BDG-Torus Union Graph - An Efficient Algorithmically Specialized Parallel Interconnect[*]

Neeraj Suri, Avi Mendelson[†] and D.K. Pradhan

Department of Electrical and Computer Engineering
Univ. of Massachusetts, Amherst, MA - 01003

Abstract

The binary deBruijn interconnect graph (BDG) is a realizable alternative to the hypercube. A primary limitation of the deBruijn structure is, though, its inability to embed a mesh or a mesh of trees in it efficiently, compared to the hypercube. Product Shuffle (PS) graphs have been proposed, to alleviate such limitations, though at the cost of a complex interconnect structure and an increased node-degree, from 4 in a BDG, to 8 in the PS graph. A simple extension of the BDG by the edge set union with a torus is shown; this provides for the missing mesh topology and achieves graph capabilities/versatility comparable to the PS graph and the hypercube within a fixed degree graph. The structure improves upon both the PS and hypercube in implementing pipelined and multi-phase algorithms. More importantly, the purpose is designing an algorithmically specialized interconnect, by characterizing algorithmic features of a wide range of algorithms as well as direct architectural support for them, instead of simply providing for a set of graph embeddings in the interconnect. A set of examples demonstrate the Union-Graph's versatility in this aspect of algorithmic support.

1 Introduction

Designing efficient algorithmically specialized architectures is the focus of much current research. The central idea is relating the structure of a range of algorithms to the targeted interconnect structure. Currently, the hypercube and its derivatives are the more popular and powerful interconnection schemes for parallel processing. Its versatility in supporting a variety of secondary graph structures – arrays, rings, spanning trees, meshes – and its ability to support a large number of algorithms, have been demonstrated extensively in literature. Its logarithmically increasing node-degree versus the size of the network precludes feasible VLSI implementation of large networks.

A number of VLSI realizable architectures, CCC, Hypertree, Shuffle-Exchange, Hypernets, Star(Cayley Graphs), Mesh of Trees [5],[6],[10],[16] and [19], among others, have been proposed as alternatives to the hypercube, their graph properties elucidated in the context of the graph embeddings feasible in these structures.

The binary deBruijn graph(BDG) first proposed as a VLSI interconnection network in [13] has been shown [1],[17],[18] to be a powerful hypercube competitor and also a VLSI implementable interconnect graph. It possesses optimal embeddings of rings, CBT's, tree-machines and shuffle exchange to support a majority of existing algorithm classes in a fixed degree graph [18]. The deBruijn graph has seen added importance since its implementation in the Galileo project [14], [18].

The torus(mesh) is another versatile and VLSI-viable computational structure used for many parallel graph and numerical algorithms such as Finite Element Analysis, Image Processing, Laplace equation solving and Matrix based operations. It has been shown in [17] that the mesh topology cannot be efficiently embedded in the deBruijn graph in a work preserving manner, a primary BDG limitation.

The binary n^2-node hypercube, in contrast, contains a direct embedding of an n x n mesh, improving on the basic mesh structure by virtue of the fact that each row and column of the embedded mesh is also a n node hypercube - this is apparent, as the node numbering in the obtained mesh is the same as the numbering of a Karnaugh map. Most of the mesh-based algorithms running on the cube use this to reduce the communication times or provide certain data movements as required in the algorithm, without having to incorporate the $O(n)$ communication restriction inherent in a mesh. Consequently, a row(column) in the embedded mesh can be used as an array, ring or a spanning tree per the requirements of an algorithm step. Examples of such usage are discussed in [7] and [12] using cyclic shifts, maximum/minimum element finding, data broadcasting or data aggregation on data elements in the row(column) of the embedded mesh.

Extending the basic deBruijn interconnect structure, product shuffle (PS) graphs [17] have been proposed. These are in fact a Cartesian product of deBruijn graphs designed to yield a structure combining the properties of the shuffle, meshes and mesh of trees of moderate sizes. The cost is a complex graph structure and an increased node-degree of 8.

[*]Research supported in part by NSF MIP-88-05586 and AFOSR 88-0205

[†]currently at Dept. of EE, Technion, Israel

This paper describes a VLSI-realizable Union-Graph (UG), linking a BDG graph and a Torus. The missing mesh computational structure in the BDG is provided and essentially, the entire varied computational power of a hypercube is matched (and improved upon) within a fixed degree graph. Additionally, the nature of the graph supports pipelined algorithms and algorithms requiring multiple concurrent embeddings, not possible in the hypercube. The design *"is not"* intended to propose yet another interconnect in terms of specifying its graph properties and the graph embeddings it supports. Rather, the intent is to obtain a different algorithmically specialized parallel interconnect - where some characteristic communication patterns are identified as the building blocks of most algorithms and the design of the desired Union-Graph (UG) is directed to support these primitives.

