matrix. We use truncated power series approximation and we truncate after 2 series terms [12]. The parallel time per iteration can be expressed as:

T_p = (a) Matrix-vector multiply for $Mz_k = r_k$ for z_k

(b) + communication for $Mz_k = r_k$

(c) + time for matrix-vector multiply

(d) + time for inner product

(e) + time for saxpy

$= 23 t_{op} \frac{N}{P_n P}$ (For (a) $\Longleftarrow$ 11 $t_{op} \frac{N}{P_n P}$ for each of (1)&(3) and $t_{op} \frac{N}{P_n P}$ for (2))

$+ (5\sqrt{\frac{N}{P}}) t_w + \frac{13N}{P_n P} t_{op}$ (For (b) $\Longleftarrow$ 4 neighbors exchange)

$+ (5\sqrt{\frac{N}{P}}) t_w + \frac{13N}{P_n P} t_{op}$ (for (c) $\Longleftarrow$ 4 neighbors exchange)

$+ \frac{6N}{P_n P} t_{op} + 2t_w$ (For (d) $\Longleftarrow$ 3 inner product)

$+ (\frac{6N^2}{P_n P}) t_{op}$ (For (e) $\Longleftarrow$ 3 saxpy operations)

5 Architectural Features of the 100 TeraOps Parallel Computer

Fig 5.1, shows the logical GH(d,k) network for $d = 2$, $k = 36$ and $P = 36 \times 36$. By using optical channels with laser technology, a physical hardware architecture of 2-D GH results [23]. We consider the key delays communication/computation delays involved

with this new architecture. We assume that each node is a card of of P_n = 8 PEs, each PE has a 16MB(ytes) cache and its own local memory of size= 32 GB(ytes), and the PEs and their memories are interconnected (in the card) via a crossbar switch.

For communication between processor and memory on the same node, shown in Figure 5.2, the time and rate taken is:

$$\begin{aligned} Time &= \frac{1}{450MHz} + \frac{1}{2000MHz} + \frac{1}{2000MHz} + \frac{1}{2000MHz} + \frac{1}{375MHz} = 0.01089(\mu s) \\ Rate &= \frac{1}{.01089 \times 10^{-6}} \times \frac{128bits}{8bits/byte} = 1.469G(iga)B(yte)/sec \end{aligned}$$

For communication between processors and memory on the different nodes, shown in Figure 5.3, we have:

$$\begin{aligned} Time &= t_{pe\_to\_remote} = \frac{1}{450MHz} + \frac{2}{2000MHz} + \frac{1}{375MHz} + \frac{1}{375MHz} \\ +\frac{2}{2000MHz} + \frac{1}{375MHz} &= 0.01489(\mu s) \\ Rate &= \frac{1}{.01489x10^{-6}} \times \frac{128bits}{8bits/byte} = 1.0746GB/sec \end{aligned}$$

If we use DMA transfer between memory to other nodes, we have:

$$\begin{aligned} Time &= t_{DMA} = \frac{1}{375MHz} = 2.66ns \\ Rate &= 375MHz \times \frac{128bits}{8bits/byte} = 6GB/sec \end{aligned} \tag{7}$$

Let *fetch/store* refer to node memory access throughout rest of the paper. We note that $t_w = t_{DMA}$ for fetch/store of contiguous vectors of elements and $t_w = Tpe\_to\_remote$, otherwise.

The design complies with the following time delays, which are compatible with the SIA projections (see [19]).

- t_d = The memory clock-cycle time, which is $\frac{1}{500MHz}$ = 2 ns

- t_p = Number of floating point units (FPUs) in a PE = 10
- t_f = Number of fetches/stores of ten 8 byte words from/to memory per cycle = 10
- $t_c = \frac{1}{1GHz} = 1ns$

Since the PE functional units are assumed to be pipelined, the peak performance of the system is $36^2 \times 8 \times 10$ TeraOps = 103.6 TeraOps. Table 5.1 shows the local/remote communication rates. The CPU/local memory equals $\frac{t_f}{t_d} \times 8bytes$. The bisection bandwidth equals $18 \times 18 \times 36 \times 6GB/sec$ in each direction of the 2 half of the GH.

Type of Data Transfer	**Bandwidth**
CPU/local-memory	40 GBs
CPU/remote memory on the same card	1.469 GBs
CPU/remote memory on another card	1.075 GBs
Memory/memory via DMA	6 GBs
Bisection bandwidth	69.9 TeraBs

Table 5.1. The Communication time steps obtained for point design network.