1.1 Motivation

Several authors [2],[3],[8], have emphasized that the key factor in the performance of an algorithm on a parallel interconnect is invariably the communication cost of positioning data onto the requisite processors, and not the actual computational time taken. This is termed as the algorithm's communication stage.

Most algorithms, though, do not have a single explicit communication pattern; rarely does there exists a direct mapping of the entire data flow or computational graph of an entire algorithm onto a single physical or logical topology. Usually, a general algorithm is usually composed of a number of computation and communication stages. Formally, consider an algorithm as an aggregation of stages (sub-problems) $s_0, s_1 \ldots s_m$. For stages $s_j,\ s_{j+1}, ..s_m$ requiring different topological structures, the need is to provide for :

- efficient operation of each individual stage (s_j), $\forall j$,
- and for the efficient transitional data movements as in $s_j \rightarrow s_{j+1}$ (or similar permutations) such that the necessary data is available at the appropriate processor at the time of the next stage's computation.

The first aspect requires the availability of requisite communication structures (tree, mesh,..) embedded in the overall graph, as needed by a communication stage. The second feature provides for efficient re-configuration, from the graph embedding required for s_j to the embedding required for s_{j+1}. This step usually constitutes the major cost of implementing an algorithm on an interconnect. Thus, simply describing the availability of a set of embeddings in an interconnect graph does not necessarily guarantee its effectiveness in supporting algorithms. The overall performance and efficiency of executing an algorithm on any architecture depends importantly on both aspects.

As an illustration, consider an algorithm which requires in stage (a) a nearest neighbor exchange and then in stage (b) a data summation or a min-max operation on the row elements of the mesh. In an interconnect, the embeddings desired would be : mesh for (a) and a tree for (b). For such algorithms, the prevalent technique for demonstrating the algorithmic suitability and flexibility of an interconnect has been to show a number of graph embeddings to be feasible in it. Usually, these results are of pure theoretical value. The overheads for message routing and reconfiguring inter-node links makes such reconfigurations unrealistic for real time operations. For this example, one would pick a target architecture, with an embedding of a mesh and a tree in it, to be flexible enough to support such an algorithm efficiently. These embeddings are individually useful, but their usefulness is apparent, only if one can move the data at the end of stage (a) from the mesh into the proper positions of the tree for stage (b) efficiently; if each row of the mesh was a tree, then the cost of mesh to tree data movement is small else the whole purpose of providing for these embeddings is lost.

What determines the overall efficiency of an algorithm are the basic communication patterns which need to be supported efficiently (independently) by the interconnect and their efficient inter-pattern data movements. Interestingly, most algorithms are formulated from a fairly standard set of operations and associated characteristic algorithmic and communication patterns, identifiable as:[1]

1) Single/Multiple Source, Data Broadcasting for I/O or Computations

2) Single/Multiple Source, Data Assimilation or Summation

3) Data Shifting/Data Permutations

4) Minimum/Maximum/Sorting Operations

5) Prefix Sum and Product Calculations

6) Nearest Neighbor Exchanges

7) Computation Partitioning/Divide and Conquer

The design procedure to formulate a flexible algorithmically specialized interconnect becomes (a) to provide for the required embeddings to support these patterns individually and (b) to support efficient data movement amongst such obtained embeddings. The effectiveness of this approach is shown through an example BDG-Torus Union-Graph.

A comprehensive study of this aspect would characterize all possible communication patterns and obtaining algorithms as permutations of these patterns. Further, depending upon the different topologies used for implementing these individual patterns, this leads to a new means of relating parallel algorithms to parallel architectures, detailed in [20].

2 BDG-Torus Union-Graph

Here a simple linkage is suggested between the BDG and the Torus graphs. In this way each graph can take advantage of the structures it can handle best without suffering (algorithmically) from the structures missing individually in each. The BDG discretely provides for most of the basic communication patterns: [(1) through

[1] This is the set of the most extensively used communication patterns in parallel algorithms, by no means, a complete set of all such patterns

(6)], because of the possible real-time reconfiguration as well as the efficient data-movements for the embeddings of CBT's, Shuffle, Tree Machines and Cycles in it. The Torus can be added to the BDG either by appending additional links to the existing BDG graph structure or by using an independent Torus (extra nodes), creating the links to associate a mesh interconnect with the BDG. Folding the full length Hamiltonian path in a snake-like manner to obtain a partial mesh in the BDG and adding extra links to provide for the full mesh structure is a simple solution for the first type of extension. This method leads to embedding an N node mesh in an N node BDG by increasing the node-degree to 6. Our interest, though, is in integrating the mesh with the BDG such that it provides useful support in data movements for the algorithms to be executed on the interconnect. Therefore the second method.

Many ways exist in which the BDG and the Torus graphs can be connected. Some useful linkage possibilities are :

Direct connection : The simplest way is to add links between nodes which have the same "original" node number. So, node $xxxx$ of the torus becomes $0xxxx$ and the node $xxxx$ of the BDG becomes $1xxxx$ and a link is added between them.

Shuffle connection A node on the Torus is connected to the "shuffled" node on the BDG; i.e., node number xyz in the torus is connected with the shuffled node number yzx on the BDG.

ϵ : Specific linkage between the embeddings of the BDG and the torus. The ϵ connection provides us an useful algorithmic connection; i.e., the connection can be used by the algorithm to improve its performance on the interconnect.

In this paper, we propose a specific ϵ mapping to admit the rows and columns of the mesh to be usable as 1D, 2D arrays, rings and as spanning trees in conjunction with the BDG, all using a single architecture.

The graph structure is defined as the augmented union of the edge sets of the BDG denoted as E(BDG) and the edge set of the torus as E(T) – with a mapping ϵ relating the $V(BDG) \xrightarrow{\epsilon} V(T)$ as an one-to-many onto mapping. This mapping also describes the additional links between the graphs (all the links are bidirectional). Let $N_{bdg} = N_{torus} = n^2 = 2^m$, for some n and m, describe the BDG and torus nodes respectively. Let a node of a N_{bdg}-node BDG be represented as $(a_{m-1}a_{m-2}...a_1a_0)$, with its neighbor set as :

$$\{\ (a_{m-2}\dots a_0\lambda),\ (\lambda a_{m-1}\dots a_1)\ \}$$

for $\forall\lambda \in \{0, 1\}$, see Fig. 1. In the N_{torus}-node torus, let a node be represented as a pair of co-ordinates $(i,\ j)$, $i, j \in \{0, \cdots, n-1\}$. Each node (i, j) has four neighbor nodes, i.e., $node(i, j-1)$, $node(i, j+1)$, $node(i-1, j)$, $node(i+1, j)$, where the additions and subtractions are $mod\ n$.

The N node Union-Graph has, $N = N_{bdg} + N_{torus}$ nodes. For simplicity, we restrict the mapping below for $N_{bdg} = N_{torus} = n^2$ and hence $N = 2n^2$. However, in general the Union Graph structure requires $2x$ nodes for a x-node BDG. The case where $x \neq n^2$ can also be formulated similarly and is given at the end of the section.

We start providing for the algorithm stages (1) through (6) of Section 1.1 by mapping the two CBT's present in the BDG onto the toroidal structure. This provides the basis for all of the other algorithm stages.

It has been shown in [18] that a $N_{bdg} = n^2$- node BDG has two link-disjoint (n^2-1)-node complete binary trees[2], or rather two n^2 node full trees (FT's)[3]. In each FT, there are 'n' nodes at *tree level* $(log\ n+1)$. Further, each of these 'n' nodes is also the root of a $(n-1)$ node CBT - Fig. 2. The nodes of levels *0* through *log n* form a 'n' node full tree.

Consider FT1 of the BDG with the root at $(00\dots00)$. For the $n.n$ sized torus we will map the 'n' *node* top tree ($<$ *tree levels* 0 *through* $log\ n >$) onto row 0 of the torus. Tree *level* $(log\ n+1)$ in FT1 contains the roots of 'n' CBT's of size $(n-1)$ each. For the remaining $(n-1).n$ sized torus, the n CBT's of size $(n-1)$ are mapped such that each such CBT maps onto a separate toroidal column of size $(n-1)$.

For the BDG having a second FT2 with root as $11\dots11$, the top tree ($<$ *levels* 0 *through* $log\ n >$) is mapped onto column 0 and the n, $(n-1)$ node CBT's are mapped onto the $(n-1)$ length rows of the $(n).(n-1)$ node torus- see Fig. 3 and Fig. 4.