6 Numerical Performance Analysis

We now provide performance evaluation characteristics for the vector/matrix operations and algorithms of the preceding section. We consider two values for the dimension : (i) $N = 10^5$ for the vectors and matrices involved in dense matrix operations/algorithms, (ii) $N = 10^9$, for all other cases. We assume that the numbers are 8-Byte floating point numbers. We will demonstrate only the evaluation of *inner product*, *matrix transpose* and *Jacobi relaxation*. We then provide a table summarizing the run-times and expected performance for the remaining cases. We assume that efficient cache utilization is achieved by using the BLAS-2, 3 formulations of the matrix algorithms (see [7], [5] and the references therein).

(a)**Inner Product:** There are $\frac{N}{P_n P} = 96451$ local memory fetches for entries of each $\vec{x}$ and $\vec{y}$. These fetches can be overlapped with computation on the local nodes. We ignore the

startup time. The estimated execution time is:

$$\begin{aligned} T &= \left[\frac{2N}{P_n P} + 2(\sqrt{P}-1) + (P_n - 1)\right]\frac{t_d}{t_f} + 2P_{pe\_to\_remote} \\ &= [2 \times 96451 + 2x35 + (8-1)] \times \frac{2 \times 10^{-3}}{10} + 2 \times 0.01489 \\ &= 38.58 + .0298 = 38.61(\mu s) \end{aligned}$$

The estimated execution rate is $\frac{2\times 10^9 Ops}{3.903\mu s} = 51.7$ TeraOps.

(b)**Matrix Transpose**: We can use DMA transfer in this operation. The estimated execution time is:

$$T = \frac{3M}{2(\sqrt{P}-1)}t_{DMA} + \frac{N^2}{P_n P}\frac{t_d}{t_f} = 879.6 + 192.9 = 1073(\mu s)$$

The estimated execution rate is $\frac{10^5 \times 10^5 elements \times 8 Bytes/elements}{1073\mu s} = 74.6$ TeraBs

(c)**Jacobi Relaxation**: Let nx=nx=nz=1000 and nc=5. We map the the Jacobi relaxation grid cube onto the 36×36 GH nodes (see Figure 6.1) by configuring the GH nodes in a $9 \times 9 \times 16$ virtual cube. Then each node is assigned $\frac{1000}{9} \times \frac{1000}{9} \times \frac{1000}{16} = 771604$ grid points. This will result in 4 subcube faces with $\frac{1000}{9} \times \frac{1000}{16}$ elements and 2 subcube faces with $\frac{1000}{9} \times \frac{1000}{9}$ elements. The total number of gridpoints at six boundary faces for each node end up to be 24720. The number of grids points per PE is n_g = 771604/8 = 96451. The number of communicated elements per PE is 24720/8 = 3090. if we overlap local fetches with computation, the total estimated execution time is:

$$\begin{aligned} T &= (nc \times 6 + nc \times nc) \times 96451\frac{t_c}{t_p} + 3090 \times t_{DMA} \\ &= 385.8 + 45.01 = 430.8(\mu s) \end{aligned}$$

The estimated execution rate is $\frac{55\times 1000^3 Ops}{430.8\times 10^{-6} s} = 92.8$ TeraOps.

Table 6.1 summarizes the performance analysis results of all algorithms we discuss in previous sections. Note for the items with (*) $N = 10^5$ and for the rest $N = 10^9$.

Problem	Execution Time	Execution Rate
Perfix-sums	1.958 μs	51.2 TeraOps
Reduction	0.994 μs	51.2 TeraOps
Circular q-shift	410.5μs	1.95 TeraBs
Irregular Scatter (A)	205 μs	3.89 TeraBs
Irregular Scatter (B)	3796 μs	0.21 TeraBs
Inner Product	3.90 μs	51.2 TeraOps
Saxpy	3.86 μs	34 TeraOps
Outer Product*	$200.3\mu s$	99.8 TeraOps
Matrix Transpose*	$486\mu s$	74.6 TeraBs
Dense Matrix $\times$ Vector	$400.8\mu s$	49.9 TeraOps
B.T. Matrix $\times$ Vector	43.97 μs	24 TeraOps
FDM Grid Matrix $\times$ Vector	18.1 μs	37 TeraOps
Matrix $\times$ Matrix*	$2.004 \times 10^7 \mu s$	99.8 TeraOps
Gaussian Elimination*	$6.5 \times 10^{12} \mu s$	98 TeraOps
Jacobi Relaxation	430.8 μs	92.8 TeraOps
Conjugate Gradient	$1.76 \times 10^{16} \mu s$	38.2 TeraOps

Table 6.1. Timings/performance results of various GH algorithms.

7 Conclusions

We have made a study of the efficient mapping of many widely used communication and computation kernels and linear algebra algorithms onto a 'point design' parallel computer. This parallel computer is based on the GH topology with its multiple port links in each dimension being implemented by optical components. We then presented a performance evaluation of the point design based on a very important set of kernels and linear algebra algorithms. We note that the single Saxpy and inner product do not reach the peak rate sustained by the design. This is not negative because dense and structured sparse linear algebra algorithms can be expressed in terms of matrix times vector or matrix times matrix operations (see [7], [5] and the references therein), which run near the peak rate. Our results show that this design can lead to viable future PetaOp computer.