Essentially, consider any node X of the BDG. From its position in FT1 of the BDG, it is linked(associated) to a toroidal node as per the FT1 mapping onto the columns of the toroid. For the same node X, from its position in FT2 of the BDG, it is linked to a second toroidal node as per the FT2 mapping. One such ϵ mapping for a 32 node Union-Graph is illustrated in Fig. 4. Node 1000 of the BDG is linked to the toroidal node in row 1, column 0 from the FT1 mapping. From the FT2, mapping the same BDG node 1000 is linked to the toroidal node at row 0, column 2. Thus, each node of the BDG is linked to two distinct torus nodes resulting in the overall degree of the graph as 6 (uniform for all BDG and torus nodes). The overall mapping associates each of the rows and columns of the torus with a CBT in the BDG.

We point out again that there are a number of ways(ϵ) of overlaying a T node CBT onto a T node row (or column) of the torus. Each achieves the same algorithmic purpose and a specific ϵ choice is chosen from the VLSI layout considerations. For one such specific ϵ mapping, the recursive algorithm for determining the placements(mapping) of a BDG node onto a toroidal node is described in [21].

Following a mapping, a BDG and a toroidal node maintain addresses of the nodes of the other sub-graph nodes linked to them in a local table. This is practical as the table size is only 2. It is useful to main-

[2]roots : $00\dots01$ and $11\dots10$

[3]root $00\dots00$, son $00\dots01$, sons $00\dots010$ and $00\dots011$

tain a logical "direct" mapping with the $(m+1)$ bit node representation as (x, sub_graph_addr) with $x = 0$ constituting a BDG node and $x = 1$ constituting the associated toroidal node from either FT 1 or FT 2 mapping. This simplifies the UG routing considerably especially when the routing requires the use of both the Torus and the BDG. Thus, obtaining a routing path as Source(Toroid) $\rightarrow$ Intermediate-Source(BDG) $\rightarrow$ Intermediate-Dest.(BDG) $\rightarrow$ Dest. (BDG or Torus) is straightforward.

For $N \neq 2n^2$, the BDG - Torus graph association is similar to the one described above. For $N = N_{bdg} + N_{torus}$, we still require $N_{bdg} = N_{torus}$ though $N_{bdg} \neq n^2$. We now need to consider a torus of dimensions $P.Q$, i.e., $N_{torus} = P.Q$ such that $P = 2^p$ and $Q = 2^q$. The general procedure for mapping the FT's of the BDG onto the torus remains the same as illustrated above. Whereas we considered root nodes at tree levels $(log\ n + 1)$ in FT1 and FT2 for mapping onto a square mesh: for the rectangular mesh available in the present case, we need to consider the root nodes to be located at tree level $log\ P + 1$ in FT1 and at level $log\ Q + 1$ in FT2 and continue the mapping procedure from Step 2.1 onwards [21].

2.1 Graph Properties

The Union-Graph clearly supports all embeddings supported by the individual sub-graphs of BDG and the Torus. Some additional properties of the combined graph structure are obtained below. For ease of explanation, we consider the specific ϵ mapping detailed in [21] - though a range of such ϵ mappings will result in identical properties. We state the Lemmas here and the proofs are detailed in [21].

Lemma 1: *The Union-Graph is pancyclic.*

Lemma 2: *The N node Union-Graph supports two link disjoint N-1 node CBT's with node congestion 2 at the leaf level.*

Lemma 3: *The* $N = 2n^2$ *node UG has a* $3n^2-6n+3$ *node mesh of trees with node congestion 2.*

Lemma 4: *The diameter of a N-node UG is 1 unit larger than the diameter of a comparable N-node BDG, in both the fault-free and single node/link fault cases.*

For any routing path in the torus exceeding the $log\ N$ bound, the routing procedure reduces to obtaining a partial BDG routing : see Section 2. The Source(Toroid) $\rightarrow$ Destination(BDG) routing is also handled similarly. At most this results in an increase of 2 hops in going to the other sub-graph and the return hop. With each of the sub-graphs of size $N/2$, a one unit increase in routing length results.

The bound for the faulty case is achieved using a novel BDG routing algorithm which has a tight upper bound of $logM + 1$ routing steps for both a fault free or with a single fault in a M-node BDG. This is discussed in [22].

The comparative graph properties are given in the table below. The graph embeddings of BDG, PS graphs and the hypercube are illustrated in Table 1.

Graph Properties : $N = 2^p$ node graphs				
	UG	BDG	HCube	PS
Nodes	$N = 2^p$	$N = 2^p$	$N = 2^p$	$N = 2^p$
valence	6	4	p	8
diameter	p + 1	p	p	p
F.T Dia.	p + 2	p + 1	p + 1	p + 1
Conn^{vty}	4	2	p	4

The versatility of the Union-Graph is based on two factors, admitting (a) almost all key embeddings and (b) flexible and efficient data movements between the embeddings. Stage 1 of an algorithm may necessitate a shuffle based operation with Stage 2 requiring data positioning in a matrix form. For example, the proposed UG will allow a shuffle operation in the BDG with a one step data movement to the mesh. Similarly one can have data placed in the torus, sorted or summed using the tree/shuffle in the BDG. The data is returned to the torus for subsequent operations. In all cases the BDG-Torus data movement is achievable in a single step providing the efficiency in data movements. It may be mentioned that most of the embeddings are also available discretely and concurrently, a unique feature of the proposed structure. The PS graph, in comparison, supports a variety of embeddings. However its complexity and the reconfiguration requirements for the different embeddings, precludes simple and efficient data movement amongst them.

3 Discussion and Examples

The interconnect design yields a graph structure possessing the architectural features of the BDG and the Torus along with efficient data movements between the UG embeddings. A whole range of algorithms exist which can not be executed in the best known time complexity bounds on either the BDG or the Torus separately. On the other hand as shown below, the UG supports a wide range of such algorithms in the optimal time complexity. We describe two distinct cases of algorithms on such a UG structure, **Multi-Phase algorithms** : requiring the use of BDG and Torus in different stages of the algorithm and **Pipelined Algs.** : requiring concurrent use of both the subgraphs BDG and Torus. Following examples illustrate these features.

3.1 Multi-Phase Algorithms

Example 1 : *Matrix-Vector Multiplication*

Matrix-Vector multiplication[4] can be best performed on a $N = n.n$ torus in $O(N^{1/2})$ time - the matrix being stored in the torus and the vector input from a row or column. The same operation takes $O(N^{1/2} + log\ N)$ time on a $N = 2n^2$ node BDG using a CBT embedding. In this the vector is stored in the leaves of the tree and the matrix input row by row. We describe an $O(log\ N)$ algorithm on the $N = 2n^2$ node Union-Graph.

[4] n x n matrix and n sized vector

For the result as $c_{i,1} = \sum_{j=1}^{n} a_{ij}b_{j1} \ \forall i,j \in \{1,..,n\}$, the matrix elements are placed in the torus with the indices of $a_{i,j}$ specifying the (row,col) positions. The vector $b_{j,1}$ is stored in the BDG nodes - at the n "root" nodes at tree level broadcast into the torus structure with element $b_{j,1}$ broadcast into the j^{th} column. The broadcasting is performed in FT1 of the BDG in $O(log\ N)$ time, and using the BDG-Torus links, in a single step the contents are moved to the columns of the torus. Directly we obtain the aligned product terms $a_{i,j}b_{j,1}$ in the torus. Each row j consists of the terms to be summed into a result term $c_{j,1}$. In a single step the datum of each row is transferred onto the CBT's of FT2 in the BDG and the summation operation performed in the CBT's in $O(log\ N)$ time. The overall algorithm is completed in $O(log\ N)$ time on the UG which improves considerably upon the time complexity on both the BDG and the Torus, taken separately.

The procedure, expressed as a sequence of communication patterns – data placement (initial setup), data broadcast (CBT1 broadcast), data assimilation (CBT2 aggregation) , took advantage of the different data positioning in the mesh and the BDG-Torus (or mesh to tree and vice versa). Also, we utilized the data movement features provided by the CBT's in the BDG with the BDG - Toroidal data movement in a single step.

Example 2 : *Non-Parametric Signal Detection*

An oft used signal processing problem is detection of a constant displacement of a signal corrupted by additive noise [23]. The constraints being insensitivity to change in input statistics, a fixed false alarm probability and sampled-data processing. This is a real time problem requiring tight bounds on the time complexity of operation.

The sampled data is present in the form of a matrix **A** which is multiplied by a vector (or a matrix) representing weights corresponding to the strength of each signal and the type of input statistics. The purpose is to obtain A^t with t rounds of multiplication till a certain false-alarm probability measure is obtained. At this stage the entire data set is sorted(ranked) according to magnitude and each sorted value is multiplied with its rank in the sorted list. The final data elements obtained are summed and compared to a predetermined threshold. This results in a final result of signal detection or absence with the desired probability.