Acknowledgement: This work was supported in part by NSF Grant ASC-9634775.

References

[1] G. A. Betzos and P. A. Mitkas, *Perf ormance Evaluation of Massively Parallel Processing Architectures with 3D Optical Interconnections*, Applied Optics, 1998, pp. 315-325.

[2] L.N. Bhuyan and D.P. Agrawal, *Generalized Hypercube and Hyperbus Structures for a Computer Network*, IEEE Trans. Comput., Vol. 33, No. 4, 1984, pp. 323-333.

[3] L. Camp, R. Sharma, and M. Feldman, *Guided-Wave and Free Space Optical Interconnect for Parallel Processing Systems: A Comparison*, Applied Optics, Vol. 33, 1994, pp. 6168-6180.

[4] A. A. Chien and R. K. Gupta, *MORPH: A System Architecture for Robust High Performance Using Customization* , 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[5] A. T. Chronopoulos and C. D. Swanson, *Parallel Iterative S-step Methods for Unsymmetric Linear Systems*, Parallel Computing. Volume 22/5, 1996, pp. 623-641.

[6] W.J. Dally, et. al., *The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms*, IEEE Micro, Vol. 12, Apr. 1992, pp. 23-39.

[7] J. J. Dongarra and D. W. Walker, *Software Libraries for Linear Algebra Computations on High Performance Computers* SIAM Review, Vol. 37, No. 2, 1995.

[8] P. Fragopoulou, S. G. Akl and H. Meijer, *Optimal communication Primitives on the Generalized Hypercube Network* Journal of parallel and distributed computing, 32, 173-187, 1996.

[9] E. Frietman, W. van Nifterick, L. Dekker, and T. Jongeling, *Parallel Optical Interconnects: Implementation of Optoelectronics in Multiprocessors Architecture*, Applied Optics, Vol. 29, 1990, pp. 1161-1167.

[10] G. Gao, K. Likharev, P. Messina and T. Sterling , *Hybrid Technology Multithreaded Architecture*, 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[11] P. M. Kogge, S.C. Bass, J.B. Brockman, D.Z. Chen, abd E. Sha, *Pursuing a Petaflop: Point Designs for 100 TF Computers Unsing PIM Technologies*, 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[12] V. Kumar, A. Grama, A. Gupta and G. Karypis, Kumar, V. et al., *Introduction to Parallel Computing Design and Analysis of Algorithms*, The BenjaminCummings Publishing Company, Inc. 1994.

[13] V. Kumar, A. Sameh, A. Grama and G. Karypis, *Architecture, Algorithms and Applications for Future Generation Supercomputers* , 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[14] D. Lenoski, J. Laudon, K. Gharachorloo, W.D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, *The Stanford FLASH Multiprocessor,* IEEE Computer, March 1992, pp. 63-79.

[15] A. Louri and H. Sung, *3D Optical Interconnects for High-Speed Interchip and Interboard Communications,* IEEE Computer, Oct. 1994, pp. 27-37.

[16] Z. B. Miled, R. Eigenmann, J. Fortes and V. Taylor, *Hierarchical Processors-and-Memory Architecture for High Performance Computing*, 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[17] V. Morozov et al., *Analysis of a Three-Dimensional Computer Optical Scheme Based on Bidirectional Free-Space Optical Interconnects*, Optical Engineering, 1995, pp. 523-534.

[18] I.R. Redmond and E. Schenfeld, *A Free-Space Optical Interconnection Network for Massively Parallel Processing,* IEEE Las. Electr. Opt. Soc. Newslett., June 1995, pp. 10-13.

[19] T. Sterling, P. Messina and P. H. Smith, *Enabling Technologies for Petaflops Computing,* The MIT Press, 1995.

[20] T. Szymanski, *A Fiber Optic Hypermesh for SIMD/MIMD Machines,* Proceed. of Supercomputing ’90 Conf., Nov. 1990, pp. 103-110.

[21] J. Torrellas and D. Padua, *The Illinois Aggressive Coma Multiprocessor Project (I-ACOMA),* 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.

[22] S.C. Tsay and H. Grebel, *Design of TransverseHolographic Optical Interconnect,* Applied Optics, Vol. 33, 1994, pp. 6747-6752.

[23] S.G. Ziavras, H. Grebel, and A.T. Chronopoulos, *A Low-Complexity Parallel System for Gracious, Scalable Performance. Case Study for Near PetaFLOPS Computing,* 6th Symp. Frontiers Massively Parallel Computation, Special Session on NSF/DARPA New Millennium Computing Point Designs, 1996, pp. 363-370.