Current techniques utilize a mesh interconnect to obtain the result of a N-element problem in O($N^{3/2}$) time. The UG improves the time bound to O$(log\ N)^2$ through the utilization of the Torus in conjunction with the shuffle in the BDG. The initial matrix-vector multiplication is performed using the Torus and CBT in the BDG, as in Example 1 in O$(log\ N)$ time. The ranking stage (bitonic sorting) is performed on the shuffle in the BDG in O$(log\ N)^2$ time and the CBT in the BDG is used to obtain the final data summation in O$(log\ N)$ time. The overall time complexity is thus O$(log\ N)^2$ – a vast improvement over the mesh interconnect. A number of image processing algorithms – noise removal through rank order filters, pixel deletion, rank updating etc. utilize similar mesh and shuffle-based algorithmic steps. This particular example has illustrated the importance of the BDG, as the graph associated with the Torus in the UG, in terms of the inherent shuffle embedding present in it. Another example highlighting the shuffle structure is the multi-phase beam forming algorithm.

Example 3 : *Beam-forming on Phased Arrays*

Beam-forming is one of the main functions for a range of phased-array processing algorithms[9]. Because we wish to support high data rates, this is extremely computationally demanding, requiring extensive FFT computations.

The basic problem is as follows : for a number of input radar sensors, the object is determining the direction of the incoming signal in the presence of distortions. (Without going into the signal processing details) This reduces to a) determining the FFT's of the sampled data input at each of the sensors and b) for the obtained FFT data matrix from (a), suppressing noise interferences by multiplication with a characteristic weight vector.

Again, the UG incorporates the topological features to support these computations. For the shuffle graph ideally suited to FFT calculations, a N/2-point FFT is performed in $O(log\ N)$ time on a N sized BDG. The first phase of operations on the UG is obtaining the FFT for each of the input sensor data, and moving it to a row/column in the Torus[5]. Subsequently, a matrix of such FFT data is multiplied with the weighted vector (or matrix) using the matrix-vector computations discussed in Example 1. The overall operation is of $O(K\ log\ N)$ time complexity for K sensors and N data points per sensor. This is also a real time problem and the time bound is an order better than the existing techniques on other array processors.

One can always select a very specific algorithm which highlights the best features of a given architecture. In this case, though, the example algorithms have been specified by the set of communication patterns and their inter-movements on the Union-Graph. Thus, the range of algorithms which are composed from these basic patterns can be efficiently supported on the Union-Graph. The claim of an efficient interconnect is thereby justified.

3.2 Aspect of Pipelining

One useful feature provided by the UG comes from the fact that the two subgraphs – BDG and Torus – are individually available as discrete graph structures. Hence, distinct operations can concurrently be performed on each of the sub-graphs individually. This is not possible with most other interconnect structures where the mesh or some other topology is available only as an embedding. Also, one can sometimes implement algorithms which are non-implementable on one or both of the sub-graphs taken alone. One such

[5]The data size of the FFT may be larger than the row/column length and this may require partitioning the problem into stages

example which is not implementable on the BDG taken alone is given.

A Jacobi algorithm is described; this incorporates the facets of pipelining in the UG, making use of the availability of a discrete mesh and a discrete tree (in the BDG) being concurrently usable in the UG. Described is a general method to solve a number of mathematical problems such as Laplace equations, differential equations, finite element analysis problems, linear equations and other algebraic problems.

Example : *Jacobi Iteration Method*

An iterative Jacobi algorithm is considered for solving a system of equations [4], [24]. The computation in step (K+1) consists of repetitively updating each point of the grid from the Kth iteration step as follows:

$$x_{i,j}^{K+1} = \frac{1}{4}[x_{i+1,j}^{K} + x_{i-1,j}^{K} + x_{i,j+1}^{K} + x_{i,j-1}^{K}]$$

The overall procedure consists of two distinct stages:

1) Computational Stage : In each $(K+1)^{th}$ iteration, each point of one grid is updated by using the values of the K^{th} iteration of the 4 neighbor in the grid.

2) Halting Stage: The overall computation is stopped when any value from an overall iteration is within a pre-defined error threshold.

The halting stage, to a large extent, determines the efficiency of the algorithm. The commonly-used method allows each individual processor to generate a local decision value and to continue until ALL the processors agree to stop. *Note* that if the system reaches some overall stabilization point (where it can halt), it will remain stable. At each iteration (or at each interval) a global decision must be made whether additional iterations are needed. If even a single processor requires a new iteration to be done, then overall, a new iteration is required, constituting the global iteration scenario. With the N-node grid the most commonly used structure for implementing the Jacobi algorithm, determining the halting point is a message intensive procedure, requiring $O(N^{1/2})$ time on it.

In the $N = 2n^2$ node UG, the computational and especially the halting stage (decision tree - CBT of BDG) can be performed in $O(log\ N)$ time, utilizing the discrete embeddings and pipelining features offered by the graph.

On the UG, the algorithm proceeds as follows :

For each mesh processor :

S1: Compute iteration K on a grid processor; obtain its "*decision*" regarding performing or terminating its next iteration.

S2: Send this decision to its linked processor on the BDG (FT1 or FT2).

Concurrently :

S3.1: Mesh Operation: Start iteration $(K+1)$.

S3.2: BDG Operation: Start assimilating the "decisions" from individual grid processors, in the CBT of the BDG

After $2log\ n$ assimilation steps, if the overall sum of "decision" values equals zero, then broadcast "halt" instruction to the CBT and then onto the mesh nodes.

S4: Continue next mesh iteration until receiving the Halt instruction.

Starting at S3.1, it takes $2log\ n$ steps before a final decision can be obtained at the root of the CBT in the BDG. Next, a further $2log\ n$ steps is necessary for the root to broadcast this decision in the CBT. Each of the CBT nodes then pass this global decision to their associated mesh nodes in the UG, and the algorithm continuation or termination is effected.

Note that there is a pipeline latency of $2log\ n$ between S3 and S4. This follows from the BDG's CBT assimilating the "decisions" in $2log\ n$ steps. Since a global decision regarding proceeding or terminating further iterations in the mesh is not available for this time period, the iterations continue in the mesh. Each iteration's decisions continuously input the pipeline in the CBT. The possible communication congestion in the BDG, with the upward CBT decision data assimilation and the downward CBT final decision data broadcast operations, is handled easily using the 2 link-disjoint CBT's of the BDG – again using a pipelined operation.

The algorithm has implemented pipelining the different stages of the Jacobi algorithm in the two (mesh and CBT) graph structures available **discretely** and **concurrently** in the UG. With each sub-graph handling the communication/computation tasks suited to its physical topology, the pipelining results in a very efficient algorithm operation not feasible in the hypercube or the PS graph. They require separate graph reconfigurations to embed and utilize the mesh and the tree structures discretely or for pipelining.

Jacobi Iteration : $N = 2n^2$ **nodes**			
	Mesh/Torus	BDG	UG
Time Cplx.	O($N^{1/2}$)	not-feasible	O(log N)

Algorithm Examples :

To conclude, Table 2 tabulates the best-known time complexities of some well known algorithms as implemented on different topologies. These are compared to the UG's performance.

An optimal VLSI layout is achievable by using the existing optimal BDG VLSI layout. This issue is discussed in [21].

4 Conclusions

The basic aspects of relating algorithmic patterns with an interconnect structure have been obtained.

Also, we have been interested in elucidating some of these principles in the design phase of an interconnect through the example of a BDG-Torus UG. A set of diverse examples justifies the effectiveness of the design methodology. Through these, the Union-Graph is shown to be a powerful and efficient interconnect structure, comparable and improving upon both the PS graphs and the hypercube.

References

[1] J. C. Bermond and G. Peyrat, "deBruijn and Kautz networks : a competitor for the hypercube ?," in *Hypercube and Distributed Computers* (F.Andre and J.P.Verjus, eds.), pp. 279–293, INRIA, Elsevier Science Publisher B.V.(North-Holland), Oct. 1989.

[2] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation - Numerical Methods*. Englewood Cliffs, NJ-07632: Prentice Hall, 1989.

[3] S. Bokhari, "On the mapping problem," *IEEE Trans. on Computers*, vol. c-30, pp. 207 – 214, Mar 1981.

[4] M. Dubois and J.-C. Wang, "Shared data contention in a cache coherence protocol," *Proc. of 1988 International Conf. on Parallel Processing*, pp. 146 – 153, 1988.

[5] J. R. Goodman and C. H. Sequin, "Hypertree : A multiprocessor interconnection topology," *IEEE Trans. on Computers*, vol. c-30, pp. 923 – 933, Dec 1981.

[6] K. Hwang and J. Ghosh, "Hypernet : A communication- efficient architecture for constructing massively parallel computers," *IEEE Trans. on Computers*, vol. c-36, pp. 1450 –1466, Dec 1987.

[7] O. P. Ibarra and S. M. Sohn, "Hypercube algorithms for some string comparison problems," *Proc. of 1988 International Conf. on Parallel Processing*, pp. 190 – 193, 1988.

[8] D. Jamieson, L. Gannon and R. Douglass, *The Characteristics of Parallel Algorithms*. Cambridge, MA: MIT Press, 1987.

[9] S. Y. Kung, "Spectral Estimation, Beam-forming and Kalman Filtering," in *VLSI Array Processors* , Prentice Hall, 1987.

[10] F. T. Leighton, "Parallel computation using mesh of trees," in *1983 Workshop on Graph-Theoretic Concepts in Computer Science* (Trauner Verlag, Linz), pp. 200 – 218.

[11] S. Levitan, "Measuring communication structures," in *The Characteristics of Parallel Algorithms* (L. Jamieson, ed.), pp. 101 – 137, MIT Press, 1987.

[12] McBryan, "Hypercube algorithms and implementations," *SIAM J. Scientific and Statistical Computing*, 1987.

[13] D. K. Pradhan, "Interconnection topologies for fault-tolerant parallel and distributed architectures," *Proc. of 1981 International Conf. on Parallel Processing*, pp. 238 – 244, 1981.

[14] D. K. Pradhan, "Fault-tolerant VLSI architectures based on deBruijn graphs (Galileo in the Mid Nineties)," In F. Roberts, F. Hwang and C. Monma, editors, *Reliability of Computer and Communication networks: proceedings of DIMACS Workshop*, pp. 183 – 195, December 1989.

[15] D. K. Pradhan and S. M. Reddy, "A fault-tolerant communication architecture for distributed systems," *IEEE Trans. on Computers*, vol. C-31, pp. 863 – 870, Sep 1982.

[16] F. P. Preparata and V. Vuillemin, "The cube-connected cycles : a versatile network for parallel computation," *Comm. of ACM*, vol. 24, pp. 300 – 309, May 1981.

[17] A. L. Rosenberg, "Product- Shuffle graphs : Toward reconciling shuffles and butterflies," COINS-TR, Univ. of Mass., Amherst, Oct 1989.

[18] M. R. Samanthan and D. K. Pradhan, "The deBruijn multiprocessor network : a versatile parallel processing and sorting network for VLSI," *IEEE Trans. on Computers*, vol. c-38, pp. 567 – 581, April 1989. Also see corrections in vol. 40, pp. 122, Jan 1991.

[19] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. on Computers*, pp. 153 – 161, Feb 1971.

[20] N. Suri and A. Mendelson, "On relating algorithms to architectures," *Unpublished manuscript.*

[21] N. Suri, A. Mendelson and D.K. Pradhan, "BDG-Torus Union graph an efficient algorithmically specialized parallel interconnect," TR91-CSE5, Univ. of Massachusetts

[22] N. Suri and D. K. Pradhan, "Notes on fault-tolerant routing in the deBruijn graphs," *Unpublished Manuscript.*

[23] J. B. Thomas, "Non-Parametric Detection," *Proc. IEEE*, vol. 58, pp. 623-631, 1985.

[24] D. Young, *Iterative Solution of Large Linear Systems.* New York: Academic Press, 1971.

Graph Embeddings : $N = 2^p$ node graphs					
Embedding	Union-Graph	BDG	Hypercube	PS Graph	Mesh
CBT's	2^p-1 node	2^p-1 node	spanning trees	2^{p-1}-1	no
Pancyclic	yes	yes	even-cycles	yes	even-cycles
SE	2^{p-1}	2^p	2^p :dil^n $O(log\ N)$	2^p : $dil.$ 2/$Congt^n$ 2	no
Mesh	$2^{p-1}(square)$	no	2^p	2^p [a]	2^p
Mesh of Trees	$3.2^{p-1} - 6.2^{(p-1)/2} + 3$	no	2^{p-1}	$3.2^{p/2} - 2^{p/2+1}$	no

[a]this is possible only for 1 case. For $N = 2^p.2^q$, mesh sub-graph is 2^p or 2^q only.

Table 1:

Time Complexities on Diff. Topologies : IN size matches problem size					
Problem Type	Tree	*Mesh*	BDG	Union-G	HCube
Broadcast	O($log\ N$)	O($N^{1/2}$)	O($log\ N$)	O($log\ N$)	O($log\ N$)
String Matching[a]	-	O(N)	-	O($log\ N$)	O($log\ N$)
Sorting	O(N)	O($N^{1/2}$)	O(log^2 N)	O($log^2 N$)	O($log^2 N$)
Mat-Vector Mult.	O($N^{1/2}$)	O($N^{1/2}$)	O($N^{1/2}$)	O($log\ N$)	O($log\ N$)
Mat-Mult	O(N^2)	O(N)	O($log\ N$)	O($log\ N$)	O($log\ N$)
Jacobi	-	O($N^{1/2}$)	-	O($log\ N$)	O($log\ N$) [b]
Sig-Detect	O(N^2)	O($N^{3/2}$)	O($N^{1/2}$	O($log\ N)^2$	O($logN)^2$ [c]

[a]Longest Common Sub-sequences
[b]with large constant and high edge congestion

Table 2: