Biographical Sketch

Antonio Fernández (Anta) graduated from the Universidad Politécnica de Madrid with the degree of Diplomado en Informática in March 1988 and with the degree of Licenciado en Informática in July 1991. In Fall of 1991 he joined the Ph.D. program in Computer Science at the University of Southwestern Louisiana and was supported by a Fulbright Scholarship. In Fall 1992 he obtained the Master of Science degree in Computer Science.

His professional experience includes positions as system manager at the Departamento de Ingeniería Telemática of the Universidad Politécnica de Madrid for nine months, as an assistant professor for 20 months, and as an associate professor for a further 20 months at the Departamento de Arquitectura y Tecnología de Computadores of the Universidad Politécnica de Madrid. He was recently awarded a post-doctoral research grant by the Spanish Ministry of Education and Science for the year 1995.

Abstract

This dissertation proposes the cartesian product operation for graphs as a unifying framework for the study of interconnection networks. In this research, we concentrate on homogeneous product networks and generate a large set of important general results which yield the characteristics of the product network from those of its factor network. From these characteristics, a network can be evaluated and different networks can be meaningfully compared.

The results of this study are grouped in four main areas. First, we obtain structural properties of homogeneous product networks. We have compiled results on the diameter, vertex degree, connectivity, and partitionability of these networks. Then, we have addressed the study of other properties and derived results on the bisection width and crossing number. To generate these results we introduce a new structural property of a graph, the maximal congestion, which seems to be interesting for future research.

Second, we have obtained simple but powerful results on embeddings between homogeneous product networks. These results allow to transfer the computational power of one network to the other by emulation.

Third, we have developed algorithms that can be implemented in any homogeneous product network without variation. These algorithms cover several important problems: sorting, summation, matrix multiplication, and minimum-weight spanning-tree finding. Some of them can be readily modified to solve many other problems.

Finally, we have studied the VLSI layout complexity of homogeneous product networks, obtaining lower bounds on the area and wire length they require and presenting methods to produce optimal-area layouts.

We have applied these results to several instances of homogeneous product networks, showing how simply the results can be used to evaluate a network. Then, we have concentrated in the study of three of them: the product of complete binary trees, shuffle-exchange graphs, and de Bruijn graphs. These three homogeneous product networks have been shown to be very powerful and interesting candidates for being used as interconnection networks.

In the rest of the proof we first show how to embed a $(r(h-3)+1)$-level tree in the first graph whose leaves are the leaves of $PT_r(2^{h-2}-1)$. Then, we show how to embed a $(3r-\lfloor\frac{r}{2}\rfloor)$-level tree in each copy of $PT_r(7)$ so that the root of the tree is the root of the copy. The combination of both embeddings to $PT_r(N)$ yields the desired embedding.

We first recall from Theorem 7.4 that $PT_r(N)$ has a subgraph isomorphic to the $(r(h-1)+1)$-level tree. By construction, the leaves of this tree are also leaves of $PT_r(N)$. The direct application of this theorem to $PT_r(2^{h-2}-1)$ allows to obtain a subgraph of this graph isomorphic to the $(r(h-3)+1)$-level tree and whose leaves are the leaves of $PT_r(2^{h-2}-1)$.

Corollary A.1 shows how to embed $T(2^{3r-\lfloor\frac{r}{2}\rfloor}-1)$ with congestion 3 and dilation 3 into each copy of $PT_r(7)$ so that the root of the tree is the root of the copy. Combining this result with the previous one we have obtained an embedding of the $(3r-\lfloor\frac{r}{2}\rfloor+r(h-3)) = (rh-\lfloor\frac{r}{2}\rfloor)$-level complete binary tree in $PT_r(N)$ with dilation 3 and congestion 3. ■

has label $x = x_k...x_1$, where $x_i = 1$ for $i = 1, ..., k$, and the edges incident to the root are edges of $PT_k(7)$.

Now consider only the roots of the embedded trees and reconnect them along dimensions $k+1$ and $k+2$. The graph so obtained contains the nodes of $PT_{k+2}(7)$ of the form $x = x_{k+2}x_{k+1}x_k...x_1$, where $x_i = 1$ for $i = 1, ..., k$, and is isomorphic to $PT_2(7)$.

Each node in the above graph is the root of an embedded complete binary tree. Then, considering again the whole graph, we have obtained an embedding of $TT(2^{3k-\frac{k-1}{2}} - 1, 2, 7)$ into $PT_{k+2}(7)$, where the $PT_2(7)$ subgraph and the first two levels of the trees are embedded with dilation 1 and congestion 1. Since the embedding defined in Lemma A.1 only changes this part of the TT graph, it can be applied here to obtain an embedding of the $(3k - \frac{k-1}{2} + 5) = (3(k+2) - \frac{(k+2)-1}{2})$-level complete binary tree in $PT_{k+2}(7)$ with dilation 3 and congestion 3.

From Lemma A.1, the root of the embedded tree is the root of the $PT_2(7)$ subgraph, that is of the form $x = x_{k+2}x_{k+1}x_k...x_1$, where $x_i = 1$ for $i = 1, ..., k+2$, and the edges incident to this root have dilation 1 and congestion 1. ■

Corollary A.1 *The $(3r - \lfloor \frac{r}{2} \rfloor)$-level complete binary tree, $T(2^{3r-\lfloor \frac{r}{2} \rfloor} - 1)$, can be embedded into $PT_r(7)$ with dilation 3 and congestion 3. In this embedding the root of the embedded tree is the root of $PT_r(7)$.*

Proof: If r is odd the above lemma can be trivially applied and the claim follows. The case of even r requires a little more elaboration.

Note that by removing all the dimension-r edges we obtain 7 disjoint copies of $PT_{r-1}(7)$. Since r is even $r-1$ is odd and the above lemma can be applied to each copy. Then $T(2^{3(r-1)-\frac{r-2}{2}})$ can be embedded in each copy with its root in the node $x = x_{r-1}...x_1$, with $x_i = 1$ for $i = 1, ..., r-1$.

We can now connect the roots of the embedded trees with a dimension-r tree. This tree has 3 levels and each of its leaves is the root of a $(3(r-1) - \frac{r-2}{2})$-level tree, therefore we have found an embedding of $T(2^{3r-\frac{r}{2}} - 1)$ into $PT_r(7)$. Since the dimension-r tree connects the roots of the 7 copies and the root of the complete binary tree embedded is the root of this tree, the root of the embedded tree is the node $x = x_r...x_1$, with $x_i = 1$ for $i = 1, ..., r$, root of $PT_r(7)$. ■

Then, we can obtain the proof of Theorem 7.5.

Proof: Note that if we remove the 2 lowest levels from every tree along each dimension in $PT_r(N)$, where $N = 2^h - 1$, we obtain a graph isomorphic to $PT_r(2^{h-2} - 1)$. Similarly, if we remove the $h-3$ top levels from every tree along each dimension we obtain a graph formed by $2^{r(h-3)}$ disjoint copies of $PT_r(7)$. Both graphs have exactly $2^{r(h-3)}$ common nodes, that are the leaves of the $PT_r(2^{h-2} - 1)$ graph and the roots of the copies of $PT_r(7)$ in the other graph.

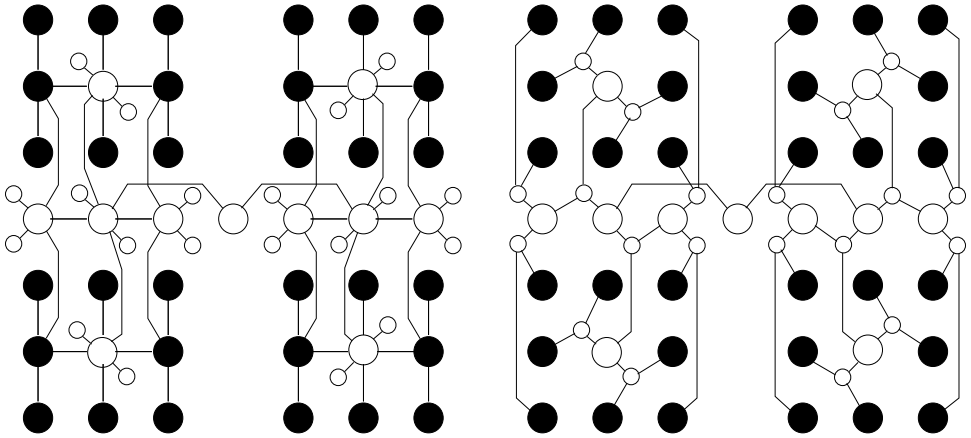

Figure A.1: Embedding the $(l+5)$-level complete binary tree into a subgraph of $TT(2^l - 1, 2, 7)$.

congestion of 3 is found in the edges connecting large empty nodes with small empty nodes in Figure A.1.(a).

Since the tree of Figure A.1.(b) has 6 levels and each dark node represents a collapsed l-level tree, we have obtained an embedding of the $(l+5)$-level complete binary tree into $TT(2^l - 1, 2, 7)$ where the dilation and the congestion are 3. From the figure it is easily verified that the root of the embedded tree coincides with the root of the $PT_2(7)$ subgraph and that the edges incident to the root of the tree are edges of $TT(2^l - 1, 2, 7)$. ■

The properties of the embedding highlighted in the statement of the lemma are needed in order to iteratively apply the embedding (in the next lemma) without increasing the congestion of the global embedding.

Lemma A.2 *The $(3r - \frac{r-1}{2})$-level complete binary tree, $T(2^{3r-\frac{r-1}{2}} - 1)$, can be embedded into $PT_r(7)$, where r is odd, with dilation 3 and congestion 3. In this embedding the root of the embedded tree is the root of $PT_r(7)$ and the edges incident to the root of the embedded tree have dilation 1 and congestion 1.*

Proof: We prove the claim by induction on the number of dimensions, r. The initial condition, $r = 1$, is trivially verified, since $PT_1(7)$ is isomorphic to $T(7)$. In the induction step we have to show that, given an embedding of $T(2^{3k-\frac{k-1}{2}} - 1)$ into $PT_k(7)$ as specified, it is possible to embed $T(2^{3(k+2)-\frac{(k+2)-1}{2}} - 1)$ into $PT_{k+2}(7)$ as described.

Note that by removing all the edges in dimensions $k+1$ and $k+2$ from $PT_{k+2}(7)$ we obtain 49 disjoint copies of $PT_k(7)$. From the induction hypothesis, we can embed a disjoint copy of $T(2^{3k-\frac{k-1}{2}} - 1)$ into each of these copies. The root of the tree embedded

Proof of Theorem 7.5

In order to simplify the proofs we will first distinguish special sets of nodes in the $PT_r(N)$ networks as follows.

Definition A.1 *A node $x = x_r...x_1$ is a leaf of $PT_r(N)$ if and only if x_i is a leaf of $T(N)$, for $i = 1, ..., r$.*

Definition A.2 *The node $x = x_r...x_1$ is the root of $PT_r(N)$ if and only if $x_i = 1$ (* i.e. *x_i is the root of $T(N)$), for $i = 1, ..., r$.*

We now define a new class of graphs that is going to be useful in this section. We do not give a special name to the graphs of this class but we instead use a short notation to identify any member of the class.

Definition A.3 *$TT(M, r, N)$ is the graph obtained by connecting the roots of N^r disjoint copies of $T(M)$ by the $PT_r(N)$ pattern,* i.e. *$TT(M, r, N)$ is obtained by "hanging" a complete binary tree $T(M)$, from each node of $PT_r(N)$.*

We start by presenting some results that will allow us to reach the final result (compiled as Theorem 7.5.) First we show that the complete binary tree with $l + 5$ levels, $T(2^{l+5} - 1)$, can be embedded into $TT(2^l - 1, 2, 7)$ with constant dilation and congestion and that this embedding has particular properties. These properties allow the iterative application of the embedding without increasing the dilation or the congestion. This fact is used to obtain the subsequent results which show how to embed the complete binary tree with $3r - \lfloor \frac{r}{2} \rfloor$ levels, $T(2^{3r - \lfloor \frac{r}{2} \rfloor} - 1)$, into $PT_r(7)$ by iteratively using this first embedding. Finally, by combining this results and Theorem 7.4 the general result is obtained.

Lemma A.1 *$T(2^{l+5} - 1)$ can be embedded into $TT(2^l - 1, 2, 7)$, where $l \geq 2$, with dilation 3 and congestion 3. In this embedding the root of the embedded tree coincides with the root of the $PT_2(7)$ subgraph of $TT(2^l - 1, 2, 7)$ and the edges incident to the root are embedded with dilation 1 and congestion 1.*

Proof: Figure A.1.(a) shows a subgraph of $TT(2^l - 1, 2, 7)$. In this figure, dark nodes represent $T(2^l - 1)$ trees collapsed into supernodes for the purpose of a suitable abstraction for the discussion below. Large empty nodes represent roots of other $T(2^l - 1)$ trees, and small empty nodes represent their immediate children in their $T(2^l - 1)$ trees. The subtrees rooted at small empty nodes are ignored. Figure A.1.(b) shows the tree that can be embedded into this subgraph. The edges shown correspond to the edges of the complete binary tree embedded.

It can be easily checked that any edge in Figure A.1.(b) corresponds to a path of length not more than 3 in Figure A.1.(a). It can be also easily seen that the maximum

Case $x_k \neq y_k$**:** Note that there are p vertex-disjoint paths between $x_k x$ and $x_k y$ along the $PT_k^k(N)$ subgraph. Similarly, there are p vertex-disjoint paths between $y_k x$ and $y_k y$. These two sets of paths follow the same pattern in their respective subgraph and one path in one set has its corresponding path in the other. Along dimension k, a unique path exists between corresponding vertices, $x_k z$ and $y_k z$, for any vertex z of $PT_{k-1}(N)$.

We obtain the first paths by taking the shortest path between $x_k x$ and $x_k y$ and the corresponding path between $y_k x$ and $y_k y$. We can obtain, then, two shortest vertex-disjoint paths given by $x_k x \rightarrow x_k y \rightarrow y_k y$ and $x_k x \rightarrow y_k x \rightarrow y_k y$.

Now, take the $p-1$ paths left between $x_k x$ and $x_k y$. For each path i take an intermediate node $x_k z^i$. Then, $y_k z^i$ is in the corresponding path between $y_k x$ and $y_k y$. We find, then, $p-1$ vertex-disjoint paths as $x_k x \rightarrow x_k z^i \rightarrow y_k z^i \rightarrow y_k y$, for $i = 1, ..., p-1$. Note that, so far, the $p+1$ paths obtained follow the same pattern along dimension k because they connect the same pair of $PT_k^k(N)$ subgraphs. Therefore, at most $2h-1$ $PT_k^k(N)$ subgraphs have been visited by them.

To find the remaining $p_k - 1$ paths, we need to consider two possibilities. In the first case, the degree of y_k is at least equal to the degree of x_k, and the $p_k - 1$ paths can traverse neighbors of $y_k y$ along dimension k. The second case arises when $\delta_x < \delta_y$ and $\delta_{x_k} > \delta_{y_k}$. In this case, we need to use the fact that y will have neighbors in its $PT_k^k(N)$ subgraph not traversed by any of the p initial paths. We may thus use these neighbors to build the required number of new paths. A more formal argument appears below.

Let $p_k \leq \delta_{y_k}$. Then $x_k x$ has exactly $p_k - 1$ and $y_k y$ has at least $p_k - 1$ neighbors along dimension k not used in the above paths. Each of these neighbors is in a different $PT_k^k(N)$ subgraph and no one of these subgraphs has been visited in the above paths. We take $p_k - 1$ of these neighbors. Let $x_k^i x$ denote the ith neighbor of $x_k x$ and $y_k^i y$ the ith neighbor of $y_k y$, for $i = 1, ..., p_k - 1$. We can choose $p_k - 1$ vertices z^i in $PT_{k-1}(N)$ not visited by any of the paths between x and y and obtain $p_k - 1$ new vertex-disjoint paths as $x_k x \rightarrow x_k^i x \rightarrow x_k^i z^i \rightarrow y_k^i z^i \rightarrow y_k^i y \rightarrow y_k y$, for $i = 1, ..., p_k - 1$.

If, on the other hand, $p_k > \delta_{y_k}$, then there are at least $p_k - \delta_{y_k}$ neighbors of y not used in the p paths between x and y. Similarly, $x_k x$ has at least $p_k - 1$ unused dimension-k neighbors. Let $x_k^i x$ denote the ith such neighbor of $x_k x$ and $y_k y^i$ the ith such neighbor of $y_k y$, for $i = 1, ..., p_k - \delta_{y_k}$. Then $p_k - \delta_{y_k}$ paths can be obtained as $x_k x \rightarrow x_k^i x \rightarrow x_k^i y^i \rightarrow y_k y^i \rightarrow y_k y$. The remaining $\delta_{y_k} - 1$ paths can be obtained by using the procedure of the case $p_k \leq \delta_{y_k}$.

We have, thus, found $p + p_k$ vertex-disjoint paths, and the claim follows.

This concludes the proof of the theorem.

different dimension-2 tree. Let $x_2x_1^i$ be one of these neighbors of x, then $y_2y_1^i = y_2x_1^i$ is also a neighbor of y, for $i = 1, ..., p_1$. Then, p_1 paths between x and y can be obtained as $x \rightarrow x_2x_1^i \rightarrow y_2x_1^i \rightarrow y$, for $i = 1, ..., p_1$.

Finally, note that each of x and y has at least $p_2 = min\{\delta_{x_2}, \delta_{y_2}\} - 1$ neighbors along dimension 2 not traversed in previous paths, each in a non-visited dimension-1 tree. We can choose p_2 of these neighbors from each of x and y. Since at most $p_1 + 1 \leq 4$ dimension-2 trees have been visited by the previous paths, there are at least $N - 4$ non-visited dimension-2 trees. We can, then, choose any one of these non-visited dimension-2 trees, v^i, and obtain a path through the ith neighbor as $x \rightarrow x_2^ix_1 \rightarrow x_2^iv^i \rightarrow y_2^iv^i \rightarrow y_2^iy_1 \rightarrow y$, for $i = 1, ..., p_2$. This completes the set of m paths and the claim is shown to be true for this case.

Case $x_2 = y_2$**:** This case in analogous to the previous one.

Case $x_2 \neq y_2, x_1 \neq y_1$**:** The two existing shortest paths can be obtained as $x \rightarrow x_2y_1 \rightarrow y$ and $x \rightarrow y_2x_1 \rightarrow y$. Note that at most $2h - 1$ (where $N = 2^h - 1$) dimension-1 and dimension-2 trees are visited by these paths and that only one neighbor of each x and y along each dimension has been traversed. Furthermore, the remaining neighbors along dimension 1 (resp. dimension 2) are in a non-visited dimension-2 (resp. dimension-1) tree. We can, then, obtain $p_{12} = min\{\delta_{x_1}, \delta_{y_2}\} - 1$ paths as $x \rightarrow x_2x_1^i \rightarrow y_2^ix_1^i \rightarrow y_2^iy_1 \rightarrow y$, for $i = 1, ..., p_{12}$. Similarly, $p_{21} = min\{\delta_{x_2}, \delta_{y_1}\} - 1$ paths can be obtained as $x \rightarrow x_2^ix_1 \rightarrow x_2^iy_1^i \rightarrow y_2y_1^i \rightarrow y$, for $i = 1, ..., p_{21}$.

The remaining $l = m - p_{12} - p_{21} - 2$ paths, if $l > 0$, may be obtained as $x \rightarrow x_2x_1^i \rightarrow u^ix_1^i \rightarrow u^iy_1^i \rightarrow y_2y_1^i \rightarrow y$ or $x \rightarrow x_2^ix_1 \rightarrow x_2^iv^i \rightarrow y_2^iv^i \rightarrow y_2^iy_1 \rightarrow y$, where u^i are dimension-1 trees not visited in previous paths and v^i are dimension-2 trees not visited in previous paths. A simple case analysis shows that such trees exist.

This completes the proof for 2 dimensions. For the purpose of induction, we take two vertices x and y of $PT_{k-1}(N)$, $x \neq y$. Assume that they are connected by p vertex-disjoint paths, where, without loss of generality, $p = \delta_x$ and $p \leq \delta_y$. In $PT_k(N)$ these vertices become x_kx and y_ky, respectively, where x_k and y_k are the labels for the dimension k. The minimum vertex degree of the pair is $p + p_k = min\{\delta_{x_kx}, \delta_{y_ky}\}$, where $1 \leq p_k \leq 3$, and x_kx has at least p_k neighbors along dimension k.

Case $x_k = y_k$**:** Each x_kx and y_ky has exactly p_k neighbors along dimension k. These neighbors are not visited by the initial p paths because they are in a different $PT_k^k(N)$ subgraph. If x_k^ix is a neighbor of x_kx, for $i = 1, ..., p_k$, then $y_k^iy = x_k^iy$ is also neighbor of y_ky, they are both in the same $PT_k^k(N)$ subgraph, and no other neighbor is in that subgraph. One path can be found, then, along this subgraph between x_k^ix and x_k^iy, for $i = 1, ..., p_k$, and therefore p_k new paths can be obtained between x_kx and y_ky. Then, the total number of paths obtained is $p + p_k$ and the claim follows.

Appendix

Proof of Theorem 7.1

Clearly, m is an upper bound on the number of vertex-disjoint paths. We need to show that it is also a lower bound, by showing how to find m such paths. For this proof we define the concepts of "use of a tree" and "visit of a tree." By "use", we mean the traversal of at least one edge in the tree. By "visit", we mean the traversal of at least one node in the tree. Clearly, the use of a tree implies a visit of the tree, whereas a visit of a tree may not use the tree (*i.e.* if no edges are traversed.)

Briefly, we obtain the paths in two phases. Initially, we obtain as many vertex-disjoint shortest paths between the nodes as possible. Then, we obtain the rest of the paths by determining routes between neighbors of the nodes not traversed by the previous paths, along trees not previously visited. This guarantees the vertex disjointness of the paths.

The proof proceeds by induction on the number of dimensions. We start by establishing the base case for 2 dimensions. We use a case-by-case study to construct the appropriate number of paths and show that the claim is true for two dimensions.

For the induction hypothesis, we assume p paths for the $PT_{k-1}(N)$ subgraph obtained by taking only $k-1$ dimensions, where p is consistent with the claims of the theorem. For the inductive step we add another dimension and we restrict our attention to the newly added dimension, while treating the rest of the graph as a unit. We show, once again by construction, that an appropriate number of paths, as suggested by the various individual cases, is added as a result of introducing the new dimension.

Along the proof we use δ_x to represent the vertex degree of a generic vertex x, while δ_{x_i} refers to the degree of the vertex x along dimension i. In addition, we use the notation x_i^j to denote the jth neighbor of x along dimension i.

Then, we start by considering the case for $r = 2$. The claim for $N = 3$ is trivially true and $N > 3$ will be assumed. Let $x = x_2x_1$ and $y = y_2y_1$ be two vertices of $PT_2(N)$, $x \neq y$.

Case $x_1 = y_1$**:** The first path is obtained by just noting that x and y are in the same dimension-2 tree and that a path can be found in this tree.

Another set of paths is derived from the fact that each of x and y has exactly $p_1 = \delta_{x_1} = \delta_{y_1}$ neighbors along dimension 1 and each of these neighbors is in a

[86] A. Youssef, "Cartesian Product Networks," in *Proceedings of the 1991 International Conference on Parallel Processing*, vol. I, pp. 684–685, Aug. 1991.

[87] M. Zubair and S. N. Gupta, "Embeddings on a Boolean Cube," *BIT*, vol. 30, pp. 245–256, 1990.

[73] D. D. Sherlekar and J. JáJá, "Embedding Graphs in Binary Trees," in *Computing and Information: Proceedings of the International Conference on Computing and Information, ICCI'89* (R. Janichi and W. W. Kczkodaj, eds.), (Toronto, Canada), pp. 111–115, Elsevier Science Publisher B. V. (North-Holland), May 1989.

[74] Y. Shiloach and U. Vishkin, "An $O(\log n)$ Parallel Connectivity Algorithm," *Journal of Algorithms*, vol. 3, pp. 57–67, 1982.

[75] H. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, vol. C-20, pp. 153–161, Feb. 1971.

[76] O. Sýkora and I. Vrťo, "On the Crossing Number of the Hypercube and the Cube Connected Cycles," in *Proceedings of 17th International Workshop, WG'91, Graph-Theoretic Concepts in Computer Science* (G. Schmidt and R. Berghammer, eds.), vol. 570 of *Lecture Notes in Computer Science*, pp. 214–218, Fischbachau, Germany: Springer Verlag, June 1991.

[77] C. D. Thompson, "Area-Time Complexity for VLSI," in *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, (Atlanta), pp. 81–88, May 1979.

[78] C. D. Thompson, *A Complexity Theory for VLSI.* PhD thesis, Carnegie-Mellon University, Aug. 1980.

[79] C. D. Thompson and H. T. Kung, "Sorting on a Mesh-Connected Parallel Computer," *Communications ACM*, vol. 20, pp. 263–271, Apr. 1977.

[80] J. D. Ullman, *Computational Aspects of VLSI.* Rockville: Computer Science Press, 1984.

[81] L. G. Valiant, "Universality Considerations in VLSI Circuits," *IEEE Transactions on Computers*, vol. C-30, pp. 135–140, Feb. 1981.

[82] P. M. Weichsel, "Products of Highly Regular Graphs," in *Progress in Graph Theory* (J. A. Bondy and U. S. R. Murty, eds.), Ontario: Academic Press, 1984.

[83] M. Yoeli, "Binary Ring Sequences," *Amer. Math. Monthly*, vol. 69, pp. 852–855, 1962.

[84] A. S. Youssef and B. Narahari, "The Banyan-Hypercube Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 160–169, 1990.

[85] A. Youssef, "Product Networks: A Unified Theory of Fixed Interconnection Networks," Tech. Rep. GWU-IIST-90-38, Institute for Information Science and Technology, The George Washington University, Washington, D.C., Dec. 1990.

[60] N. Ranganathan and S. Venugopal, "An Efficient VLSI Architecture for Template Matching," in *Proceedings of the 1994 International Conference on Parallel Processing*, vol. I, (St. Charles, IL), pp. 224–231, CRC Press Inc., Aug. 1994.

[61] A. L. Rosenberg, "Product-Shuffle Networks: Toward Reconciling Shuffles and Butterflies," *Discrete Applied Mathematics*, vol. 37/38, pp. 465–488, July 1992.

[62] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867–872, 1988.

[63] G. Sabidussi, "Graphs with Given Group and Given Graph-Theoretical Properties," *Canadian Journal of Mathematics*, vol. 9, pp. 515–525, 1957.

[64] G. Sabidussi, "Graph Multiplication," *Math. Zeitschr.*, vol. 72, no. 5, pp. 446–457, 1960.

[65] K. Sado and Y. Agarasi, "Some Parallel Sorts on a Mesh-Connected Processor Array and their Time Efficiency," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 398–410, Sept. 1986.

[66] I. Scherson, S. Sen, and A. Shamir, "Shear-Sort: A True Two-Dimensional Sorting Technique for VLSI Networks," in *Proceedings of the IEEE-ACM International Conference on Parallel Processing*, pp. 903–908, Aug. 1986.

[67] C. P. Schnorr and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, (Berkeley, CA), pp. 255–263, May 1986.

[68] E. J. Schwabe, "Embedding Meshes of Trees into deBruijn Graphs," *Information Processing Letters*, vol. 43, pp. 237–240, 1992.

[69] H. S. Shapiro, "The Embedding of Graphs in Cubes and the Design of Sequential Relay Circuits," unplublished Bell Telephone Laboratories Memorandum, July 1953.

[70] D. D. Sherlekar and J. JáJá, "Layouts of Graphs of Arbitrary Degree," in *Proceedings of the 25th Annual Allerton Conference*, Sept. 1987.

[71] D. D. Sherlekar and J. JáJá, "Balanced Graph Dissections and Layouts for Hierarchical VLSI Layout Design," Tech. Rep. CSE-TR-22-89, Department of Electrical Engineering and Computer Sciente, University of Michigan, Ann Arbor, 1989.

[72] D. D. Sherlekar and J. JáJá, "Input Sensitive VLSI Layouts for Graphs of Arbitrary Degree," in *Proceedings of the 3rd Aegean Workshop on Computing, AWOC 88: VLSI Algorithms and Architectures* (J. H. Reif, ed.), vol. 319 of *Lecture Notes in Computer Science*, pp. 268–277, Corfu, Greece: Springer Verlag, July 1988.

[48] K. J. Liszka and K. E. Batcher, "A Generalized Bitonic Sorting Network," in *Proceedings of the 1993 International Conference on Parallel Processing*, vol. I, pp. 105–108, 1993.

[49] T. Nakatani, S.-T. Huang, B. W. Arden, and S. K. Tripathi, "K-Way Bitonic Sort," *IEEE Transactions on Computers*, vol. 38, pp. 283–288, Feb. 1989.

[50] D. Nassimi and S. Sahni, "Bitonic Sort on a Mesh-Connected Parallel Computer," *IEEE Transactions on Computers*, vol. C-27, pp. 2–7, Jan. 1979.

[51] D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI Networks for Parallel Processing Based on Orthogonal Trees," *IEEE Transactions on Computers*, vol. C-32, pp. 569–581, June 1983.

[52] J. Nešetřil and V. Rödl, "Products of Graphs and Their Applications," in *Proceedings of Graph Theory, Lagów 1981* (M. Borowiecki, J. W. Kennedy, and M. M. Syslo, eds.), vol. 1018 of *Lecture Notes in Mathematics*, pp. 151–160, Lagów: Springer Verlag, 1981.

[53] S. R. Öhring and S. K. Das, "The Folded Petersen Cube Networks: New Competitors for the Hypercube," in *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Computing*, pp. 582–589, Dec. 1993.

[54] S. R. Öhring and S. K. Das, "The Folded Petersen Network: A New Communication-Efficient Multiprocessor Topology," in *Proceedings of the 1993 International Conference on Parallel Processing*, vol. I, pp. 311–314, Aug. 1993.

[55] S. R. Öhring and S. K. Das, "Mapping Dynamic Data and Algorithm Structures into Product Networks," in *Proceedings of ISAAC'93*, (Hong Kong), pp. 147–156, Dec. 1993.

[56] S. R. Öhring and D. H. Hohndel, "Optimal Fault-Tolerant Communication Algorithms on Product Networks using Spanning Trees," in *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, (Dallas,TX), Oct. 1994.

[57] R. B. Panwar and L. M. Patnaik, "Solution of Linear Equations on Shuffle-Exchange and Modified Shuffle Exchange Networks," in *Proceedings of the 26th Allerton Conference*, pp. 1116–1125, 1988.

[58] F. Preparata and J. Vuillemin, "Area-optimal VLSI Network for Matrix Multiplication," in *Proceedings of the 14th Princeton Conference on Information Science and Systems*, pp. 300–309, 1980.

[59] F. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications ACM*, vol. 24, pp. 300–309, May 1981.

[34] P. C. Kainen, "A Lower Bound for Crossing Numbers of Graphs with Applications to K_n, $K_{p,q}$, and $Q(d)$," *Journal of Combinatorial Theory*, vol. 12, pp. 287–298, 1972.

[35] D. J. Kleitman, "The Crossing Number of $K_{5,n}$," *Journal of Combinatorial Theory*, vol. 9, pp. 315–323, 1971.

[36] D. Knuth, *Searching and Sorting*, vol. 3 of *The Art of Computer Programming*. Reading, MA: Addison-Wesley, 1973.

[37] R. Koch, T. Leighton, B. Maggs, S. Rao, and A. L. Rosenberg, "Work-Preserving Emulations of Fixed-Connection Networks," in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, (Seattle), pp. 227–240, May 1989.

[38] D.-L. Lee and K. E. Batcher, "On Sorting Multiple Bitonic Sequences," in *Proceedings of the 1994 International Conference on Parallel Processing*, vol. I, (St. Charles,IL), pp. 121–125, Aug. 1994.

[39] F. T. Leighton, "New Lower Bound Techniques for VLSI," in *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pp. 1–12, 1981.

[40] F. T. Leighton, "A Layout Strategy for VLSI Which Is Provably Good," in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, (San Francisco, CA), pp. 85–98, May 1982.

[41] F. T. Leighton, *Complexity Issues in VLSI*. Cambridge: The MIT Press, 1983.

[42] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. San Mateo: Morgan Kaufmann, 1992.

[43] C. E. Leiserson, "Area-Efficient graph Layout (for VLSI)," in *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pp. 270–281, Oct. 1980.

[44] C. E. Leiserson, *Area-Efficient VLSI Computation*. PhD thesis, Carnegie-Mellon University, 1981. The MIT Press, 1983.

[45] E. L. Leiss and H. N. Reddy, "Embedding Complete Binary Trees into Hypercubes," *Information Processing Letters*, vol. 38, pp. 197–199, 1991.

[46] R. J. Lipton and R. E. Tarjan, "A Separator Theorem for Planar Graphs," *SIAM Journal on Applied Mathematics*, vol. 36, pp. 177–189, Apr. 1979.

[47] R. J. Lipton and R. E. Tarjan, "Applications of a Planar Separator Theorem," *SIAM Journal on Computing*, vol. 9, pp. 615–627, Aug. 1980.

[22] K. Efe, "Embedding Mesh of Trees in the Hypercube," *Journal of Parallel and Distributed Computing*, vol. 11, pp. 222–230, Mar. 1991.

[23] K. Efe and K. Ramaiyer, "Congestion and Fault Tolerance of Binary Tree Embeddings on Hypercube," in *Proceedings of the 5th International Parallel Processing Symposium*, (Anaheim, CA), pp. 458–463, May 1991.

[24] T. El-Ghazawi and A. Youssef, "A Unified Approach to Fault-Tolerant Routing," in *Proceedings of the 12th International Conference on Distributed Computing Systems*, (Yokohama, Japan), pp. 210–217, June 1992.

[25] R. Feldmann and P. Mysliwietz, "The Shuffle Exchange Network has a Hamiltonian Path," in *Proceedings of Mathematical Foundations of Computer Science*, pp. 246–254, 1992.

[26] J. P. Fishburn and R. A. Finkel, "Quotient Networks," *IEEE Transactions on Computers*, vol. 31, pp. 288–295, Apr. 1982.

[27] R. W. Floyd and J. D. Ullman, "The Compilation of Regular Expressions into Integrated Circuits," *J. ACM*, vol. 29, pp. 603–622, July 1982.

[28] E. Ganesan and D. K. Pradhan, "The Hyper-deBruijn Networks: Scalable Versatile Architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 962–978, Sept. 1993.

[29] D. Greenberg, L. Heath, and A. L. Rosenberg, "Optimal Embeddings of Butterfly-like Graphs in the Hypercube," *Mathematical Systems Theory*, vol. 23, no. 1, pp. 61–77, 1990.

[30] F. Harary, "On the Group of the Composition of Two Graphs," *Duke Mathematical Journal*, vol. 26, pp. 29–34, Mar. 1959.

[31] I. Havel and P. Liebl, "Embedding the Polytomic Tree into the n-cube," *Časopis pro Pěstován í Matematiky*, vol. 98, pp. 307–314, 1973.

[32] R. Heckmann, R. Klasing, B. Monien, and W. Unger, "Optimal Embeddings of Complete Binary trees into Lines and Grids," in *Proceedings of 17th International Workshop, WG'91, Graph-Theoretic Concepts in Computer Science* (G. Schmidt and R. Berghammer, eds.), vol. 570 of *Lecture Notes in Computer Science*, pp. 25–35, Fischbachau, Germany: Springer Verlag, June 1991.

[33] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, "Computing Connected Components on Parallel Computers," *Communications ACM*, vol. 22, pp. 461–464, Aug. 1979.

[11] S. N. Bhatt and C. E. Leiserson, "Minimizing Wire Delay in VLSI Layouts," MIT VLSI memo 82-86, 1982.

[12] L. Bhuyan and D. P. Agrawal, "Generalized Hypercubes and Hyperbus Structures for a Computer Network," *IEEE Transactions on Computers*, vol. C-33, pp. 323–333, 1984.

[13] M. Y. Chan, F. Y. L. Chin, and C. K. Poon, "Optimal Specified Root Embedding of Full Binary Trees in Faulty Hypercubes," in *Proceedings of the 2nd International Symposium on Algorithms* (W. L. Hsu and R. C. T. Lee, eds.), vol. 557 of *Lecture Notes in Computer Science*, pp. 241–250, Taipei, R.O.C.: Springer Verlag, Dec. 1991.

[14] M. Y. Chan and S.-J. Lee, "Fault-Tolerant Embedding of Complete Binary Trees in Hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 277–288, Mar. 1993.

[15] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs, "The Bandwidth Problem for Graphs and Matrices-A Survey," *Journal of Graph Theory*, vol. 6, pp. 223–254, 1982.

[16] F. R. K. Chung, "Labelings of Graphs," in *Selected Topics in Graph Theory 3* (L. W. Beineke and R. J. Wilson, eds.), pp. 151–168, Academic Press, 1988.

[17] J. Chvátalová, "Optimal Labelling of a Product of Two Paths," *Discrete Mathematics*, vol. 11, pp. 249–253, 1975.

[18] O. Collins, S. Dolinar, R. McEliece, and F. Pollara, "A VLSI Decomposition of the deBruijn Graph," *J. ACM*, vol. 39, pp. 931–948, Oct. 1992.

[19] R. Cypher and C. G. Plaxton, "Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, (Baltimore, Maryland), pp. 193–203, May 1990.

[20] S. K. Das and A. K. Banerjee, "Hyper Petersen Networks: Yet Another Hypercube-Like Topology," in *Proceedings of the 4th Symposium on the Frontiers of Massively Parallel Computation*, (McLean, VA), pp. 270–277, Computer Society Press, Oct. 1992.

[21] A. M. Despain and D. A. Patterson, "X-Tree: A Tree Structured Multi-Processor Computer Architecture," in *Proceedings of the 5th Annual Symposium on Computer Architecture*, pp. 144–151, 1978.

Bibliography

[1] F. Annexstein, M. Baumslag, and A. L. Rosenberg, "Group Action Graphs and Parallel Architectures," *SIAM Journal on Computing*, vol. 19, pp. 544–569, June 1990.

[2] K. Batcher, "Sorting Networks and their Applications," in *Proceedings of the AFIPS Spring Joint Computing Conference*, vol. 32, pp. 307–314, 1968.

[3] K. E. Batcher, "On Bitonic Sorting Networks," in *Proceedings of the 1990 International Conference on Parallel Processing*, vol. I, pp. 376–379, 1990.

[4] M. Baumslag and F. Annexstein, "A Unified Framework for Off-Line Permutation Routing in Parallel Networks," *Math. Systems Theory*, vol. 24, no. 4, pp. 233–251, 1991.

[5] S. N. Bhatt, F. R. K. Chung, J. W. Hong, F. T. Leighton, and A. L. Rosenberg, "Optimal Simulations by Butterfly Networks," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, (Chicago), pp. 192–204, May 1988.

[6] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "Optimal Simulations of Tree Machines," in *Proceedings of 27th Annual Symposium on Foundations of Computer Science*, pp. 274–282, Oct. 1986.

[7] S. N. Bhatt, F. R. K. Chung, J.-W. Hong, F. T. Leighton, B. Obrenić, A. L. Rosenberg, and E. J. Schwabe, "Optimal Emulations by Butterfly-Like Networks," *J. ACM*, 1994. to appear.

[8] S. N. Bhatt and I. C. F. Ipsen, "How to Embed Trees in Hypercubes," Tech. Rep. RR-443, Department of Computer Science, Yale University, New Haven, CT, 1985.

[9] S. N. Bhatt and F. T. Leighton, "A Framework for Solving VLSI Graph Layout Problems," *Journal of Computer and System Sciences*, vol. 28, pp. 300–343, 1984.

[10] S. N. Bhatt and C. E. Leiserson, "How to Assemble Tree Machines," in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, (San Francisco, CA), pp. 77–84, May 1982.

lay out these networks in such a way that, in many cases, the layout is optimal in area and almost optimal in wire length.

- We have exhaustively applied all the above results to several specific instances of product networks, as a proof of the power of the approach. Then we have concentrated in three specific networks and we have been able to show, among other properties, that these product networks are more powerful than their respective factor graphs, with a small increase in cost. The emulation capabilities of these networks present them as important candidates for popular interconnection networks.

As a conclusion, we have been able to create a basic framework for a general theory where existing and new homogeneous product networks can be evaluated. Observe that we have reached generality without losing efficiency. The methods to obtain VLSI layouts for the networks, besides their generality, produce very efficient layout in most of the cases (layouts optimal in area and almost optimal in wire length.) Similarly, the sorting algorithm developed has optimal complexity in a bounded-dimension grid and has the same complexity as the most popular sorting algorithm in the hypercube.

Like any research of this generality, there is room for extensions. More results on other structural properties of homogeneous product networks can be derived. Even our bounds on the bisection width and the crossing number can be made tighter. For instance, it is not known the exact value of the bisection width of such a simple product network as the N^r-node r-dimensional grid when N is odd [42].

Many new general algorithms can be developed for homogeneous product networks. Specifically, because we have assumed a SIMD model of computation, we have only briefly addressed the routing problems in homogeneous product networks (sections 1.1 and 5.2.) However, if a MIMD model is assumed, many new routing problems arise which can be addressed using the proposed framework. For instance, it seems to be very interesting to obtain a general wormhole-routing algorithm for homogeneous product networks.

Finally, since general product networks need not be built with the same factor graph in each dimension, it is necessary to extend all the investigations conducted and the open problems presented to heterogeneous product networks. For instance, at the end of Section 6.4 we briefly outlined how the results obtained in VLSI complexity for homogeneous product networks could be extended to heterogeneous product networks.

Chapter 8

Conclusions

The main contribution of the dissertation described in this prospectus is to conduct an exhaustive study of homogeneous product networks as interconnection networks. To our knowledge, this is the first study of this kind realized. This study is useful given the number of product networks already proposed in the literature. This study can also be used to evaluate properties of new interconnection networks for parallel architectures.

The comprehensive investigation of homogeneous product networks has been developed along several lines:

- We have obtained general results on the structural properties of product networks. We have presented important characteristics like the vertex degree, the diameter, the partitionability, and the connectivity. We specially emphasize the results on the bisection width and the crossing number, since they are not easily derived from same properties of the factor graph. To obtain these properties we have defined a new parameter of a graph, the maximal congestion, which we believe will be important in future research.

- We have obtained several general embedding properties, applicable to any product network. The combination of these properties with embeddings between factor graphs allows to obtain important embedding results for product networks, as has been seen in the presented examples. These results will allow to meaningfully compare the relative powers of product networks.

- We have produced general algorithms whose performance is optimal for some instances of product networks. The algorithms developed allow to sort, compute summations, multiply matrices, and find the minimum-weight spanning tree of a graph in homogeneous product networks. Other algorithms with similar structure are simply derived from the ones presented.

- We have obtained lower bounds on the VLSI layout area and wire length required by homogeneous product networks. We have also developed procedures to effectively

our networks emulate the grid efficiently while they require logarithmic dilation to host it. Observe again that the cost in layout area of the additional power is not high if we bound the number of dimensions.

Therefore, depending on the specific purpose of the network we offer three interesting candidates with bounded degree. If still more computational power is needed, it might be necessary to use higher-cost networks with unbounded vertex degree, like the hypercube.

	$PT_r(N)$	$PS_r(N)$	$PD_r(N)$
$PR_r(N)$	$d = 3$, $c = 2$	$d = 3$, $c = 2$	Subgraph
$PL_r(N)$	$d = 3$, $c = 2$	Subgraph	Subgraph
M.o.T.	Subgraph	$d = 2$, $c = 2$	Subgraph
C.B.T.	Subgraph	$d = 2$, $c = 2$	Subgraph
$S(N^r)$	-	$d = 2r$, $c = 2$	$d = 4r$, $c = 4$
$D(N^r)$	-	$d = 2r$, $c = 8$	$d = r$, $c = 4$
$PT_r(N-1)$	N/A	$d = 2$, $c = 2$	Subgraph
$PS_r(N)$	-	N/A	$d = 2$, $c = 2$
$PD_r(N)$	-	$d = 2$, $c = 2$	N/A

Table 7.2: Embedding capabilities of the product of complete binary trees, shuffle-exchange, and de Bruijn graphs.

From the figures in Table 7.1 we can see that all the three networks present interesting properties. Their diameter is logarithmic with respect to the number of nodes, they have large bisection width, and rather large connectivity if r is large. Also, all of them perform very efficiently the algorithms presented. Finally, the layout area required for them is reasonably small. The product of complete binary trees has same asymptotic layout area as the mesh of trees, while we have shown that the former is more powerful. The other two networks have similar asymptotic layout area than their factor networks if r is bounded, while we proved that they are more powerful.

However, the real power of the networks presented comes from their embedding capabilities. The product of complete binary trees (and its extension) can efficiently emulate the torus, the mesh of trees, and the complete binary tree. Any embedding of the n-node mesh of trees into the similar-size grid requires $\Omega(n/\log n)$ dilation, given their respective diameters, while no efficient emulations of the grid by the mesh of trees is known. Thus, the $PT_r(N)$ network (and specially the $PX_r(N)$ network) seems more powerful than both these networks, and the logical option if we need a network with the capabilities of the grid and the mesh of trees. If we add to this that the network has the same layout area complexity as the mesh of trees and, for more than 2 dimensions, than the grid, it looks like a clear substitute to them.

Now, if further computational power is needed, we can use the products of shuffle-exchange or de Bruijn graphs. By emulation, these networks give us the power of the $PT_r(N)$ network plus the power of hypercube-derived networks. This fact will speed up some computations not suited to be performed neither on the grid nor on the mesh of trees (for instance, sorting) but which are very efficiently performed in a hypercube-derived network. Another advantage of these networks over the traditional hypercube-derived networks used (pure shuffle-exchange, de Bruijn, butterfly, cube-connected cycles) is that

Property/algorithm	$PT_r(N)$	$PS_r(N)$	$PD_r(N)$
Nodes	N^r	N^r	N^r
Edges	$r(N-1)N^{r-1}$	$3rN^r/2$	$2rN^r$
Diameter	$2r(\log(N+1)-1)$	$r(2\log N-1)$	$r\log N$
Connectivity	r	r	$2r$
δ	r	$3r$	$4r$
Δ	$3r$	$3r$	$4r$
Partitionability	2^i, $i=0,...,\log(N+1)$	-	-
Maximal congestion	$O(N^{r+1})$	$O(N^r\log N)$	$O(N^r\log N)$
Bisection width	$\Theta(N^{r-1})$	$\Theta(N^r/\log N)$	$\Theta(N^r/\log N)$
Crossing number	$\Omega(N^{2(r-1)})$	$\Omega(N^{2r}/\log^2 N)$	$\Omega(N^{2r}/\log^2 N)$
Sorting	$O(r^2N)$	$O(r^2\log^2 N)$	$O(r^2\log^2 N)$
Summation	$O(r\log N)$	$O(r\log N)$	$O(r\log N)$
Matrix multiplication	$O(r\log N)$	$O(r\log N)$	$O(r\log N)$
Min.-weight span. tree	$O(r^2\log^2 N)$	$O(r^2\log^2 N)$	$O(r^2\log^2 N)$
Layout area	$\Theta(N^{2(r-1)})$ (for $r>2$)	$\Theta(N^{2r}/\log^2 N)$	$\Theta(N^{2r}/\log^2 N)$
Max. wire length	$O(N^{r-1})$ (for $r>2$)	$O(N^r/\log N)$	$O(N^r/\log N)$

Table 7.1: Comparison of the properties of the product of complete binary trees, shuffle-exchange, and de Bruijn graphs.

Therefore, the first and last edges of the paths are traversed by two of the four paths and the internal edges of the paths are traversed by the four paths. Since the edges traversed by these four paths are not traversed by any other path, we can conclude that the congestion of the embedding is at most four. Since for $r = 2$ the paths have length 2 and have no internal edges, the congestion in this case is only 2. ■

It has been shown in [26] that $D(2^k)$ can host $D(2^{k+j})$ with unit dilation cost. In the resultant emulation, each vertex of $D(2^k)$ is assigned exactly 2^j nodes (the load of the embedding.) The proof is based on the observation that, by erasing the rightmost j bits from vertex labels of $D(2^{k+j})$, we obtain a graph isomorphic to $D(2^k)$. By the same observation, the following results can be stated.

Corollary 7.3 *$D(N^{r+j})$ can be embedded onto $PD_r(N)$ with dilation r, congestion 4 (2 if $r = 2$), and load N^j.*

And moreover,

Corollary 7.4 *$PD_r(N2^k)$ can be embedded onto $PD_r(N)$ with unit dilation, unit congestion, and load 2^{kr}.*

Therefore, for fixed r, a small size $PD_r(N)$ architecture can easily emulate larger size machines with proportional slowdown in the running time.

Finally, the next result shows that products of de Bruijn graphs are more powerful than the pure de Bruijn graphs (this is an extension of a similar result in [61] given for two dimensions.)

Theorem 7.13 *Any embedding of $PD_r(N)$ onto $D(N^r)$ requires dilation $\Omega(\log(r \log N))$.*

Proof: From Corollary 4.4, we know that the grid $PL_r(N)$ is a subgraph of $PD_r(N)$. It is shown in [7] that any embedding of $PL_r(N)$, for $r > 1$, onto $D(N^r)$ requires dilation cost $\Omega(\log \log N^r)$. Hence, the claim follows. ■

Like the $PS_r(N)$ network, the VLSI layout area required by $PD_r(N)$ is asymptotically the same as that required by $D(N^r)$ if r is bounded. The increase in power comes at reasonable cost.

7.4 Discussions and Conclusions

In this chapter we have completed the study in depth of three new homogeneous product networks. In Table 7.1 we have compiled their structural and VLSI complexity properties, as well as the time complexity of running the algorithms presented for homogeneous product networks.

Proof: The case for $r = 2$ was shown in [61]. For higher dimensions, since $PD_r(N) = D(N) \otimes PD_{r-1}(N)$, a vertex of $PD_r(N)$ can be written as $u = u_{rn-1}u_{rn-2}...u_{(r-1)n}|S'$, where S' is a vertex in $PD_{r-1}(N)$. For the discussion below, we are only interested in the leftmost bit in S', so we can write it as $S' = sS$. That is,

$$u = u_{rn-1}u_{rn-2}...u_{(r-1)n}|sS.$$

In $D(N^r)$, the outgoing edges are to

$$v = u_{rn-2}...u_{(r-1)n}s|Su_{rn-1}$$

and

$$w = u_{rn-2}...u_{(r-1)n}s|S\overline{u}_{rn-1}.$$

For $PD_r(N)$, u has two neighbors at the highest dimension, x and y, where

$$x = u_{rn-2}...u_{(r-1)n}u_{rn-1}|sS$$

and

$$y = u_{rn-2}...u_{(r-1)n}\overline{u}_{rn-1}|sS.$$

Let x_ℓ denote the leftmost $(\log N)$-bit substring of x (i.e. the part to the left of "|".) Then, observe that

$$v_\ell = w_\ell = \begin{cases} x_\ell & \text{if } u_{rn-1} = s, \\ y_\ell & \text{otherwise} \end{cases}$$

This means that, by following one of the outgoing edges from u at the highest dimension, we correct the leftmost $\log N$ bits of the address towards v. Since the next set of $\log N$ bits can be corrected by the same method as above, $r - 1$ additional steps are needed to reach v.

We can study now the congestion of the embedding. Note that the first edge of the path from u to v and the first edge of the path from u to w are the same, since depending on s the correction of the leftmost $\log N$ bits of u takes both paths to either x or y. Furthermore, the paths from u to v and form u to w share all the edges but the last one, where the rightmost $\log N$ bits are corrected.

Similarly, there exist edges in $D(N^r)$ from the node

$$u' = \overline{u}_{rn-1}u_{rn-2}...u_{(r-1)n}|sS$$

to v and w. The paths in $PD_r(N)$ from u' to v and from u' to w have a common first edge, that depending on s takes the paths to either x or y. From there they share all the edges but the last one. The paths from u to v and from u' to v share all the edges but the first one, and same thing happens with the paths from u to w and from u' to w.

Proof: The proof is very simple. It has been shown in [7] that any embedding of the grid $PL_r(N)$, for $r > 1$, into the similar-size shuffle-exchange requires dilation $\Omega(\log\log N^r)$. Since this grid is a subgraph of $PS_r(N)$, the claim follows. ■

The results presented for $PS_r(N)$ show that the network is more powerful than its factor network. Observe that, if the number of dimensions is bounded, their VLSI layout area complexity is asymptotically the same.

7.3 Products of de Bruijn Graphs

We finally study in this section the product of de Bruijn graphs, denoted $PD_r(N)$. Comparing to the product networks in the previous sections, the vertex degree of this network increases by 25%, while $PD_r(N)$ has better properties in other respects. Diameter reduces by 50%, and the minimum number of parallel paths between an arbitrary pair of vertices doubles. It also has better embedding properties as is shown below.

It is well known that shuffle-exchange and de Bruijn networks are computationally equivalent. That is, every computation which can be performed on one of them, can be also performed on the other with constant slowdown. We presented in Corollary 4.6 that this is also true for their respective product versions. However, $PD_r(N)$ presents better dilation and congestion in most of the embeddings.

We first present a whole family of tori as subgraphs of $PD_r(N)$.

Theorem 7.11 *For all $k \leq N$, $PR_r(k)$ is a subgraph of $PD_r(N)$.*

Proof: Due to Theorem 4.1, it suffices to note that the de Bruijn network is pancyclic [83], *i.e.* for every value of $k \leq N$, $D(N)$ contains a cycle of length k. ■

In Section 4.2 we have presented the fact that $PT_r(N)$ is a subgraph of $PD_r(N)$. Therefore, the following corollary is immediate.

Corollary 7.2 *The r-dimensional mesh of $(N-1)$-node trees is a subgraph of $PD_r(N)$.*

Observe that this embedding is much better than the embedding of the 2-dimensional mesh of trees into the pure de Bruijn graph presented in [68], which presents dilation of 2, congestion of 8, and load of 2.

The next two results show that $PD_r(N)$ is more powerful than the de Bruijn graph $D(N^r)$.

Theorem 7.12 *$D(N^r)$ can be embedded onto $PD_r(N)$ with dilation r, and congestion 2 when $r = 2$, or congestion 4 when $r > 2$.*

above.) Then, observe that

$$v_\ell = \begin{cases} x_\ell^s & \text{if } u_{rn-1} = s, \\ x_\ell^e & \text{otherwise} \end{cases}$$

That is, x^e is at a distance of two from u, and going from u to x^e corrects just the leftmost $\log N$ bits of the address towards v. Since the next set of $\log N$ bits can be corrected by the same method as above, $2(r-1)$ additional steps are needed to reach v.

To study the congestion of the described embedding we take two vertices of $S(N^r)$, where the leftmost $\log N$ bits and the rightmost $\log N$ bits are explicitly shown, and $S' = sS$ is a vertex in $PS_{r-2}(N)$, as

$$u = u_{rn-1}u_{rn-2}...u_{(r-1)n}|sS|u_{n-1}u_{n-2}...u_0$$

and

$$u' = \overline{u}_{rn-1}u_{rn-2}...u_{(r-1)n}|sS|u_{n-1}u_{n-2}...u_0$$

whose respective shuffle neighbors are

$$v = u_{rn-2}...u_{(r-1)n}s|Su_{n-1}|u_{n-2}...u_0u_{rn-1}$$

and

$$v' = u_{rn-2}...u_{(r-1)n}s|Su_{n-1}|u_{n-2}...u_0\overline{u}_{rn-1}$$

The paths in $PS_r(N)$ for the edges (u, v) and (u', v') meet at the node

$$u_{rn-2}...u_{(r-1)n}s|sS|u_{n-1}u_{n-2}...u_0$$

after the leftmost $\log N$ bits have been corrected (by traversing one or two edges, depending on whether $u_{rn-1} = s$.) From there, both paths share the same edges until the node

$$u_{rn-2}...u_{(r-1)n}s|Su_{n-1}|u_{n-2}...u_0u_{n-1}$$

is reached. Since no other paths contain these edges the congestion in the edges traversed until this point is at most 2.

Let assume now, without loss of generality, that $u_{n-1} = u_{rn-1}$. Then, the vertex reached by the paths is v and the edge (u, v) has been completely mapped. The path from u' to v' still needs to traverse an exchange edge to invert its rightmost bit. The path only shares this edge with the exchange edge (v, v') in $S(N^r)$ and then, the congestion of the edge is 2.

Hence, the congestion of the embedding is 2 and the proof is complete. ■

Theorem 7.10 *Any embedding of $PS_r(N)$ into $S(N^r)$ requires dilation $\Omega(\log(r \log N))$.*

and

$$w^{s,c} = u_{2n-2}...u_{n+1}u_n u_{2n-1}|u_{n-1}u_{n-2}...u_1u_0$$

In the following discussion, subscripts "ℓ" and "r" are used to denote the left-hand half of a label, and the right-hand half of a label. For example, $w_\ell^{s,c}$ denotes the left-hand half of the vertex $w^{s,c}$ above. There are two cases to consider:

Case 1: $u_{2n-1} = u_{n-1}$. In this case the reader can easily verify that $v = w_\ell^{s,c}|w_r^{s,r}$. This means that one can go from u to v in $PS_2(N)$ in two steps: by moving to the shuffle neighbor of u in the column and then to the shuffle neighbor in the row. Alternatively, one can move to the shuffle neighbor in the row first, and then in the column.

Case 2: $u_{2n-1} \neq u_{n-1}$. Then, given u and v as above, the left-hand half and the right-hand half of v can be computed as:

$$v_\ell = w_\ell^{s,c} + w_\ell^{e,c}$$

and

$$v_r = w_r^{s,r} + w_r^{e,r}$$

where the "+" sign denotes sequencing of the two moves. That is, $w_r^{s,c} + w_r^{e,c}$ denotes moving to the shuffle neighbor in the column, followed by moving to the exchange neighbor in the column. Since $v = v_\ell|v_r$, a sequence of four moves yields the desired vertex label.

To extend these arguments for $r > 2$, since $PS_r(N) = S(N) \otimes PS_{r-1}(N)$, a vertex of $PS_r(N)$ can be written as $u = u_{rn-1}u_{rn-2}...u_{(r-1)n}|S'$, where S' is a vertex in $PS_{r-1}(N)$. For the discussion below, only the leftmost bit of S' is relevant, so we can write $S' = sS$. That is,

$$u = u_{rn-1}u_{rn-2}...u_{(r-1)n}|sS.$$

In $S(N^r)$, the shuffle neighbor is

$$v = u_{rn-2}...u_{(r-1)n}s|Su_{rn-1}$$

For the product network, u has a shuffle neighbor x^s, where

$$x^s = u_{rn-2}...u_{(r-1)n}u_{rn-1}|sS$$

which in turn has an exchange neighbor x^e, where

$$x^e = u_{rn-2}...u_{(r-1)n}\overline{u}_{rn-1}|sS$$

Let x_ℓ denote the leftmost $\log N$-bit substring of x (i.e. the part to the left of "|"

shuffle-exchange graph. However the resulting embedding has larger load, dilation, and congestion than the one presented here into $PS_r(N)$.

The next two results consider the embedding of the shuffle-exchange graph into its product version, and the reverse embedding of the product network into the pure shuffle-exchange graph.

Theorem 7.9 *$S(N^r)$ can be embedded onto $PS_r(N)$ with dilation $2r$ and congestion 2.*

Proof: First consider the case for $r = 2$. Both $S(N^2)$ and $PS_2(N)$ are labeled by $(2 \log N)$-bit strings. For the product graph, the rightmost $\log N$ bits determine the "row address," while the leftmost $\log N$ bits determine the "column address." We show that whenever (u, v) is an exchange edge in $S(N^2)$, it is also an exchange edge in $PS_2(N)$. Alternatively, whenever (u, v) is a shuffle edge in $S(N^2)$, there is a path of length at most 4 from u to v in $PS_2(N)$.

Consider the vertex:

$$u = u_{2n-1}u_{2n-2}...u_{n+1}u_n|u_{n-1}u_{n-2}...u_1u_0$$

Here "|" separates the left-hand half of the label from the right-hand half. We use directed edges as we did in Definition 2.10 to simplify the proof. Then, it suffices to focus on the outgoing edges only. If v is the exchange neighbor of u in $S(N^2)$, then

$$v = u_{2n-1}u_{2n-2}...u_{n+1}u_n|u_{n-1}u_{n-2}...u_1\overline{u}_0$$

In the $PS_2(N)$ graph, u has an exchange neighbor w in its row, whose address is obtained by complementing the rightmost bit of the address. Clearly, $w = v$. In fact, it is true for arbitrary r that whenever (u, v) is an exchange edge of $S(N^r)$, it is also an exchange edge of $PS_r(N)$. Therefore, the rest of this proof only needs to consider the shuffle edges.

Now suppose (u, v) is a shuffle edge in $S(N^2)$. If u is as above, v must be:

$$v = u_{2n-2}...u_{n+1}u_nu_{n-1}|u_{n-2}...u_1u_0u_{2n-1}.$$

For $PS_2(N)$, the row neighbors of u are

$$w^{e,r} = u_{2n-1}u_{2n-2}...u_{n+1}u_n|u_{n-1}u_{n-2}...u_1\overline{u}_0$$

and

$$w^{s,r} = u_{2n-1}u_{2n-2}...u_{n+1}u_n|u_{n-2}...u_1u_0u_{n-1}$$

where the superscripts "e, s, r" stand for "exchange," "shuffle," and "row," respectively. The column neighbors of u, indicated by the superscript "c," are

$$w^{e,c} = u_{2n-1}u_{2n-2}...u_{n+1}\overline{u}_n|u_{n-1}u_{n-2}...u_1u_0$$

left subtree. ■

It can be easily shown that this new product network has the same VLSI layout complexity as the original $PT_r(N)$. Therefore it seems to be even more interesting than the original network.

7.2 Products of Shuffle-Exchange Graphs

The second network studied in this chapter is the product of shuffle-exchange graphs, denoted $PS_r(N)$. We have observed many interesting properties of this network. This network seems to be more powerful than the pure shuffle-exchange graph from several points of view. Its connectivity is larger, as well as its bisection width. This network can emulate the grid with constant dilation, while any embedding of the grid into the shuffle-exchange of similar size requires unbounded dilation.

Here we present several results that further show the power of this network, to finally prove that it is even more powerful than the pure shuffle-exchange.

We first show that products of binary trees can be embedded in the products of shuffle-exchange graphs with dilation 2 and congestion 2. While this result carries all the embedding properties of $PT_r(N)$ to the $PS_r(N)$ graph, it may be better to find direct embeddings for some cases. Next, it is shown that the shuffle-exchange graph $S(N^r)$ can be embedded onto $PS_r(N)$ with dilation $2r$ and congestion 2. For an implementation with a fixed number of dimensions, this embedding can be considered of constant dilation, particularly because N can grow independently from r. Moreover, it is shown that $PS_r(N)$ cannot be embedded onto $S(N^r)$ with less than logarithmic dilation. This makes the product network more powerful than the shuffle-exchange network itself.

Theorem 7.8 *$PT_r(N-1)$ can be embedded into $PS_r(N)$ with dilation 2 and congestion 2.*

Proof: Due to corollaries 4.1 and 4.2, it suffices to show that $T(N-1)$ can be embedded into $S(N)$ with dilation 2 and congestion 2. The labeling of the complete binary tree described in Definition 2.8 and Figure 2.5 induces the desired embedding. ■

The following result is now immediately observed.

Corollary 7.1 *As in Theorem 7.3, a hierarchy of meshes of trees can be embedded into $PS_r(N)$ with dilation 2 and congestion 2.*

There are known efficient embeddings of the 2-dimensional mesh of trees into the shuffle exchange, since we can apply the embedding of the mesh of trees into the de Bruijn graph presented by Schwabe [68] and then embed the de Bruijn graph onto the

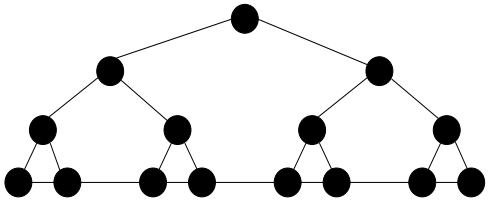

Figure 7.3: Extending the complete binary tree by connecting the leaves.

Finally, we present a simple but very interesting extension of the $PT_r(N)$ network which contains the torus as a subgraph.

Consider connecting the leaves of the complete binary tree as shown in Figure 7.3. We denote the resulting graph as $X(N)$ which is a subgraph of the X-tree graph [21]. In a modular implementation, all the nodes of a tree could be designed with the same number of I/O channels, and the unused channels at the leaves could be used to connect the leaves in this fashion. Moreover, the extra channels at the roots can be used for I/O with the external world.

If we construct the product of these trees, denoted $PX_r(N)$, the resulting network has the power of the $PT_r(N)$. The next result shows that it also contains the torus (and hence the grid) as a subgraph.

Theorem 7.7 *$PX_r(N)$ contains $PR_r(N)$ as a subgraph.*

Proof: We show that $X(N)$ contains a hamiltonian cycle. The claim then directly follows from Theorem 4.1.

We first show that $X(N)$ contains the following hamiltonian paths

LL-path: A path from the leftmost leaf to the rightmost leaf.

LR-path: A path from the leftmost leaf to the root.

Note that $X(N)$ is symmetric and a LR-path can be converted into a path from the rightmost leaf to the root (symmetric LR-path.)

We proceed by induction on the number of levels in $X(N)$. If we have two levels, $X(3)$ is just a triangle and the above paths are contained in it. Therefore assume that these paths exist in the h-level tree, $X(2^h - 1)$, where $h > 1$.

The LL-path for the $(h+1)$-level tree $X(2^{h+1} - 1)$ is obtained as the LR-path in the left subtree of the root, followed by the root, followed by the symmetric LR-path in the right subtree.

The LR-path for the $(h+1)$-level tree $X(2^{h+1} - 1)$ is obtained as the LL-path in the left subtree of the root, followed by the LR-path in the right subtree, followed by the root.

The hamiltonian cycle for any tree $X(N)$ is, then, obtained as the LR-path in the right subtree of the root, followed by the root, followed by the symmetric LR-path in the

dilation and constant congestion. The following theorem presents this fact (the proof can be found in the Appendix.)

Theorem 7.5 *The complete binary tree of* $r\log(N+1)-\lfloor\frac{r}{2}\rfloor$ *levels can be embedded into* $PT_r(N)$*, where* $N>3$*, with dilation 3 and congestion 3.*

The complete binary tree that the above theorem embeds in $PT_r(N)$ is the largest possible for $r \leq 3$ and very close to the largest (when not the largest) for small values of r. For instance, $PT_9(7)$ has enough nodes to contain a 25-level complete binary tree and the above theorem embeds a 23-level tree into it.

The case $N = 3$ is not considered in Theorem 7.5 although it is specially interesting because $PT_r(3)$ is isomorphic to the grid $PL_r(3)$. Theorem 7.4 allows to obtain a complete binary tree subgraph of $PT_r(3)$ that is the largest possible for $r \leq 3$. For larger values of r it is possible to apply an approach similar to the one used in Theorem 7.5.

Observe that, if the number of dimensions is bounded, the above embeddings have bounded expansion.

The next result shows that complete binary tree cannot emulate its comparable-size product network with less than logarithmic dilation.

Theorem 7.6 *Any embedding of* $PT_r(N)$ *into the large-enough complete binary tree requires dilation* $\Omega(\log(r\log\log N))$.

Proof: To prove the claim we show that $PT_r(N)$ contains a subgraph, G_1, and that there exists a supergraph of the complete binary tree, G_2, such that any embedding of G_1 into G_2 requires the claimed amount of dilation.

G_1 is the r-dimensional grid. Since $T(N)$ contains a path of length $M = 2(\log(N+1)-1)$, $PT_r(N)$ contains $PL_r(N)$ as a subgraph. We can select G_2 as the de Bruijn graph since it contains the complete binary tree as a subgraph. It is shown in [7] that any embedding of $PL_r(N)$ (for $r > 1$) into the de Bruijn graph requires a dilation of at least $\Omega(\log\log M^r)$. That gives the claimed result. ■

We have also presented in Chapter 5 several algorithms that perform efficiently in the product of complete binary trees. Some of them give better performance that the mesh of trees for problems specially suited for this last network. This presents the $PT_r(N)$ network as a very interesting candidate to take over the position of the mesh of trees between the interconnection networks.

One last fact will make this assertion stronger: both networks present same VLSI layout area complexity. We obtained the bounds for the $PG_r(N)$ in Section 6.4. They can be compared with the bounds for the mesh of trees obtained in [41]. This fact implies that the increase in area necessary to create a network more powerful than the mesh of trees is bounded.

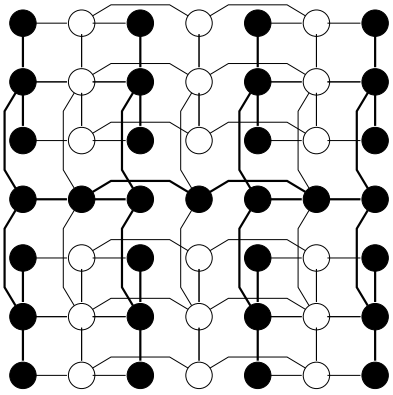

Figure 7.2: Embedding of the complete binary tree into the two-dimensional product of complete binary trees.

the construction of the two-dimensional mesh of trees, *i.e.* all the colored nodes, red or blue, are contained in the two-dimensional mesh of trees. For induction, assume that the $(r-1)$-dimensional mesh of trees has been already colored in the $PT_{r-1}(N)$ graph. The coloring rule for the r-dimensional mesh of trees is the same. That is, when going from $PT_{r-1}(N)$ to $PT_r(N)$, color the internal nodes of a dimension-r tree in blue if and only if it has red leaves. The set of vertices colored red or blue gives the largest r-dimensional mesh of trees contained in $PT_r(N)$.

Once the largest mesh of trees is colored, successively smaller meshes of trees are then obtained by removing all the colored vertices and coloring the remaining vertices by the same strategy. ■

Observe that the largest mesh of trees subgraph of $PT_r(N)$ has $\Theta(N^r)$ nodes. The expansion of this embedding is, hence, constant.

The next results show that $PT_r(N)$ is strictly more powerful than the similar-size complete binary tree.

Theorem 7.4 *The complete binary tree of* $r(\log(N+1)-1)+1$ *levels is a subgraph of* $PT_r(N)$.

Proof: For $r = 2$, the embedding of 5-level complete binary tree into $PT_2(7)$ is shown in Figure 7.2. Note, in particular, that the tree in the middle row constitutes the highest 3 levels of the tree. The leaves of this row tree correspond to the roots of column trees. This pattern can be recursively repeated for larger values of N in two dimensions. Assuming that the claim is true for $PT_{r-1}(N)$, the embedding proof for r dimensions follows from the recursive construction of $PT_r(N)$. ■

Note that, for $r = 2$, the tree embedded by the above method is the largest tree possible. In general, for larger values of r, larger trees can be embedded with constant

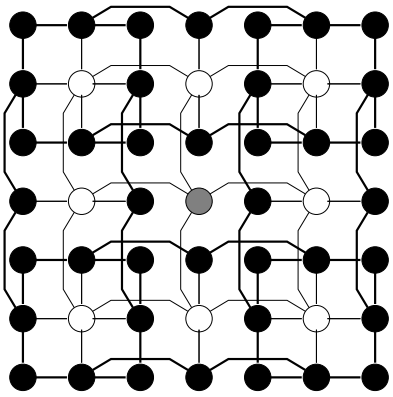

Figure 7.1: Embedding meshes of trees into products of complete binary trees.

product of complete binary trees can emulate the torus (and, hence, the grid) very efficiently. The following result shows that this network is in fact more powerful than the grid, by presenting an optimal-dilation embedding of $PT_r(N)$ onto the grid.

Theorem 7.2 *The optimal dilation of embedding $PT_r(N)$ into $PL_r(N)$ is $\lceil \frac{N-1}{2\log(N+1)-2} \rceil$.*

Proof: Section 3 in [32] presents an embedding of the complete binary tree $T(N)$ onto the linear array $L(N)$ with dilation cost $\lceil \frac{N-1}{2\log(N+1)-2} \rceil$. The dilation of this embedding is optimal as it matches the trivial lower bound obtained by comparing the respective diameters of both networks. Then by the simple application of Corollary 4.1 the claimed embedding is obtained. Optimality of the dilation follows from the trivial lower bound obtained by comparing the diameters of the networks. ■

We show now that meshes of trees of comparable size are subgraphs of $PT_r(N)$. In fact, the $PT_r(N)$ graph contains not just one mesh of trees, but a hierarchy of meshes of trees as shown next.

Theorem 7.3 *For all $i = 1, ..., \log(N+1)$, $PT_r(N)$ contains the mesh of $(N+1)/2^i$-leaf trees as a subgraph.*

Proof: Figure 7.1 shows the two-dimensional meshes of trees contained in $PT_2(7)$. Note that in this figure there are three meshes of trees contained, one with $(N+1)/2 = 4$ leaves for each tree (shown in dark nodes), one with $(N+1)/4 = 2$ leaves for each tree (shown in empty nodes), and one degenerate with $(N+1)/8 = 1$ leaf (the central node.) In general, the largest mesh of trees contained in $PT_r(N)$ is obtained as follows. Start with the $PT_2(N)$ graph and select the uth dimension-1 trees such that u is a leaf. For those trees selected, color the leaves in red and the internal nodes in blue. Do not color the non-selected trees. For the dimension-2 trees, color the internal nodes of a tree in blue if and only if it has red leaves. Note that this coloring scheme is consistent with

Chapter 7

Interesting Product Networks

In the previous chapters we have derived many general results for homogeneous product networks. In each chapter, the results obtained have been applied to several instances of product networks to show the results' power.

In this chapter we concentrate on three of these specific instances of homogeneous product networks and we propose them as new interconnection networks. The three networks to be covered are the product of complete binary trees, $PT_r(N)$, the product of shuffle-exchange graphs, $PS_r(N)$, and the product of de Bruijn graphs, $PD_r(N)$. On top of the capabilities of these networks presented in previous chapters, we compare them here with their respective factor network and with other popular non-product networks. We show that all of them are very powerful networks and very interesting candidates for their use as interconnection networks.

7.1 Products of Complete Binary Trees

In previous chapters we have covered different aspects of the product of complete binary trees. We have obtained that it has logarithmic diameter, large bisection width, and bounded vertex degree when the number of dimensions is bounded. We have seen also that it has connectivity of r and many ways to be partitioned.

The first result of this section shows that the number of vertex-disjoint paths between any two nodes in this network is larger, in some cases, than the lower bound defined by its connectivity. The proof of this theorem is presented in the Appendix.

Theorem 7.1 *Every pair of vertices in $PT_r(N)$, where $r > 1$, is connected by exactly m vertex-disjoint paths, where m is the minimum vertex degree of the vertices in the pair.*

Despite its simple structure, the product of complete binary trees have very interesting embedding properties. For instance, while tori and meshes of trees are powerful architectures, they have different strengths and weaknesses. We have shown that the

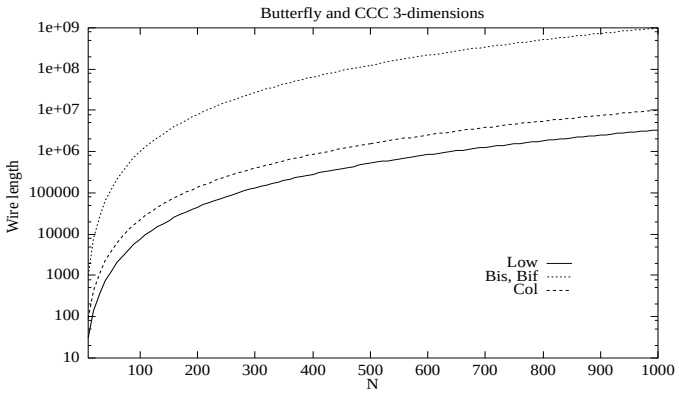

Figure 6.11: Comparison of the maximum wire length bounds obtained for $PB_3(N)$ and $PC_3(N)$.

by only a polylogarithmic function of N.

From Table 6.3 and figures 6.8 to 6.11, only if the number of dimensions r is bounded the collinear method obtains bounds that match the lower bounds. In most cases it gave better bounds than the other two approaches. The only exception we have is the product of complete binary trees. When applicable, the use of bisectors seems to give same maximum wire lengths as the use of bifurcators.

The above analyses suggest the method based on collinear layouts as a very useful and powerful approach to the layout problem for homogeneous product networks. More research may help in finding normal collinear layouts with small wiring width and small bandwidth for a variety of factor graphs.

Clearly, it is still necessary to study how these results can be extended to obtain layouts for heterogeneous product networks. If the heterogeneous product network is obtained from same-size factor graphs it is not difficult to derive bounds similar to those presented by just considering the worst case. For instance, the lower bounds presented in theorems 6.5 and 6.6 are still valid if we define C as the maximum of the maximal congestions of the factor graphs. Similarly, if f is the largest asymptotic complexity bisector of all the factor graphs, then the product graph has a $O(x^{(r-1)/r} f(x^{1/r}))$-bisector. The results for bifurcators and collinear layouts can be generalized in a similar way.

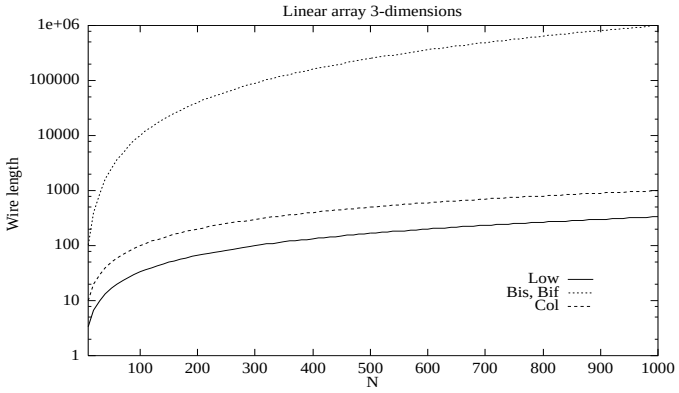

Figure 6.8: Comparison of the maximum wire length bounds obtained for $PL_3(N)$.

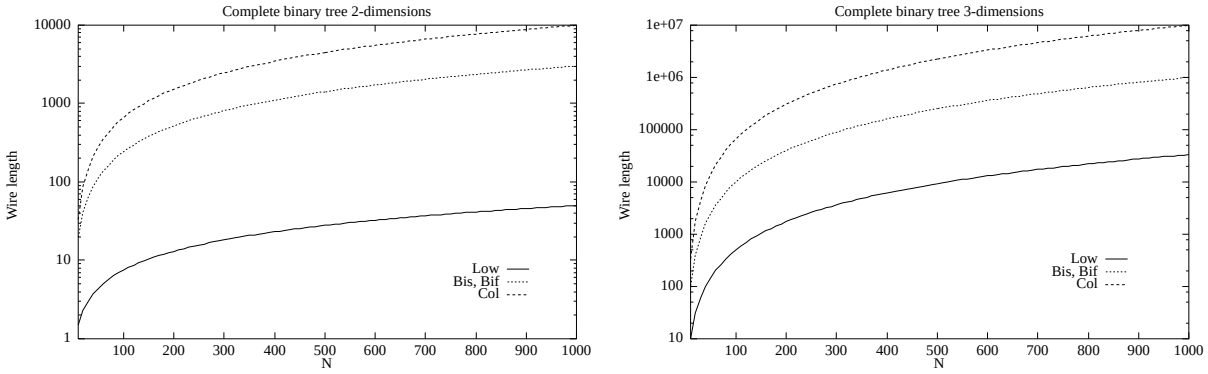

Figure 6.9: Comparison of the maximum wire length bounds obtained for $PT_2(N)$ and $PT_3(N)$, respectively.

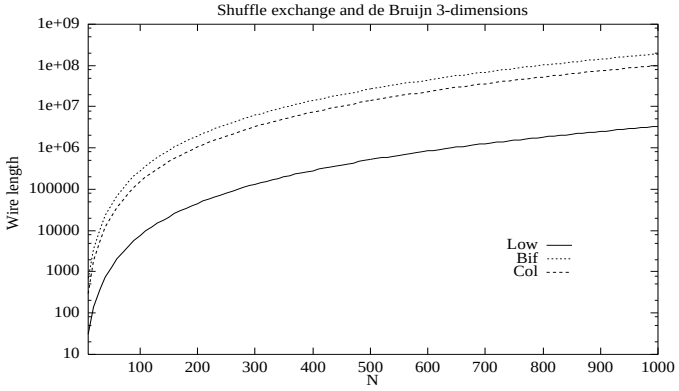

Figure 6.10: Comparison of the maximum wire length bounds obtained for $PS_3(N)$ and $PD_3(N)$.

on the layout area in Table 6.2. We do not know the bandwidth of these layouts to obtain a bound on the wire length. Instead, we use the wiring width and bandwidth of a collinear layout presented in [78] for the shuffle-exchange graph, which has wiring width $O(N/\log^{1/2} N)$ and bandwidth $O(N/\log^{1/2} N)$. Note, then, that the maximum wire length bounds presented in Table 6.3 for these networks may not be achievable with the optimal area layout presented in Table 6.2.

The normal collinear layout obtained by placing the levels of the butterfly in order one after the other has wiring width $O(N/\log N)$ and bandwidth $O(N/\log N)$. A similar approach can be used for the cube-connected cycles to obtain the same bounds. The hypercube has, as factor network, the 2-node linear array which is laid out with wiring width 1 and bandwidth 1 (see Figure 6.4.(a).) The Petersen graph can be laid down in a normal collinear layout with both wiring width and bandwidth $O(1)$. Also, it is possible to obtain a normal collinear layout for $K(N)$ with wiring width $O(N^2)$.

In figures 6.5 to 6.11 we plot these bounds in a graphical form for those networks considered more interesting. In these figures the x axis shows the value of N (the number of nodes in the factor graph) while the y axis shows the value of the bounds on a logarithmic scale, with all the constant factors neglected (we ignore the Ω and the O.) The curves have been labeled with "Low" (lower bound), "Bis" (upper bound obtained by using bisectors), "Bif" (upper bound obtained by using bifurcators), and "Col" (upper bound obtained by using collinear layouts.) In the figures we fix the value of r to 3, representing also the value $r = 2$ when interesting. For larger values of r the shape of the figures will remain practically the same, since the difference between bounds is a function of N. The area bounds for the linear array are not plotted for 2 dimensions for the triviality of the layout in this case. For more dimensions all the approaches yield area-optimal layouts and the plot is not interesting. The bounds are not plotted neither for the hypercube nor for the product of Petersen graphs, since N is fixed for both networks. Also, we did not plot the case for products of complete graphs since there is only one upper bound result. For hypercubes and products of Petersen and complete graphs the area of collinear layouts are optimal, but in maximum wire length they differ from the lower bound by a factor of r.

From the results presented in Table 6.2 and figures 6.5 to 6.7, the proposed method based on collinear layouts seems to generate layouts with optimum area in most of the cases. Only for products of complete binary trees the layout area is not minimum, and it is not possible to reach an optimal area layout for this network using this method, since we would need a normal collinear layout for the complete binary tree with constant wiring width. The layouts obtained by using bisectors (when applicable) are also quite area-efficient, since they have optimal area for more than two dimensions in the studied cases (see figures 6.5 and 6.7.) In fact, the layout obtained for the product of complete binary trees is also optimal for 2 dimensions since, as we see in Chapter 7, this network has the mesh of trees as a subgraph, which requires area $\Omega(N^2 \log^2 N)$ for two dimensions [41]. The layouts obtained by using bifurcators are not always area-optimal, but are off

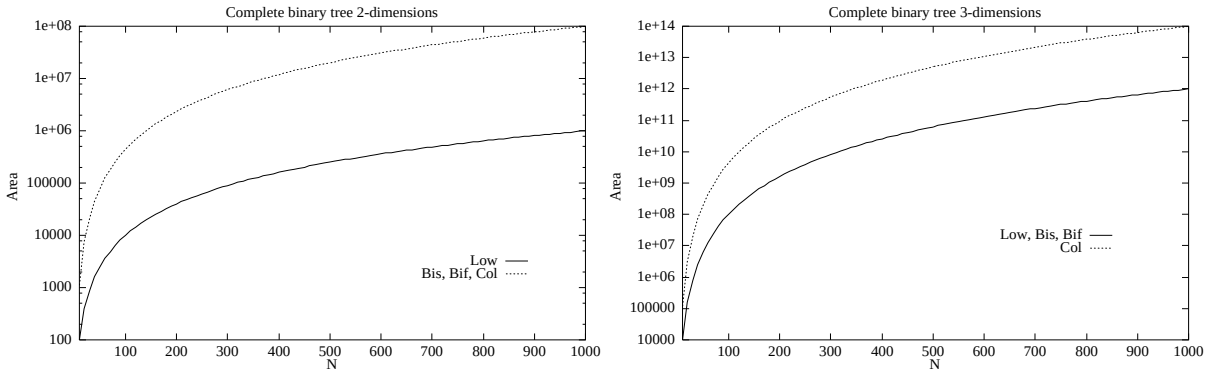

Figure 6.5: Comparison of the area bounds obtained for $PT_2(N)$ and $PT_3(N)$, respectively.

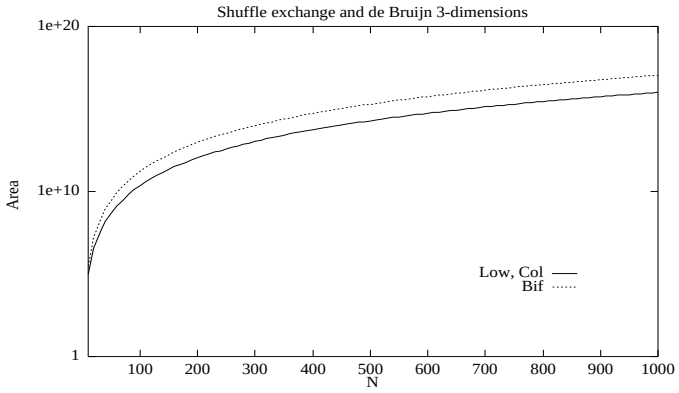

Figure 6.6: Comparison of the area bounds obtained for $PS_3(N)$ and $PD_3(N)$.

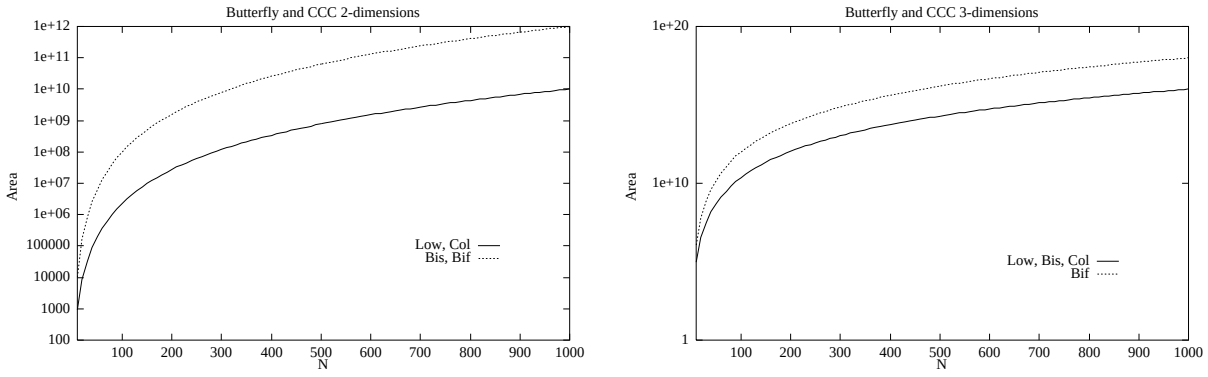

Figure 6.7: Comparison of the area bounds obtained for $PB_2(N)$ and $PC_2(N)$, and $PB_2(N)$ and $PC_2(N)$, respectively.

6.4 Application to Specific Networks

In tables 6.2 and 6.3 we have compiled the bounds obtained by applying the presented results to several networks. In both tables UN stands for "unknown" and N.A. stands for "not applicable." Table 6.2 presents the bounds on layout area. The upper bounds marked in this table with "*" are optimal. Table 6.3 presents the bounds on maximum wire length. The upper bounds marked in this table with "*" are optimal if r is bounded. We first present how these bounds have been obtained.

The second column in both tables presents the lower bounds obtained by direct application of theorems 6.5 and 6.6. The value of the maximal congestion for all the factor networks was already obtained in Section 3.3.

The third and fourth columns of the tables present upper bounds obtained from bisectors of the factor graphs. It is easy to observe that the linear array and the complete binary tree have O(1)-bisectors. By applying Corollary 6.1 the presented bounds are directly obtained. There have not been found, as far as we know, tight bisectors for the shuffle-exchange and the de Bruijn graphs. Thus we present the corresponding bounds as unknown. We can easily show that the $(n2^n)$-node butterfly can be bisected by removing $O(2^n)$ edges, resulting in two butterflies with one less level and several isolated nodes. Therefore we conclude that the butterfly has a $O(x/\log x)$-bisector. Similarly it can be shown that the cube-connected cycles has a $O(x/\log x)$-bisector. To obtain the bounds on wire length we use $x/\log x = x^{\alpha}$ for some $\alpha > 0$ and, therefore, $\alpha > 0$ in Corollary 6.1 for both networks. Since the hypercube and the product of Petersen graphs can only grow by increasing the number of dimensions, they are considered here as networks with unbounded number of dimensions, and the bisector approach can not be applied to them. Similarly, this approach can not be applied to the product of complete graphs since $K(N)$ has not bounded vertex degree.

The fifth and sixth columns contain the bounds obtained from bifurcators of the factor networks. The linear array and the complete binary tree have 0-special bifurcators. The value of the bifurcators for the shuffle-exchange and de Bruijn networks are obtained from known layouts of area $O(N^2/\log^2 N)$ [41], that implies the existence of $O(N/\log N)$-bifurcators for these networks [9]. It is easy to see that the butterfly and the cube-connected cycles have 1-special bifurcators. We then apply corollaries 6.2 and 6.3 to obtain the bounds on layout area and maximum wire length for all these networks. Again, the hypercube and the product of complete and Petersen graphs are not considered.

The last column of the tables present the upper bounds obtained from collinear layouts for the factor networks. If the nodes of the linear array are laid down in a line we obtain a collinear layout with wiring width 1 and bandwidth 1. The complete binary tree has a collinear layout with wiring width $O(\log N)$ and bandwidth $O(N)$, which can be obtained by just labeling the nodes in in-order. For the shuffle-exchange and de Bruijn graphs we can apply lemmas 6.3 and 6.4 to their optimal $O(N/\log N) \times O(N/\log N)$ area layouts [41] to obtain normal collinear layouts with wiring width $O(N/\log N)$, hence the bounds

$G(N)$	Area	Max. Wire Length	Condition
C: max. congestion d: diameter	$\Omega(N^{2(r+1)}/C^2)$	$\Omega(N^{r+1}/Crd)$	
$f(x)$-bisector	$O(N^2 f^2(N)\log^2 N)$		$r=2$
	$O(N^{2(r-1)}f^2(N))$		$r>2$
$O(x^\alpha)$-bisector	$O(N^2\log^2 N)$	$O(N\log N/\log\log N)$	$\alpha=0$ and $r=2$
	$O(N^{2(r+\alpha-1)})$	$O(N^{r+\alpha-1})$	Otherwise
F-bifurcator	$O(N^{2(r-1)}F^2\log^2(N/F))$	$O(N^{r-1}F\frac{\log(N/F)}{\log\log(N/6(2+\sqrt{2})F)})$	
α-special bifurcator	$O(N^2\log^2 N)$	$O(N\log N/\log\log N)$	$\alpha=0$ and $r=2$
	$O(N^{2(r+\alpha-1)})$	$O(N^{r+\alpha-1})$	Otherwise
w: wiring width b: bandwidth	$O(w^2N^{2(r-1)})$	$O(bwN^{r-2})$	

Table 6.1: Results on VLSI layout complexity obtained.

Factor network	Lower bounds	Upper bound for the area of product network				
		Bisector ($r=2$)	Bisector ($r>2$)	Bifurcator ($r=2$)	Bifurcator ($r>2$)	Collinear
$L(N)$	$\Omega(N^{2(r-1)})$	$O(N^2\log^2 N)$	$O(N^{2(r-1)})$ *	$O(N^2\log^2 N)$	$O(N^{2(r-1)})$ *	$O(N^{2(r-1)})$ *
$T(N)$	$\Omega(N^{2(r-1)})$	$O(N^2\log^2 N)$ *	$O(N^{2(r-1)})$ *	$O(N^2\log^2 N)$ *	$O(N^{2(r-1)})$ *	$O(N^{2(r-1)}\log^2 N)$
$S(N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$	UN		$O(N^{2r}\frac{\log^2\log N}{\log^2 N})$		$O(N^{2r}/\log^2 N)$ *
$D(N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$	UN		$O(N^{2r}\frac{\log^2\log N}{\log^2 N})$		$O(N^{2r}/\log^2 N)$ *
$B(N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$	$O(N^4)$	$O(\frac{N^{2r}}{\log^2 N})$ *	$O(N^{2r})$		$O(N^{2r}/\log^2 N)$ *
$C(N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$	$O(N^4)$	$O(\frac{N^{2r}}{\log^2 N})$ *	$O(N^{2r})$		$O(N^{2r}/\log^2 N)$ *
Q_1	$\Omega(2^{2(r-1)})$	N.A.		N.A.		$O(2^{2(r-1)})$ *
$P(10)$	$\Omega(10^{2(r-1)})$	N.A.		N.A.		$O(10^{2(r-1)})$ *
$K(N)$	$\Omega(N^{2(r+1)})$	N.A.		N.A.		$O(N^{2(r+1)})$ *

Table 6.2: Bounds on the layout area obtained by application of the presented methods.

Factor network	Lower bounds	Upper bound for the wire length of product network				
		Bisector ($r=2$)	Bisector ($r>2$)	Bifurcator ($r=2$)	Bifurcator ($r>2$)	Collinear
$L(N)$	$\Omega(N^{r-2}/r)$	$O(N\frac{\log N}{\log\log N})$	$O(N^{r-1})$	$O(N\frac{\log N}{\log\log N})$	$O(N^{r-1})$	$O(N^{r-2})$ *
$T(N)$	$\Omega(\frac{N^{r-1}}{r\log N})$	$O(N\frac{\log N}{\log\log N})$	$O(N^{r-1})$	$O(N\frac{\log N}{\log\log N})$	$O(N^{r-1})$	$O(N^{r-1}\log N)$
$S(N)$	$\Omega(\frac{N^r}{r\log^2 N})$	UN		$O(N^r\frac{\log\log N}{\log N\log\log\log N})$		$O(N^r/\log N)$
$D(N)$	$\Omega(\frac{N^r}{r\log^2 N})$	UN		$O(N^r\frac{\log\log N}{\log N\log\log\log N})$		$O(N^r/\log N)$
$B(N)$	$\Omega(\frac{N^r}{r\log^2 N})$	$O(N^r)$		$O(N^r)$		$O(N^r/\log^2 N)$ *
$C(N)$	$\Omega(\frac{N^r}{r\log^2 N})$	$O(N^r)$		$O(N^r)$		$O(N^r/\log^2 N)$ *
Q_1	$\Omega(2^{r-2}/r)$	N.A.		N.A.		$O(2^{r-2})$ *
$P(10)$	$\Omega(10^{r-2}/r)$	N.A.		N.A.		$O(10^{r-2})$ *
$K(N)$	$\Omega(N^{r+1}/r)$	N.A.		N.A.		$O(N^{r+1})$ *

Table 6.3: Bounds on the wire length obtained by application of the presented methods.

enough connection points in each side of the node when needed. Figure 6.4.(b) presents this initial situation for our example graph.

For each row of nodes we apply the following iterative process. We start by creating w new rows above the row of nodes. The nodes in the row are divided in $N^{\lceil r/2 \rceil - 1}$ groups of N adjacent nodes each, and the nodes in each group are connected using the created rows with the wires laid down as specified by the normal collinear layout of $G(N)$. This completes the connections for the first dimension of the product graph. We subsequently create wN new rows, divide the nodes in a row into $N^{\lceil r/2 \rceil - 2}$ groups of N^2 adjacent nodes each, and use the wN new rows in groups of w each to connect N nodes of the second dimension. These nodes are N nodes apart from one another.

In the ith iteration we create wN^{i-1} new rows, divide the nodes in $N^{\lceil r/2 \rceil - i}$ groups of N^i nodes each, and connect sets of N nodes in the ith dimension, each N^{i-1} nodes apart from one another.

This process is applied $\lceil r/2 \rceil$ times for each row of nodes. The total number of wiring rows created is $w \sum_{i=0}^{\lceil r/2 \rceil - 1} N^i$. This is the distance between two rows of processors. Two adjacent processors in the same row are still touching each other. Figure 6.4.(c) presents the example layout after completion of the above process. To obtain this layout we applied the iterative step twice.

The same iterative process can be applied $\lfloor r/2 \rfloor$ times to connect the columns. As a result, we find that the columns of processors are at distance $w \sum_{i=0}^{\lfloor r/2 \rfloor - 1} N^i$. This completes the proof. Figure 6.4.(d) shows the final layout obtained for our example graph. ■

From this theorem we can obtain bounds on the area and maximum wire length for the layout.

Corollary 6.4 *If $G(N)$ has a normal collinear layout with wiring width w and bandwidth b, then $PG_r(N)$ can be laid out in an area of dimensions $\Theta(wN^{r-1}) \times \Theta(wN^{r-1})$ with maximum wire length $\Theta(bwN^{r-2})$.*

Proof: The length of the layout obtained from the above theorem along the horizontal dimension is $N^{\lceil r/2 \rceil}(\Delta \lceil r/2 \rceil + w \sum_{i=0}^{\lfloor r/2 \rfloor - 1} N^i)$. Since $w \geq \Delta/2$, $\sum_{i=0}^{\lfloor r/2 \rfloor - 1} N^i = \Theta(N^{\lfloor r/2 \rfloor - 1})$, and $N^{\lfloor r/2 \rfloor - 1} \geq \lceil r/2 \rceil$ for $N \geq 2$ and $r \geq 2$, then this length is $\Theta(wN^{r-1})$. The length along the vertical dimension is $N^{\lfloor r/2 \rfloor}(\Delta \lceil r/2 \rceil + w \sum_{i=0}^{\lceil r/2 \rceil - 1} N^i) = \Theta(wN^{r-1})$. Similarly, the length of the longest edge is at most $2w \sum_{i=0}^{\lceil r/2 \rceil - 1} N^i + b(N^{\lceil r/2 \rceil - 1}(\Delta \lceil r/2 \rceil + w \sum_{i=0}^{\lfloor r/2 \rfloor - 1} N^i)) = \Theta(bwN^{r-2})$. ■

In Table 6.1 we have compiled the results obtained in this chapter.

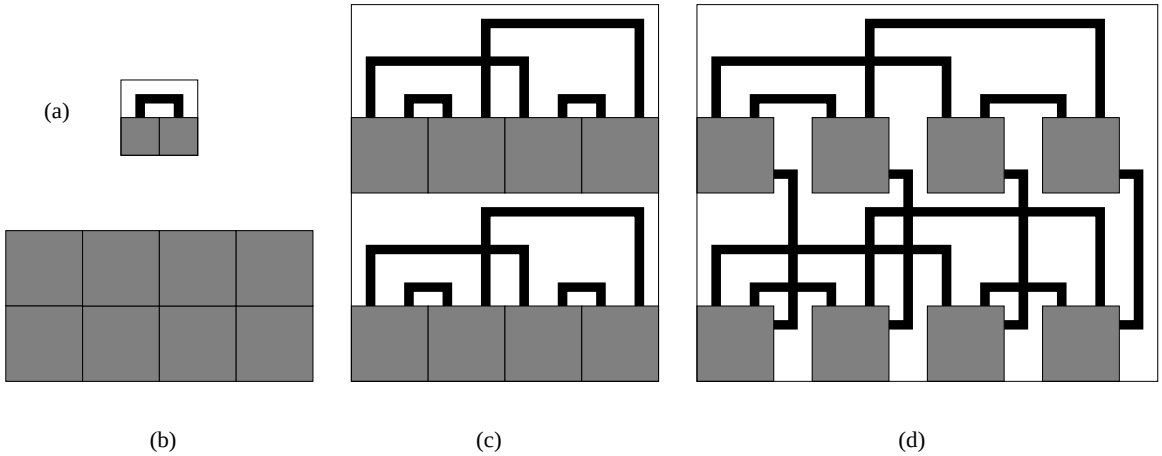

Figure 6.4: Layout for the 3-dimensional hypercube.

Since the N-node directed complete graph can be embedded onto the graph $G(N)$ with congestion C, it follows that any embedding of $G(N)$ onto $L(N)$ requires congestion at least $N^2/2C$ if N is even, or at least $(N^2-1)/2C$ if N is odd, since otherwise we could obtain an embedding of the directed complete graph with congestion smaller than its maximal congestion.

Since the congestion of any embedding of $G(N)$ onto $L(N)$ is a lower bound on the number of rows needed to route the edges of $G(N)$, the result follows. ■

The Layout Method for Product Graphs

The following theorem represents the main result of this section. The proof gives an algorithm to obtain the layout for a product graph from a normal collinear layout of its factor graph.

Theorem 6.12 *If $G(N)$ has a normal collinear layout with wiring width w, then $PG_r(N)$ has a layout with square nodes of side $\Delta\lceil r/2\rceil$ placed regularly in $N^{\lceil r/2\rceil}$ columns of $N^{\lfloor r/2\rfloor}$ nodes each, where two adjacent columns of nodes are at distance $w\sum_{i=0}^{\lfloor r/2\rfloor-1} N^i$ and two adjacent rows of nodes are at distance $w\sum_{i=0}^{\lceil r/2\rceil-1} N^i$.*

Proof: We show the iterative process that can be used to obtain the desired layout. The proof is illustrated in Figure 6.4, which presents the construction of a layout for the 3-dimensional hypercube. Figure 6.4.(a) presents a normal collinear layout for the 2-node linear array.

Initially, we place the N^r nodes of $PG_r(N)$ in the layout as squares of side $\Delta\lceil r/2\rceil$ in a grid fashion with $N^{\lceil r/2\rceil}$ columns of nodes and $N^{\lfloor r/2\rfloor}$ rows of nodes. Each node touches its neighbor nodes in the layout. The size of the nodes will guarantee that there are

side across the columns just created. Figure 6.3.(c) presents the 2 new columns created in this step. Then, move u to the bottom rows, after resizing it to $\delta_u \times \Delta$. Finally, use the newly created columns as well as the rows originally allocated to u to reroute the edges from the bottom rows. Since u had at least δ_u rows and we have δ_u columns, this rerouting can be done. Figure 6.3.(d) presents the final result for our example.

This ends the transformation. Note that the total number of added columns is $\sum_{u \in V} \delta_u$, where V is the set of nodes of $G(N)$ and, therefore, the length of the layout is $O(l + N\Delta)$. ■

The above lemma shows that any layout can be transformed into a seminormal collinear layout with wiring width of the same order as the width of the original layout. While the transformation increases the length of the collinear layout, we will see that it is the width of the normal collinear layout which dominates the layout area complexity for the product graph. The collinear layout obtained can now be compressed to obtain a normal collinear layout with at most same wiring width. This is shown in the following lemma.

Lemma 6.4 *If $G(N)$ has a seminormal collinear layout with wiring width w and bandwidth b, it also has a normal collinear layout with wiring width at most w and bandwidth b.*

Proof: The original layout gives us a possible labeling (*i.e.* the order in which the nodes of $G(N)$ can be placed) to obtain the desired wiring width w. This is all we need for the purpose of obtaining the desired normal layout. We start by placing the N nodes touching each other along a straight line. The ith node in this line corresponds to the ith node in the seminormal collinear layout. We then connect these nodes by three-segment wires (two vertical and one horizontal) as required by the original layout.

Since there is a seminormal collinear layout of width w that uses the same node order, we can obtain a layout which has at most w rows used for wires. The bandwidth of the layout remains the same. ■

Note that, in the above obtained layout, the length of the longest wire is at most $2w + b\Delta$, where b is the bandwidth of the layout.

We finish this section by presenting a lower bound on the wiring width of any normal collinear layout for arbitrary graphs.

Theorem 6.11 *If the maximal congestion of $G(N)$ is C then the wiring width for any normal collinear layout of $G(N)$ is at least $N^2/2C$ if N is even, and $(N^2-1)/2C$ if N is odd.*

Proof: Note first that any embedding of the N-node directed complete graph onto the linear array $L(N)$ requires congestion $N^2/2$ if N is even, and $(N^2-1)/2$ if N is odd.

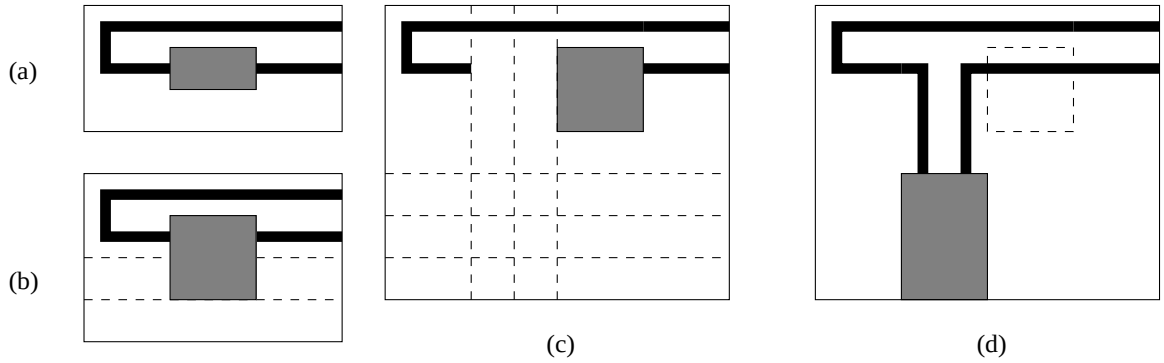

Figure 6.3: Transformation of a compact layout into a collinear layout.

isolated nodes, and at step i at most $F/\sqrt{2^i}$ edges are removed to divide the graph, for $i = 0, ..., 2\log F$. Then, the number of rows needed to route the edges of $G(N)$ are at most $\sum_{i=0}^{2\log F} F/\sqrt{2^i} = O(F)$. ■

Finally, we present a general method to obtain a normal collinear layout from an arbitrary layout. The following result shows that there is always a seminormal collinear layout with small wiring width.

Lemma 6.3 *If $G(N)$ has a layout of length l and with w, $G(N)$ also has a seminormal collinear layout of length $O(l + N\Delta)$ and wiring width $O(w)$.*

Proof: We prove the lemma by showing how to transform the given layout into a seminormal collinear layout with the claimed dimensions. The transformation is illustrated in Figure 6.3, where we show only the process for one node of the layout. The appearance of this node in the original layout is shown in Figure 6.3.(a).

We first transform the nodes of the given layout by adding enough rows so that each node uses at least δ_u rows, where δ_u is the vertex degree of the node u being transformed. The width of the resulting layout is at most $O(w)$. Figure 6.3.(b) shows the result of enlarging our example node to use 2 rows.

We subsequently create Δ new rows at the bottom of the layout. We will eventually move all the nodes in the layout to these new rows. No other rows are added in the rest of the transformation process, therefore the wiring width of the new layout will be $O(w)$. In our example we assumed $\Delta = 3$ and then, at the bottom of Figure 6.3.(c), the 3 new rows introduced are shown.

Then, the following step is applied iteratively until all the nodes are in the created bottom rows and we have a seminormal collinear layout. The step searches from left to right for the first column with tiles assigned to nodes not yet moved. This column can have tiles from several nodes. If so, we take one node arbitrarily.

Let u be the node we have chosen. Create δ_u new columns on the left side of u. When creating these columns do not stretch the end of the wires incident to u in that

also on the bandwidth of the collinear layout. Below, we present several ways to obtain normal collinear layouts with small wiring width for arbitrary graphs.

3. It is applicable to any graph regardless of the vertex degree, while the applicability of bisector or bifurcator-based approaches are limited to graphs with bounded degree.

4. The aspect ratio of layouts is always $O(1)$, which is a desirable characteristic for fabrication.

Methods for Obtaining Normal Collinear Layouts

In this section we are interested in obtaining normal collinear layouts with small wiring width and small bandwidth, because these are the properties that influence the characteristics of the layout of the product graph that we obtain. We can devise several methods to obtain normal collinear layouts for any graph.

First, observe that any graph $G(N)$ has a normal collinear layout of wiring width at most $\frac{1}{2}\sum_{u \in V} \delta_u$, where V is the set of nodes of $G(N)$, since this is the number of wires in the layout and each wire requires no more than one row.

Second, the problem of finding an efficient collinear layout is closely related to the class of problems known as "graph labeling" [16]. Given a way of labeling the nodes of an N-node graph with integer labels $1, ..., N$ (or, equivalently, given a way of placing the nodes of the graph on a line), the maximum distance between two connected nodes is the *bandwidth* of the labeling, while the maximum number of edges that cross a vertical line placed between any two nodes is the *cutwidth* of the labeling. Thus, if there is an embedding of a graph $G(N)$ onto the linear array $L(N)$ with bandwidth b and cutwidth c it is trivial to obtain a normal collinear layout for $G(N)$ with wiring width c, bandwidth b, and longest edge of length $O(b\Delta + c)$. For an arbitrary graph, we can obtain a labeling which minimizes the bandwidth and the cutwidth by using dynamic programming algorithms, or heuristics.

Third, it is shown in [44] how to construct normal collinear layouts for a graph $G(N)$ with a $f(x)$-separator. The layout has wiring width $O(f(N)\log N)$ in general but, if $f(x) = \Omega(x^{\alpha})$ for $\alpha > 0$, then the wiring width is $O(f(N))$. For graphs with F-bifurcators we obtain a similar result in the following lemma.

Lemma 6.2 *If $G(N)$ has a F-bifurcator, then it has a normal collinear layout with wiring width $O(F)$.*

Proof: The construction of the layout is similar to the construction shown in [44] for separators. We use a divide-and-conquer process that divides the graph into two subgraphs, obtains a collinear layout for each, and reconnects the two layouts by adding at most as many new rows as edges were removed in the partition step. From the definition of bifurcator, the division process is applied at most $2\log F$ times before we obtain

and j to count the partitions within an application of the basic process, varying j from 0 to $r-1$. The absolute count of partition steps for the whole graph is, then, $k = ir + j$.

In the ith application of the basic process the size of the factor subgraphs that we are considering is $m = N/2^i$ and we remove at most $O((N/2^i)^\alpha)$ edges to partition this subgraph. Then, the kth absolute partition step removes at most $O((N/2^i)^\alpha (N/2^i)^{r-j-1} (N/2^{i+1})^j)$ edges. We can write

$$(N/2^i)^\alpha (N/2^i)^{r-j-1} (N/2^{i+1})^j = \frac{N^{r+\alpha-1}}{2^{k-i(1-\alpha)}}.$$

Since $k - i(1-\alpha) \geq k/2$ for $r \geq 2$ we can conclude that $PG_r(N)$ has a $O(N^{r+\alpha-1})$-bifurcator. We still need to show in which cases this is a $(r+\alpha-1)$-type B bifurcator. Note that

$$\frac{N^{r+\alpha-1}}{2^{k-i(1-\alpha)}} = \frac{(N^r)^{1-(1-\alpha)/r}}{(2^k)^{1-i(1-\alpha)/k}}.$$

Since $(1-\alpha)/r \geq i(1-\alpha)/k$ then $1 - i(1-\alpha)/k \geq 1 - (1-\alpha)/r$ and, therefore,

$$\frac{(N^r)^{1-(1-\alpha)/r}}{(2^k)^{1-i(1-\alpha)/k}} = O((N^r/2^k)^{1-(1-\alpha)/r})$$

where $1 - (1-\alpha)/r > 1/2$ (*i.e.* we have a type B bifurcator) if either $r > 2$, or $r = 2$ and $\alpha > 0$. ■

We can now apply theorems 6.3 and 6.4 combined with this theorem to obtain the following corollary.

Corollary 6.3 *If $G(N)$ has a α-special bifurcator then $PG_r(N)$ can be laid out in an area of $O(N^2 \log^2 N)$ with maximum wire length $O(N \frac{\log N}{\log\log N})$ if $r = 2$ and $\alpha = 0$, or in an area of $O(N^{2(r+\alpha-1)})$ with maximum wire length $O(N^{r+\alpha-1})$ otherwise.*

6.3.3 Upper Bounds Based on Collinear Layouts

In this section we present another approach to obtain layouts for product networks. We use a collinear layout for the factor network to obtain a layout for the homogeneous product network. This collinear approach for laying out product graphs has several advantages:

1. It gave the optimal area layouts for all the cases we considered (with only one exception), and wire lengths were quite close to optimal.

2. It depends on obtaining a collinear layout for the factor graph, which is much easier to obtain than good bisectors or bifurcators. The area of the layout only depends on the wiring width of the collinear layout. The maximum wire length depends

the worst case, we take the larger of the two obtained subgraphs. We can partition this subgraph by dimension 2 by removing at most $H\lceil N/2\rceil N^{r-2}$ edges. This value is smaller than $HN^{r-1}/\sqrt{2}$.

We can continue in this way partitioning the subgraphs by each dimension. When dividing the largest subgraph by dimension i we remove at most $H\lceil N/2\rceil^{i-1}N^{r-i}$ edges, that is smaller than $HN^{r-i}/\sqrt{2}^{i-1}$. After dividing by dimension r each subgraph obtained has at most $\lceil N/2\rceil$ nodes along each dimension.

If we start the process again, the next division will remove at most $H\lceil N/2\rceil^{r-1}/\sqrt{2}$ edges, that is smaller than $HN^{r-1}/\sqrt{2}^{r}$. Therefore, the process can be repeated without exceeding the maximum number of edges allowed by the definition of bifurcator.

As in the proof of Theorem 6.7, we can apply the partition process just described to each of the 2^r subgraphs of $PG_r(N)$ obtained, to the subgraphs obtained from them, and so on, until all the nodes are isolated. By repeating this process at most $\log N + 1$ times all the nodes in $PG_r(N)$ will be isolated, and the theorem follows. ■

We recall here that Bhatt and Leighton [9] showed that if $G(N)$ can be laid out in an area A it has a $\sqrt{A}$-bifurcator. Thus, if $G(N)$ can be laid out in an area A then $PG_r(N)$ has a $N^{r-1}6(2+\sqrt{2})\sqrt{A}$-bifurcator. From Theorem 6.9 we can obtain bifurcator-based bounds for the area and maximum wire length by using Theorem 6.3.

Corollary 6.2 *If $G(N)$ has a F-bifurcator then $PG_r(N)$ can be laid out in an area of $O(N^{2(r-1)}F^2\log^2(N/F))$ with maximum wire length $O(N^{r-1}F\frac{\log(N/F)}{\log\log(N/6(2+\sqrt{2})F)})$.*

The above theorem and corollary are universally applicable. However, as Bhatt and Leighton [9] noted, there are graphs with special characteristics which allow to improve the above bounds. This fact is reflected in the following results.

Theorem 6.10 *If $G(N)$ has a α-special bifurcator then $PG_r(N)$ has a $O(N^{r+\alpha-1})$-bifurcator. This is a $(r+\alpha-1)$-type B bifurcator if either $r>2$, or $r=2$ and $\alpha>0$.*

Proof: Let N be a power of two for simplicity. From Theorem 5 in [9] we know that $G(N)$ has a partition process where each partition in the ith partition step bisects the corresponding graph without removing more than $6\sum_{s=i}^{p}O((N/2^s)^\alpha)$ edges, for $i=0...\log N-1$, and p is the number of steps of the original partition process. This summation is a decreasing geometric series, that is essentially on the order of its first term. Then, $G(N)$ has a partition process such that each partition in the ith partition step bisects the corresponding graph without removing more than $6O((N/2^i)^\alpha)=O((N/2^i)^\alpha)$ edges, for $i=0...\log N-1$.

The partition process is similar to the one presented in the proof of Theorem 6.9. We apply a basic process $\log N$ times, partitioning the graphs r times in each application. We will use i to count the applications of the basic process, i varying from 0 to $\log N-1$,

the proof is complete. ■

Once we obtain a bisector for product networks we are ready to apply it to obtain bounds on the layout parameters. We can use Theorem 6.1 to obtain the following result.

Theorem 6.8 *If $G(N)$ has a $f(x)$-bisector then $PG_r(N)$ can be laid out in a square of side $O(Nf(N)\log N)$ when $r = 2$, or side $O(N^{(r-1)}f(N))$ when $r > 2$.*

Proof: In Theorem 6.7 we have obtained that $PG_r(N)$ has a $O(x^{(r-1)/r}f(x^{1/r}))$-bisector. Since $PG_r(N)$ has N^r nodes, we can obtain the value of the summation presented in Theorem 6.1 as $\sum_{i=0}^{r\log_4 N} 2^i O((N^r/4^i)^{(r-1)/r}f(N/4^{i/r})) = O(f(N)\sum_{i=0}^{r\log_4 N} 2^i(\frac{N^r}{4^i})^{(r-1)/r})$ since $f(x)$ is a monotonically non-decreasing function. The value of this last summation is $O(N\log N)$ when $r = 2$, or $O(N^{r-1})$ when $r > 2$ [80]. Therefore, the value of the first summation is $O(Nf(N)\log N)$ when $r = 2$, or $O(N^{r-1}f(N))$ when $r > 2$, and the claim follows. ■

The most studied kind of bisectors has been $O(x^\alpha)$-bisectors, for bounded α. Theorem 6.2 can be directly applied to product networks to obtain the next corollary.

Corollary 6.1 *If $G(N)$ has a $O(x^\alpha)$-bisector, for bounded α, then $PG_r(N)$ can be laid out in an area of $O(N^2\log^2 N)$ with maximum wire length $O(N\log N/\log\log N)$ when $\alpha = 0$ and $r = 2$, or in an area of $O(N^{2(r+\alpha-1)})$ with maximum wire length $O(N^{r+\alpha-1})$ otherwise.*

6.3.2 Upper Bounds Based on Bifurcators

The following theorem and its corollary present the initial general results of this section. After these we present additional results applicable to graphs with α-special bifurcators, which yield tighter bounds.

Theorem 6.9 *If $G(N)$ has a F-bifurcator then $PG_r(N)$ has a $N^{r-1}6(2+\sqrt{2})F$-bifurcator.*

Proof: From Theorem 6 in [9] we know that if $G(N)$ has a F-bifurcator then it has a $H = 6(2+\sqrt{2})F$-bifurcator (balanced bifurcator) that bisects the graph at each partition. Then, after at most $\log N+1$ partitions $G(N)$ is transformed into N isolated nodes. Along the rest of the proof we will denote $6(2+\sqrt{2})F$ as H for brevity.

The proof is very similar to the proof of Theorem 6.7. We show that given $PG_r(N)$ we can obtain 2^r subgraphs, each being the r-dimensional (possibly heterogeneous) product of factor graphs with $H/\sqrt{2}$-bifurcators and at most $\lceil N/2\rceil$ nodes.

We initially consider dimension 1. To partition $PG_r(N)$ we can bisect each $G(N)$-subgraph in this dimension, removing no more than HN^{r-1} edges in total. Each dimension-1 $G(N)$-subgraph is so divided into a $\lfloor N/2\rfloor$-node and a $\lceil N/2\rceil$-node subgraphs. To follow

Case N odd: The logic in this case is similar to the logic in the above case, but we must be careful because by simply bisecting each subgraph along a dimension we are not bisecting the whole graph. What we do in this case is breaking each $G(N)$-subgraph in a given dimension into two subgraphs, with $(N-1)/2$ nodes each, and one isolated node. As the isolated nodes are connected between themselves by the other dimensions, we also remove these connections and distribute the so obtained isolated nodes evenly between the two large connected subgraphs.

We can initially take dimension 1. By bisecting each dimension-1 $G(N)$-subgraph we remove no more than $N^{r-1}f(N)$ edges and we obtain two subgraphs with $N^{r-1}(N-1)/2$ and $N^{r-1}(N+1)/2$ nodes, respectively. Clearly, $PG_r(N)$ has not been bisected. Now, we can take the subgraph with the larger number of nodes and isolate one node along dimension 1 from each of the dimension-1 subgraphs, the same node in each subgraph. Since we are assuming that $G(N)$ has bounded vertex degree, we can do so by removing a bounded number of edges from each dimension-1 subgraph. This leads to a total of $O(N^{r-1})$ edges removed.

Now we have two subgraphs with $N^{r-1}(N-1)/2$ nodes each, and a $(r-1)$-dimensional subgraph, isomorphic to PG_{r-1}. From Lemma 6.1, the factor graph $G(N)$ that generates the $(r-1)$-dimensional subgraph has no more than $O(Nf(N))$ edges. Therefore we can isolate the nodes of this subgraph by removing at most $(r-1)N^{r-2}O(Nf(N)) = O(N^{r-1}f(N))$ edges.

As a result of the above process we have two subgraphs with the same number of nodes and some isolated nodes. If we distribute the isolated nodes evenly between the two subgraphs our bisection is done. The total number of edges removed has been $N^{r-1}f(N) + O(N^{r-1}) + O(N^{r-1}f(N)) = O(N^{r-1}f(N))$ from the initial N^r-node graph.

This process can be applied to each dimension as in the case of N even. In each application $O(N^{r-1}f(N))$ edges are removed from a $\Theta(N^r)$-node graph. After the graph has been bisected in this way along each dimension, we have 2^r disjoint r-dimensional subgraphs, each being the product of $((N-1)/2)$-node graphs with $f(x)$-bisector, plus several isolated nodes distributed evenly between them.

We have now 2^r subgraphs of $PG_r(N)$ each being the r-dimensional product of factor graphs with $\lfloor N/2 \rfloor$ nodes and $f(x)$-bisectors. Note that in the above described process we only use the fact of $PG_r(N)$ having the same number of nodes along each dimension and of each factor graph having a $f(x)$-bisector. Since the obtained subgraphs fullfil these requirements, the described process can be applied again to each of them. Subsequently, the subgraphs obtained from them will also fullfil the requirements, and the process can be applied to each of them, and so on, until all the nodes are isolated.

Since in each bisection of the whole process the number of edges removed does not exceed the limits imposed by the definition of $g(x)$-bisector for $g(x) = O(x^{(r-1)/r}f(x^{1/r}))$,

Proof: We initially present the following lemma that shall be used in the proof.

Lemma 6.1 *If $G(N)$ has a $f(x)$-bisector, then it has at most $O(Nf(N))$ edges.*

Proof: Assume for simplicity that N is a power of 2. By the definition of bisector, $G(N)$ can be divided into two subgraphs by removing no more than $f(N)$ edges. Then, we obtain 2 subgraphs with $N/2$ nodes each, which can be bisected by removing no more that $f(N/2)$ edges from each. After i bisections of this kind, we obtain 2^i subgraphs with $N/2^i$ nodes each, which in turn can be bisected by removing no more that $f(N/2^i)$ edges from each. After applying the bisection process $\log N$ times we obtain N isolated nodes. The maximum number of edges removed in the whole process can be easily computed as, $f(N) + 2f(N/2) + 2^2 f(N/2^2) + ... + 2^{\log N-1} f(N/2^{\log N-1}) = \sum_{i=0}^{\log N-1} 2^i f(N/2^i) = O(Nf(N))$. ■

The proof now shows how to divide $PG_r(N)$ into isolated nodes by repeatly applying bisections that respect the definition of $O(x^{(r-1)/r} f(x^{1/r}))$-bisector.

Initially, we show how to divide $PG_r(N)$ into 2^r disjoint subgraphs and, possibly, some isolated nodes. This process is done in r bisection steps, each of which removes $O(N^{r-1} f(N))$ edges from its corresponding graph. At the end of the process, each of the obtained subgraphs is the r-dimensional (possibly heterogeneous) product of factor graphs with $\lfloor N/2 \rfloor$ nodes and $f(x)$-bisectors.

A partition process similar to the one applied to $PG_r(N)$ can then be applied to each of these subgraphs, to the subgraphs obtained from them, and so on, until all the nodes are isolated.

The basic partition process considers two cases, when N is even and when N is odd.

Case N even: By definition of bisector, each of the $G(N)$-subgraphs in each dimension can be bisected by removing no more than $f(N)$ edges. We can initially consider only the $G(N)$-subgraphs in dimension 1. $PG_r(N)$ can be divided into two subgraphs with the same number of nodes in each by bisecting each of the dimension-1 $G(N)$-subgraphs. As there are N^{r-1} such subgraphs, we have removed no more than $N^{r-1} f(N)$ edges from the N^r-node graph.

Now, we can take one of the two subgraphs of $PG_r(N)$ obtained and divide it into two subgraphs with same number of nodes by bisecting each of its dimension-2 $G(N)$-subgraphs. The number of edges removed this time is no more than $N^{r-1} f(N)/2$ from a graph with $N^r/2$ nodes.

We can continue this process, bisecting the obtained subgraphs along each dimension. When bisecting the subgraphs by dimension i we are removing no more than $N^{r-1} f(N)/2^{i-1} = O(N^{r-1} f(N))$ edges from $N^r/2^{i-1} = \Theta(N^r)$-node graphs.

After bisecting the subgraphs by dimension r we obtain 2^r disjoint subgraphs, each being the r-dimensional product of $(N/2)$-node graphs with $f(x)$-bisectors (because they are bisections of graphs with $f(x)$-bisectors.)

two parameters anymore, since the maximal congestion gives us the same bounds that we can obtain from them.

Now we present a lower bound on the length of the longest wire in any layout of a product graph.

Theorem 6.6 *If the maximal congestion of $G(N)$ is C and its diameter is d, then the length of the longest wire in any layout of $PG_r(N)$ is at least $\Omega(\frac{N^{r+1}}{Crd})$.*

Proof: Theorem 5-2 in [41] shows that any layout of a graph with diameter D and minimum layout area A has some wire of length at least $A^{1/2}/3D$. From Theorem 3.1, the diameter of $PG_r(N)$ is $D = rd$ and, from Theorem 6.5, its layout area is at least $\Omega(\frac{N^{2(r+1)}}{C^2})$. Therefore, we can conclude that any layout of $PG_r(N)$ has some wire of length at least $\frac{\Omega(N^{r+1}/C)}{3rd} = \Omega(\frac{N^{r+1}}{Crd})$. ■

6.3 Upper Bounds

In this section we first present upper bounds obtained by traditional frameworks, namely bisectors and bifurcators. We show that, given a bisector or a bifurcator for the factor graph, we can obtain a bisector or a bifurcator for the product graph. Since these frameworks are only applicable to networks with bounded vertex degree, we will assume that the factor graph has bounded vertex degree and that the number of dimensions of the product network is also bounded. These assumptions are not very restrictive if we are dealing with factor networks that can grow without increasing the vertex degree.

Sherlekar and JáJá [70, 71] investigated the use of separators and bifurcators to obtain efficient layouts for unbounded-vertex-degree graphs. However, the kinds of separators and bifurcators they use are so restrictive that it does not seem possible to obtain simple general results for product graphs by using them.

Subsequently, we present another approach that is universally applicable and does not have any restriction on the vertex degree or on the number of dimensions. This method of obtaining efficient layouts for product networks is based on the existence of efficient collinear layouts for the factor networks. We show that it is always possible to find reasonably efficient collinear layouts for any network and present a specific technique to do so.

6.3.1 Upper Bounds Based on Bisectors

The following theorem presents the basic result of this section.

Theorem 6.7 *If $G(N)$ has a $f(x)$-bisector, then $PG_r(N)$ has a $O(x^{(r-1)/r} f(x^{1/r}))$-bisector.*

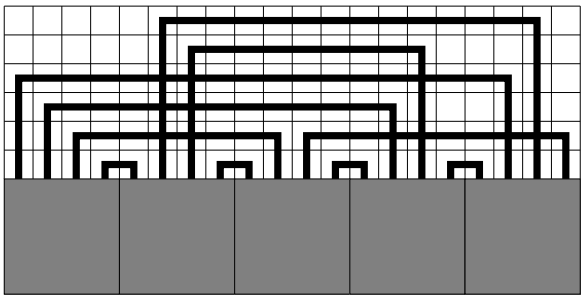

Figure 6.2: Normal collinear layout for $K(5)$.

Definition 6.7 *The* wiring width *of a collinear layout is the number of rows used to route the wires in the layout.*

The value of the wiring width is always the width of the layout minus the maximum vertex degree Δ.

Definition 6.8 *The* bandwidth *of a collinear layout is the maximum distance, in number of nodes, between any two connected nodes.*

The maximum wire length is closely related to the bandwidth of a layout as we discuss later. Figure 6.2 presents a normal collinear layout for $K(5)$ with wiring width 6 and bandwidth 4.

6.2 Lower Bounds

In this section we obtain lower bounds on the layout area and maximum wire length required by any layout of $PG_r(N)$.

In [78], Thompson showed that the square of the bisection width is a lower bound (within a constant factor) on the wire area required by any layout of a graph. Similarly, Leighton [41] presented the crossing number as a lower bound on the wire area of any layout of a graph. Then, we can use theorems 3.9 and 3.10 to prove the following theorem.

Theorem 6.5 *If the maximal congestion of $G(N)$ is C then the layout area of $PG_r(N)$ is at least $\Omega(\frac{N^{2(r+1)}}{C^2})$.*

This result emphasizes the importance of the maximal congestion as a parameter of a graph. Previously, the two approaches to obtain lower bounds on the layout area, bisection width and crossing number, were considered independent from each other. Here, we have shown the maximal congestion as a link between both approaches, which allow to obtain tight lower bounds on the area. In fact, in many cases we do not need the other

Definition 6.3 *A n-node graph has a α-special bifurcator, $0 \leq \alpha \leq 1$, if it has a $O(max\{\sqrt{n}, n^{\alpha}\})$-bifurcator such that no more than $O((n/2^i)^{\alpha})$ edges are removed in each partition at the ith step of the partition process, where $i = 0$ initially.*

Note that when $\alpha = 1/2$ we have the definition of $\sqrt{n}$-bifurcator, but for $\alpha \neq 1/2$ the partition process defined is more restrictive than the one implied in Definition 6.2. We now define a subclass of graphs with α-special bifurcators. This subclass was originally considered in [9].

Definition 6.4 *A graph has a α-type B bifurcator if it has a α-special bifurcator, where $\alpha > 1/2$.*

From the value of the bifurcator of a graph, there have been presented the following results in [9].

Theorem 6.3 *A n-node graph with a F-bifurcator can be laid out in an area of $O(F^2 \log^2 \frac{n}{F})$ with maximum wire length of $O(F \frac{\log \frac{n}{F}}{\log\log \frac{n}{F}})$.*

Theorem 6.4 *A n-node graph with a α-type B bifurcator can be laid out in an area of $O(n^{2\alpha})$ with maximum wire length of $O(n^{\alpha})$.*

Again, the bifurcator framework is restricted to be used with bounded degree networks.

6.1.4 Collinear Layouts

The last approach to obtain layouts for homogeneous product networks we investigate is based on the existence of efficient collinear layouts of the factor graph. A VLSI layout is called *collinear* if all the nodes are placed along a straight line. We will use collinear layouts of the factor graph to generate layouts for the product graph.

To be able to use them, we impose several restrictions on the collinear layouts. We assume that the nodes are aligned horizontally.

Definition 6.5 *A collinear layout is* seminormal *if all the nodes in the layout are placed at the bottom rows of the layout, a node u occupies Δ rows and Δ_u columns, and all the wires are laid down above the row Δ.*

Definition 6.6 *A collinear layout is* normal *if it is seminormal, all the nodes are adjacent, and all the wires are laid down as two vertical sections connected by a horizontal section.*

For these two classes of layouts we can define two new parameters.

Definition 6.1 *Let $f(x)$ be a monotonically non-decreasing function. A n-node graph has a $f(x)$-bisector either if it has only one node or if by removing at most $f(n)$ of its edges it can be divided into two subgraphs with the same number of nodes (within one), both with $f(x)$-bisectors.*

In general, separators need not bisect the graph at each stage. Our definition is more restrictive, for instance, than the definition of separator used by Leiserson [44]. However, Ullman [80] showed how to obtain a bisector (he calls it strong separator) from separators as defined by Leiserson.

The separator framework is restricted to lay out graphs with bounded vertex degree. This is an important restriction on the applicability of this framework.

We also present several results for graphs with given bisectors. The following result can be found in [80].

Theorem 6.1 *A n-node graph with a $f(x)$-bisector can be laid out in a square area whose side is $O(max\{\sqrt{n}, \sum_{i=0}^{\log_4 n} 2^i f(n/4^i)\})$.*

The kind of bisectors most commonly considered are those of the form $f(x) = O(x^\alpha)$. For them the above summation takes a value $O(\sqrt{n})$ if $\alpha < 1/2$, a value $O(n^\alpha)$ if $\alpha > 1/2$, and a value $O(\sqrt{n}\log n)$ if $\alpha = 1/2$. For graphs with these kind of bisectors it has been shown that the above areas can be obtained with the following maximum wire length [41].

Theorem 6.2 *A n-node graph with a $f(x)$-bisector, where $f(x) = O(x^\alpha)$, can be laid out in an area of $O(n)$ with maximum wire length of $O(\sqrt{n}\log n)$ if $\alpha < 1/2$, in an area of $O(n^{2\alpha})$ with maximum wire length of $O(n^\alpha)$ if $\alpha > 1/2$, and in an area of $O(n\log^2 n)$ with maximum wire length of $O(\sqrt{n}\log n/\log\log n)$ if $\alpha = 1/2$.*

6.1.3 Bifurcators

Bifurcators appeared as an alternative to separators. They solve some of the problems and restrictions of the separator framework.

Definition 6.2 *A graph has a F-bifurcator either if it has only one node or if by removing at most F of its edges it can be divided into two subgraphs, both with $F/\sqrt{2}$-bifurcators.*

The definition of bifurcator implies a way to iteratively partition the graph so that in the ith step of this partition process (with $i = 0$ initially) no more than $F/\sqrt{2}^i$ edges are removed in each partition (at the ith step of the partition process we are partitioning 2^i disjoint graphs.) Special cases of graphs will be considered based on additional restrictions imposed in this partition process.

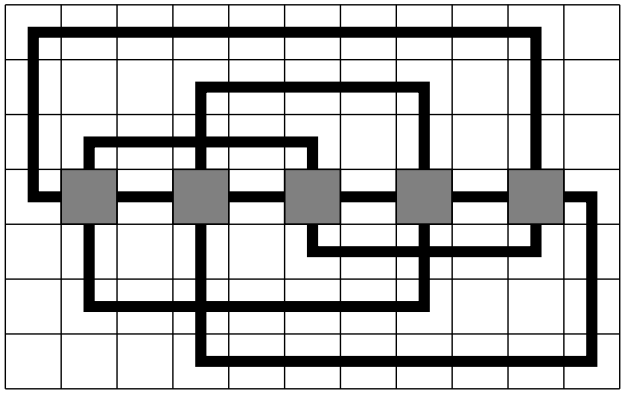

Figure 6.1: Collinear layout for $K(5)$.

node u with vertex degree δ_u is laid out as a rectangle with sides of length at least $\Omega(\delta_u)$.

Under this model, the *wire area* of a layout is the number of tiles that hold either a section of a wire or a wire crossing. The *length of a wire* is the number of tiles traversed by the wire from its source node to its destination node. For technological reasons [80], the *layout area* is defined as the area of the smallest rectangle that contains all the allocated tiles of the layout. This value is fully described with the length and the width of this rectangle. We assume that the *width* of a layout is the length of the shorter side of the rectangle and the *length* of the layout is the length of the longer side. We also assume that the rectangle is oriented in the grid with the longer side horizontally placed.

Figure 6.1 shows a layout for the 5-node complete graph, $K(5)$, with layout area of 77, wire area of 55, width of 7, length of 11, and length of the longest wire of 15.

The area of a layout strongly determines its fabrication cost. It has been shown that the larger the area, the smaller the yield of the manufacturing process [80]. Furthermore, the reduction of the yield is exponential with the area. Therefore it is interesting to have layouts with the least possible area.

In the other side, we also try to reduce the length of the longest wire in the layout, since it imposes a restriction on the speed of the system [80]. This is due to the propagation time of signals from one extreme to the other of the wire, which at the processing speeds of a VLSI system is not negligible.

Thompson [77, 78] showed that the square of the bisection width is a lower bound on the wire area of any network. Leighton [41] presented the crossing number as an even tighter bound for the wire area. These fact will be used to obtain lower bounds on the area for our networks.

6.1.2 Separators

In this chapter we consider a special kind of separators, which we denote as "bisectors."

Chapter 6

VLSI Layout Complexity

This chapter explores the VLSI layout complexity of homogeneous product networks. Here, we obtain lower and upper bounds on the area and wire length of layouts for these networks.

In the following section we present relevant background for this chapter. Then we present the lower bounds obtained. Finally, we present upper bounds derived by using two traditional frameworks, separators and bifurcators, and a new approach based on collinear layouts.

6.1 Foundations

In this section we start by defining the VLSI layout model used. Then, we refer to previous popular frameworks used to derive layouts under this model: separators and bifurcators. Finally, we will present a special kind of layouts (collinear layouts) that will be used in this research to obtain efficient layouts for product graphs.

6.1.1 The Thompson's Grid Model

The VLSI layout model we use was defined by Thompson [77, 78]. In this model, the layout area is divided into square "tiles" of unit area, placed in a grid fashion. Each tile can hold either a section of wire, a node, or a wire crossing. The wires of the layout run either horizontally or vertically on this grid. If two wires enter the same tile they must have different directions and they cannot change direction in the tile.

Observe that since a node is assigned to a tile, the nodes are not allowed to have more than 4 incident wires. When a node has a degree larger than 4, Thompson has proposed to model it with a set of adjacent tiles whose perimeter is at least the desired degree. Although the smallest area required to have a perimeter of δ_u for a node u has only $O(\delta_u)$ tiles, it is much more realistic to assume that a node with vertex degree δ_u will require area of at least $\Omega(\delta_u) \times \Omega(\delta_u)$. In this dissertation we shall assume that any

to the square of the number of nodes of the input graph. This is the best time complexity known for any algorithm solving this problem.

In the $n \times n$ grid it is know an algorithm that finds the tree in $O(n)$ time. If we consider $n = N^{r/2}$, our algorithm takes $O(r^2 N \log N)$ time, that is better for a large enough value of r.

Factor network	Sorting	Summation	Matrix mult.	Min. weight sp. tree
$L(N)$	$O(r^2N)$ *	$O(rN)$ *	$O(rN)$ *	$O(r^2N\log N)$
$T(N)$	$O(r^2N)$ *	$O(r\log N)$ *	$O(r\log N)$ *	$O(r^2\log^2 N)$
$S(N)$	$O(r^2\log^2 N)$	$O(r\log N)$ *	$O(r\log N)$ *	$O(r^2\log^2 N)$
$D(N)$	$O(r^2\log^2 N)$	$O(r\log N)$ *	$O(r\log N)$ *	$O(r^2\log^2 N)$
Q_1	$O(r^2)$	$O(r)$ *	$O(r)$ *	$O(r^2)$

Table 5.1: Time complexity of the presented algorithms in several networks.

From the results presented, it can be observed that the time taken by the sorting algorithm in the grid and the product of complete binary trees with bounded number of dimensions is $O(N)$, which is optimal. In Q_r the algorithm takes $O(r^2)$ time steps, reaching the asymptotic bound of the odd-even merge sorting algorithm in the hypercube. Although there are asymptotically better sorting algorithms for the hypercube [19], they are not practically useful for reasonable number ($\leq 2^{20}$) of keys.

In other networks our sorting algorithm improves the computation time taken by alternative ways of sorting. For instance, in the product of de Bruijn or shuffle-exchange graphs we could try to embed the N^r-node instance of the pure factor graph and use the sorting algorithm for the factor graph to sort in the product graph. If we use the odd-even merge sorting algorithm in the factor graph, this algorithm will take $O(r^2\log^2 N)$ steps that have to be emulated by the product graph. Since the embedding has congestion $\Omega(r)$, the emulation will not have constant slowdown, and our algorithm will be preferable. If we consider the number of dimensions bounded then both options present same asymptotic complexity, but our algorithm has a smaller constant.

The values obtained for the summation algorithm are asymptotically optimal, since they match the lower bound defined by the diameter of the respective networks.

For the same reason, the time complexities for matrix multiplication are asymptotically optimal. In some cases we might be poorly using the amount of processors we have. For instance, there are algorithm that can multiply larger matrices in the grid $PL_3(N)$ than ours. However, in this case, we can always emulate this network with any homogeneous product network if $G(N)$ is connected.

For most of the studied network the matrix multiplication algorithm performs very efficiently. The implementation in the product of complete binary trees performs as efficiently as the algorithm for mesh of trees presented in [58], while it uses less processors ($4N^r - 3N^{r/3}$ opposed to N^r.) The implementation in the hypercube obtains the same time complexity as the fastest algorithm for hypercubes (in fact both algorithms turn out to be almost the same.) Again, for products of shuffle-exchange and de Bruijn the algorithm outperforms the option of emulating the best algorithm for the factor network.

The presented complexities for the minimum-weight spanning tree problem are worst-case values. In most cases we obtain that at most we need a number of steps proportional

For the other algorithms, if we take as node 0 the node in the center of $L(N)$, then broadcasting, point-to-point communication, summation, and search of the minimum takes at most $\lfloor N/2 \rfloor + 1$ steps. Hence, the time to obtain the summation of N^r values in $PL_r(N)$ is $r\lfloor N/2 \rfloor + 1$, the time to multiply two $N^{r/3} \times N^{r/3}$ matrices is $2r\lfloor N/2 \rfloor/3 + 1$, and the time to obtain the minimum-weight spanning tree of a $N^{r/2}$-node graph is $4r^2 \lfloor N/2 \rfloor \log_{4/3} N + 1$.

Product of complete binary trees. For this network we can directly apply Corollary 5.1 and obtain that $PT_r(N)$ can sort N^r values in $O(r^2 N)$ time steps.

If the special node 0 is the root of $T(N)$, it can do all the required operations for the other algorithms in $\log(N+1)$ steps. We can obtain, then that the summation algorithm will take $r\log(N+1) - r + 1$ time steps in $PT_r(N)$, the matrix multiplication algorithm will take $2r(\log(N+1) - 1)/3 + 1$ time steps, and the minimum-weight spanning-tree algorithm will take $4r^2(\log(N+1) - 1)\log_{4/3} N + 1$ time steps.

Product of de Bruijn and shuffle-exchange networks. We can sort in their two-dimensional instances by using the embeddings of their respective factor networks, which will be presented in Chapter 7 and have constant dilation and congestion for bounded number of dimensions. Given the existence of algorithms to sort n keys in any hypercubic network in $O(\log^2 n)$ time, we can sort by emulation N^2 keys in a two-dimensional product network in $O(\log^2 N^2) = O(\log^2 N)$ time steps. Then, our algorithm will take $O(r^2 \log^2 N)$ time steps in these networks. Again, if r is bounded the expression simplifies to $O(\log^2 N)$.

The diameter of $S(N)$ is $2\log N - 1$ and the diameter of $D(N)$ is $\log N$. This value plus one is the time that any of the operations required by the other algorithms require. Then we obtain that their product take, respectively, $r(2\log N - 1) + 1$ and $r\log N + 1$ time steps to obtain the summation of N^r values, $2r(2\log N - 1)/3 + 1$ and $2r\log N/3 + 1$ time steps to multiply two matrices, and $4r^2(2\log N - 1)\log_{4/3} N + 1$ and $4r^2 \log N \log_{4/3} N + 1$ to find the minimum-weight spanning tree of a graph.

Hypercube. From the above analysis of the grid, and given that for the hypercube $N = 2$, the time to sort in Q_r with our algorithm is $O(r^2)$.

Similarly, the time to broadcast, communicate, perform a summation, or find the minimum in the factor graph is 2 time steps. That yields $r + 1$ time steps to perform the summation, $2r/3 + 1$ time steps to multiply matrices, and $4r^2 \log_{4/3} 2 + 1$ time steps to find the minimum-weight spanning tree of a graph in Q_r.

Table 5.1 summarizes the results derived in asymptotic form. The time complexities of the sorting algorithm marked with "*" are optimal if the number of dimensions is bounded. The time complexities of summations and matrix multiplications are optimal, and are also marked with "*".

- Send $w(i)$ and $L'(i)$ to the column root $(0, i)$.

And the value of $P(i)$ is obtained as follows.

- Multicast the values $w(i)$ and $L'(i)$ from the row roots $(i, 0)$ to every (i, j).
- Select nodes (i, j) such that $L(i) = L(j)$. These values were already in the nodes from the above process.
- Compute the minimum weight of the values $w(i)$ from the selected nodes in the column root $(0, j)$. Make $P(j)$ equal to the associated value $L'(i)$ received.
- Send $P(j)$ to the row root $(j, 0)$.

The analysis of the time steps taken by the whole process follows. In the process of obtaining $L'(i)$ we need to do a multicast in a $PG_{r/2}(N)$-subgraph, the computation of a minimum in a $PG_{r/2}(N)$-subgraph, and the communication between column and row roots. Therefore, this first process takes $\frac{r}{2}(\mathcal{B}(N) + \mathcal{M}(N) + 2\mathcal{C}(N))$.

The computation of $P(i)$ is similar and takes same amount of time. Adding these times with the pointer jumpings, the total time taken by the set of steps described above is:

$$r(\mathcal{B}(N) + \mathcal{M}(N) + 2\mathcal{C}(N)) + r(\mathcal{B}(N) + 3\mathcal{C}(N)) = r(2\mathcal{B}(N) + 5\mathcal{C}(N) + \mathcal{M}(N))$$

It is shown in [42, pp. 336-338] that this set of steps is repeated at most $\log_{4/3} n = \log_{4/3} N^{r/2}$ times. Therefore, the total time taken by the algorithms is:

$$\frac{r^2}{2} \log_{4/3} N (2\mathcal{B}(N) + 5\mathcal{C}(N) + \mathcal{M}(N))$$

5.6 Application to Specific Networks

Here we obtain the time taken by the presented algorithms in several product networks. These times are subsequently compiled in Table 5.1.

Since the concatenation of execution of several algorithms for the factor graph saves us some steps in the computation, we will apply a small trick to obtain the actual value of the time steps taken by an algorithm. We will consider in the formulas the value of $\mathcal{B}(N)$, $\Sigma(N)$, $\mathcal{C}(N)$, and $\mathcal{M}(N)$ one unit less than its actual value and we will add a unit to the final result obtained.

Grid. Schnorr and Shamir [67] showed that it is possible to sort N^2 keys in a 2-dimensional grid $PL_2(N)$ in $O(N)$ time steps. This value of $\mathcal{S}_2(N)$ implies that our sorting algorithm will take $O(r^2 N)$ time steps to sort N^r keys in $PL_r(N)$. If we consider the number of dimensions r bounded, this expression simplifies to $O(N)$.

5.5.2 The Minimum-Weight Spanning-Tree Algorithm

We will present now the algorithm in detail. The logic and the terminology is similar to that used in [42, pp. 325-338]. We will refer there for some of the details of the algorithm.

The algorithm works by grouping the nodes of the graph in sets, denoted as *supernodes*. Then, it iteratively searches the minimum-weight edge incident to every supernode, adds those edges to the list of edges of the tree, and joins the supernodes connected by these edges. Initially, the set of supernodes is simply the set of nodes of the graph.

In the minimum-weight spanning-tree algorithm we use an algorithm to find the minimum value of a set of values placed in the nodes of a product graph $PG_k(N)$. This problem is similar in structure to the summation problem, and the summation algorithm can be easily modified to obtain an algorithm for this problem. Then, if there is an algorithm to find the minimum in $G(N)$ in $\mathcal{M}(N)$ time, we can find the minimum in $PG_k(N)$ in $k\mathcal{M}(N)$ time.

In the physical implementation of the minimum-weight spanning-tree algorithm, each node (i,j) holds the weight $w_{i,j}$. Each (row and column) root $(i,0)$ and $(0,i)$ has a value $L(i)$ that is the identifier of the supernode to which node i belongs. This value is the smallest label of the nodes of the supernode. Additionally, each root will manipulate other values (for instance $P(i)$, $L'(i)$, or $P'(i)$.)

Initially, the value $L(i) = i$ and $P'(i) = i$. The algorithm repeats the following set of steps for each node i until all the nodes belong to the same supernode (all the values $L(i)$ are equal.)

- If the supernode is "available", obtain $L'(i)$ and $P(i)$.
- Execute one step of pointer jumping to obtain $P'(i) = P(P(i))$.
- If $P'(i) = i$ and $P(i) > i$ then i is the smallest label in the new supernode and we make $P(i) = i$.
- Execute one step of pointer jumping to obtain $P(L(i))$ and make it the new value $L(i)$.

We need to detail the first step a little more. We say that a supernode is *available* if and only if $P(i) = i$ and $P'(j) = i \Rightarrow L(j) = i$, for all j.

The value $L'(i)$ is obtained as follows.

- Multicast the value $L(i)$ from $(i,0)$ to every (i,j) and from $(0,i)$ to every (j,i), for $i,j = 0, ..., N^{r/2} - 1$.
- Select nodes (i,j) such that $L(i) \neq L(j)$. These are the two values received from the row and column roots.
- Compute the minimum of the weights $w(i)$ of the selected nodes (i,j) and its associated value $L(j)$ in $(i,0)$, for $i = 0, ..., N^{r/2} - 1$. Make $L'(i)$ equal to the value $L(j)$ received.

with the algorithm described in Section 5.2.1. It is easy to see that these communications follow edge-disjoint paths.

In the heart of the algorithm there is a process that we will call "pointer jumping."

5.5.1 Pointer-Jumping Algorithm

Let the nodes $(i,0)$ and $(0,i)$ both contain values $X(i)$ and $Y(i)$. The pointer jumping process computes each value $Z(i) = X(Y(i))$ and leaves it in both $(i,0)$ and $(0,i)$, for $i = 0, ..., N^{r/2} - 1$. This is a key operation of the minimum-weight spanning-tree algorithm.

The process is done in three basic steps.

1. Simultaneously, multicast the value $X(i)$ from $(i,0)$ to each node (i,k) and the value $Y(j)$ from $(0,j)$ to each node (k,j), for each $i,j,k = 0, ..., N^{r/2} - 1$. To do so we use the broadcasting algorithm presented in Section 5.2.2 in $PG_{r/2}(N)$-subgraphs of $PG_r(N)$. The two multicast operations are applied to different dimensions and, hence, there will be no contention. Each of them is actually the broadcasting in several disjoint copies of $PG_{r/2}(N)$.

2. Select the node (l,j) such that $l = Y(j)$, for each $j = 0, ..., N^{r/2} - 1$. The selected node (l,j) sends the value received from the row root to the column root $(0,j)$, for each $j = 0, ..., N^{r/2} - 1$. This can be done with multiple point-to-point communications in disjoint $PG_{r/2}(N)$-subgraphs of $PG_r(N)$.

3. Finally, each column root $(0,j)$ sends the obtained value $Z(j)$ to its corresponding row root $(j,0)$, for each $j = 0, ..., N^{r/2} - 1$.

The time taken by the pointer-jumping algorithm can be obtained very simply. If the time to broadcast in $G(N)$ from the node 0 is denoted as $\mathcal{B}(N)$, then step 1 takes $\frac{r}{2}\mathcal{B}(N)$, since it is a multicasting in $PG_{r/2}(N)$ subgraphs.

If $\mathcal{C}(N)$ is the maximum time taken by a point-to-point communication algorithm between the node 0 and any other node in $G(N)$, from the point-to-point algorithm presented in Section 5.2.2 we see that the point-to-point communication time between a root $(0,i)$ and any other node (k,i) of its column is at most $\frac{r}{2}\mathcal{C}(N)$. Similarly, the time of a point-to-point communication between a root of a row and a root of a column is at most $r\mathcal{C}(N)$.

Therefore, step 2 takes time $\frac{r}{2}\mathcal{C}(N)$ and step 3 takes time $r\mathcal{C}(N)$. The total time taken by this algorithm is, then,

$$\frac{r}{2}\mathcal{B}(N) + \frac{r}{2}\mathcal{C}(N) + r\mathcal{C}(N) = \frac{r}{2}(\mathcal{B}(N) + 3\mathcal{C}(N))$$

After this process, each node (i,k,j) contains the values $a_{i,k}$ and $b_{k,j}$, and it can compute the product of these values. All it remains to do is to add these values to obtain $c_{i,j}$. This is done by using the summation algorithm described in the previous section, to add all the products held in nodes (i,k,j), for $k = 0, ..., N^{r/3} - 1$, into the nodes $(i,0,j)$, for $i,j = 0, ..., N^{r/3} - 1$. This process is done by applying the summation algorithm to disjoint copies of $PG_{r/3}(N)$, subgraphs of $PG_r(N)$.

At the end of this process the product has been computed. The element $c_{i,j}$ of C is held in node $(i,0,j)$.

The total time taken by the execution of the algorithm depends on the time required by a broadcasting in $G(N)$ and a summation in $G(N)$. Let $\mathcal{B}(N)$ be the time to broadcast in $G(N)$ from the node 0, and $\Sigma(N)$ the time to obtain the summation in $G(N)$ into the node 0. From the above sections, we know that the time to broadcast from 0...0 in $PG_k(N)$ is $k\mathcal{B}(N)$, and the time to obtain the summation into 0...0 is $k\Sigma(N)$.

Therefore, to multicast the values of the matrices A and B to all the nodes takes $\frac{r}{3}\mathcal{B}(N)$ time and to compute the elements of C takes $\frac{r}{3}\Sigma(N)$ time. The total time taken by the algorithm is, hence,

$$\frac{r}{3}(\mathcal{B}(N) + \Sigma(N)).$$

Clearly, the algorithm can be used to multiply non-square matrices. The choice of dividing the labels of the $PG_r(N)$-nodes into three equal-length subtuples simplifies the analysis, but is not mandatory. The division of the labels can be done in such a way that it adapts the best to the specific dimensions of the matrices.

5.5 Minimum-Weight Spanning-Tree Algorithm

In this section we solve the problem of finding the minimum-weight spanning-tree in a graph described by its weight matrix W. Let us assume the nodes of the graph are labeled from 0 to a value $n-1$. The matrix W will have dimensions $n \times n$ and its element $w_{i,j}$ is the weight of the edge connecting node i to node j.

As we did in the previous section we assume that the set of nodes of $G(N)$ is $\{0, ..., N-1\}$. The maximum distance from a node to all the others is minimum for the node 0 and, therefore, the broadcasting time in $G(N)$ from 0 is minimum.

We assume that the number of dimensions of the product network, r, is even. Then, we divide each tuple of a node of $PG_r(N)$ into two subtuples of equal length, each seen as a N-ary number. Hence, we see the nodes of $PG_r(N)$ as labeled with a pair of values (i,j), where $i,j = 0, ..., N^{r/2} - 1$. Initially, node (i,j) holds the element $w_{i,j}$ of the weight matrix W.

We call the node $(i,0)$ the "root" of "row" i, for $i = 0, ..., N^{r/2} - 1$. Similarly, we call $(0,j)$ the root of "column" j, for $j = 0, ..., N^{r/2} - 1$. It will be very common to transfer information between the root of the row i, $(i,0)$, and the root of the column i, $(0,i)$, for $i = 0, ..., N^{r/2} - 1$. This can be done with simultaneous point-to-point routings obtained

lems have a similar structure as the summation problem (obtaining the maximum, the minimum, etc.) Algorithms for these problems can be obtained with straightforward modifications of the summation algorithm presented here. For instance, in Section 5.5 we use an algorithm to obtain the minimum of a set of values that is assumed to have the same structure as the presented summation algorithm.

5.4 Matrix-Multiplication Algorithm

This section is devoted to present an algorithm to perform the product of two $n \times n$ matrices in a n^3-node network. Let A and B be the matrices to be multiplied, then we want to obtain an algorithm that computes a product matrix C. If $a_{i,k}$ are the elements of matrix A, and $b_{k,j}$ are the elements of matrix B, for $i,k,j = 0,...,n-1$, then the element $c_{i,j}$ of C is obtained as

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

Let first introduce notation that will simplify the presentation. For the sake of simplicity, we will assume that the set of vertices of the graph $G(N)$ is $\{0,...,N-1\}$. The time taken by a broadcasting in $G(N)$ is assumed to be minimum if started from node 0 and, similarly, the time taken by a summation is assumed to be minimum if the final result is held in node 0. Then, the time taken by the broadcasting from, and the computation of the summation into, the node 0...0 is minimum in $PG_r(N)$.

We also assume that the number of dimensions of the product graph $PG_r(N)$ is a multiple of 3. Then, the label of each node of $PG_r(N)$ is a tuple of length multiple of 3, where each symbol of the tuple is between 0 and $N-1$.

We divide now each of the tuples into three subtuples of same length. Then, each node is considered labeled with a triple of subtuples, each of length $r/3$. Each subtuple is a sequence of symbols between 0 and $N-1$ and, therefore, can be considered as a N-ary number that, in decimal, has a value between 0 and $N^{r/3}-1$.

Therefore, by the above process, we have each node of $PG_r(N)$ labeled with a triple of values (i,k,j), where $i,k,j \in \{0,...,N^{r/3}-1\}$. Observe that two nodes with two of the elements i, k, j, equal are in a same $PG_{r/3}(N)$-subgraph of $PG_r(N)$.

Then, we assume that the element $a_{i,k}$ of the matrix A is initially held in node $(i,k,0)$ and the element $b_{k,j}$ of the matrix B is initially held in the node $(0,k,j)$, for $i,k,j = 0,...,N^{r/3}-1$.

The algorithm starts by simultaneously multicasting in the appropriate $PG_{r/3}(N)$ subgraphs the values of the elements of A and B. Then, $a_{i,k}$ is sent from node $(i,k,0)$ to all the nodes (i,k,j), for $j = 0,...,N^{r/3}-1$. Simultaneously, the value $b_{k,j}$ is sent from node $(0,k,j)$ to all the nodes (i,k,j), for $i = 0,...,N^{r/3}-1$. To do so we use the broadcasting algorithm presented in Section 5.2.2. Since different dimensions are used in each multicasting, no contention can be observed in the network.

5.2.2 Broadcasting Algorithm

Similarly, a broadcasting algorithm for $PG_r(N)$ can be simply derived from a broadcasting algorithm for $G(N)$. The algorithm chooses an order in the dimensions and applies the broadcasting in each of the $G(N)$-subgraphs of each dimension in that order. Then, if we want to broadcast from the node $x = x_r...x_1$ to all the nodes of $PG_r(N)$ by using the dimensions in descending order, we first broadcast from x to all the nodes in its dimension-r $G(N)$-subgraph. Then, we broadcast from all the nodes of this $G(N)$-subgraph to all the nodes in their dimension-$(r-1)$ $G(N)$-subgraphs, and so on.

Again, if the broadcasting algorithm for $G(N)$ takes optimal time, then this algorithm also takes optimal time.

5.3 Summation Algorithm

Here we present an algorithm to compute the summation of a set of N^r values in $PG_r(N)$. Initially, each value is in a different node of the network. At the end of the execution of the algorithm the value of the summation will be obtained in one given node of the network.

We initially assume the existence of a summation algorithm for $G(N)$. This algorithm computes the summation of the N values held in the N nodes of the network in $\Sigma(N)$ time steps, and leaves the result in a given node u of $G(N)$.

Our algorithm applies the algorithm for $G(N)$ to the $G(N)$-subgraphs in each dimension in some order. After this process, the desired value will be held in the node $x = x_r...x_1$, where $x_i = u$ for $i = 1, ..., r$. Since we apply r times the summation algorithm for $G(N)$, the time taken by this algorithm is $r\Sigma(N)$.

We can simply observe that the algorithm actually computes the desired summation. If we consider the dimensions in ascending order, we first apply the summation algorithm to all the dimension-1 $G(N)$-subgraphs. Then, after this step each node $y_r...y_2u$ will contain the summation of values of the corresponding dimension-1 $G(N)$-subgraph. We can then apply the summation algorithm to each dimension-2 $G(N)$-subgraph in the uth $PG_r^1(N)$ subgraph of $PG_r(N)$. In this step the summations obtained in the previous step are added up. At the end of this step each node $y_r...y_3uu$ will contain the summation of the values of a different $PG_r^{1,2}(N)$ subgraph.

We apply this process to each dimension. In the ith step we apply the summation algorithm for $G(N)$ to each dimension-i $G(N)$-subgraph of the $(u, ..., u)$th $PG_r^{1,...,i}(N)$ subgraph of $PG_r(N)$. At the end of this step each node $y_r...y_{i+1}u...u$ will contain the summation of the values of a different $PG_r^{1,...,i}(N)$ subgraph of $PG_r(N)$.

After r steps as presented, the summation of all the values will be obtained in the node $u...u$. All the computation performed has been the execution r times of the summation algorithm for $G(N)$ and, therefore, the total execution time of the algorithm is $r\Sigma(N)$.

This algorithm will be used in the following section. Observe that several prob-

Third, it is always possible to obtain an algorithm for $PG_2(N)$ with complexity $O(N)$, given that $G(N)$ is connected. To do so we simply emulate the 2-dimensional grid in $PG_2(N)$ by embedding the linear array onto each $G(N)$ subgraph as shown in Theorem 3.15 of [42] . Since this embedding has constant dilation and congestion, the emulation has constant slowdown [37]. Therefore, the $O(N)$-complexity algorithm presented by Schnorr and Shamir [67] can be emulated by $PG_2(N)$ with complexity $O(N)$. Hence, any arbitrary N^r-node r-dimensional product network can sort with complexity $O(r^2N)$.

The combination of these three results yields the proof of the corollary. ■

This corollary will be used in Section 5.6 to obtain the time complexity of this algorithm in several product networks.

5.2 Routing Algorithms

Many routing algorithms have been already presented for product networks. These algorithms cover most of the routing needs of a network under the SIMD model of computation. Therefore, we will not present new routing algorithms in this section and we refer the interested reader to the specific source (see Section 1.2.)

However, to simplify the reference, we briefly present here two of the simplest algorithms, presented in [86]. These algorithms perform point-to-point communication and broadcasting in the product network, respectively, and they will be used in the following sections.

5.2.1 Point-to-Point Routing Algorithm

If we assume the existence of a point-to-point routing algorithm for $G(N)$ the point-to-point routing algorithm in $PG_r(N)$ from a node x to a node y simply applies the algorithm for $G(N)$ to each dimension in which x and y differ, in some arbitrary order. For instance, if $x = x_r...x_1$ and $y = y_r...y_1$ differ in every symbol position and the algorithm is applied to the dimensions in descending order, then the path from x to y will be as follows:

$$x \rightarrow y_r x_{r-1}...x_1 \rightarrow ... \rightarrow y_r...y_2 x_1 \rightarrow y$$

Where the ith arrow represents the path defined by the algorithm for $G(N)$ in the corresponding dimension-i $G(N)$-subgraph from x_i to y_i, for $i = 1, ..., r$.

It is easy to see that, if the algorithm for $G(N)$ yields a shortest path, this algorithm also yields the shortest path between any to nodes of $PG_r(N)$. This fact follows from Observation 2.2, that implicitly states that the traversal of an edge in $PG_r(N)$ changes only one of the symbol positions of the node labels.

Therefore, the value of $\mathcal{M}_k(N)$ can be recursively expressed as:

$$\mathcal{M}_k(N) = \mathcal{M}_{k-1}(N) + 2\mathcal{S}_2(N) + 4\mathcal{R}(N)$$

with initial condition

$$\mathcal{M}_2(N) = \mathcal{S}_2(N)$$

that yields

$$\mathcal{M}_k(N) = 2(k-2)\mathcal{S}_2(N) + 4(k-2)\mathcal{R}(N) + \mathcal{S}_2(N)$$

■

We can now derive the value of $\mathcal{S}_r(N)$.

Theorem 5.1 *Sorting N^r keys in $PG_r(N)$ takes $\mathcal{S}_r(N) = (r-1)^2\mathcal{S}_2(N) + 2(r-1)(r-2)\mathcal{R}(N)$ time steps.*

Proof: The time taken to sort N^r keys in $PG_r(N)$ is the time taken to sort each 2-dimensional subgraph $PG_r^{1,\ldots,r-2}(N)$ and then merge blocks of N sorted sequences into increasing number of dimensions. The expression of this time is as follows:

$$\begin{aligned}\mathcal{S}_r(N) &= \mathcal{S}_2(N) + \mathcal{M}_3(N) + \mathcal{M}_4(N) + \ldots + \mathcal{M}_{r-1}(N) + \mathcal{M}_r(N) \\ &= (r-1)\mathcal{S}_2(N) + (2\mathcal{S}_2(N) + 4\mathcal{R}(N))\sum_{i=3}^{r}(i-2) \\ &= (r-1)^2\mathcal{S}_2(N) + 2(r-1)(r-2)\mathcal{R}(N)\end{aligned}$$

■

The following corollary presents the asymptotic complexity of the algorithm. Since $\mathcal{S}_2(N)$ may not be easy to obtain for an arbitrary network, the corollary uses upper bounds on this parameter to obtain expressions of the complexity only dependent on $\mathcal{S}(N)$.

Corollary 5.1 *The time complexity of sorting N^r keys in $PG_r(N)$ is at most $O(r^2\ min\{\mathcal{S}_2(N), \mathcal{S}(N)\log N, N\})$.*

Proof: First, since the value $\mathcal{S}_2(N)$ is never smaller than $\mathcal{R}(N)$, the time obtained in Theorem 5.1 is $\mathcal{S}_r(N) = O(r^2\mathcal{S}_2(N))$.

Second, it is trivial to obtain a sorting algorithm for $PG_2(N)$ that takes $O(\mathcal{S}(N)\log N)$ time steps, by simply generalizing the algorithm presented for the grid in [65] and [66]. This yields that, at worse, our algorithm has complexity $O(r^2\mathcal{S}(N)\log N)$ for any arbitrary network.

Step 2 can be simply implemented by selectively permuting the obtained subsequences along dimension 1. There are several possibilities for this permutation that are valid. We have presented one in the previous section. Following this option, the key in node $x_r...x_3ji$ will be routed to the node $x_r...x_3j((i+j)\ mod\ N)$. It can be easily seen that this respects the condition of each row having one subsequence from each of the original sequences and one subsequence $B_{i,N-1}$. Now each $PG_r^1(N)$ subgraph contains one subsequence from each original sequence.

The above routing has placed each subsequence in a different $PG_r^{1,2}(N)$ subgraph, each sorted in snakelike order. We can recursively merge these sequences into a sorted sequence of N^{r-1} keys. If the number of dimensions is $r-1=2$, this step is done by directly sorting with an algorithm for $PG_2(N)$. In the proof of Corollary 5.1 we present ways to obtain such an algorithm if it is not already available.

Step 4 is directly done by considering the dimension 1 of $PG_r(N)$ in the order. No movement of data is involved in this step and we obtain a sequence sorted almost completely.

The cleaning of the dirty area is done as described in above sections. We take each 2-dimensional subgraph $PG_r^{3,...,r}(N)$ and sort the keys in it using alternate orders in consecutive subgraphs (order depends on whether $p_3(x)$ is either odd or even.) We then perform two steps of odd-even transposition. In the first step we make the nodes with $p_3(x)$ odd exchange, if appropriate, their key with the corresponding node (same node) in the predecessor 2-dimensional subgraph and those with $p_3(x)$ even with the corresponding node in the successor 2-dimensional subgraph. In the second step the nodes with $p_3(x)$ odd exchange with the successor and those with $p_3(x)$ even exchange with the predecessor. A final sorting on each of the 2-dimensional subgraph ends the merge process.

Analysis of Time Complexity

To analyze the time taken by the algorithm we will initially study the time taken by the merge process in a k-dimensional network. This time will be denoted as $\mathcal{M}_k(N)$.

Lemma 5.3 *Merging N sorted sequences of N^{k-1} keys each in $PG_k(N)$ takes $\mathcal{M}_k(N) = 2(k-2)\mathcal{S}_2(N) + 4(k-2)\mathcal{R}(N) + \mathcal{S}_2(N)$ time steps.*

Proof: The time taken by step 1 of the merge process is just the time to reverse the order of the keys in a $G(N)$-subgraph. This process can be done with a permutation routing in $G(N)$, that takes time $\mathcal{R}(N)$. Similarly, step 2 can be done with a permutation routing along dimension 1.

Step 3 is a recursive call to the merge procedure for $k-1$ dimensions and, hence, will take $\mathcal{M}_{k-1}(N)$ time. Step 4 does not take any computation time. Finally, step 5 takes the time of one sorting in $PG_2(N)$, two permutation routings in $G(N)$, and one more sorting in $PG_2(N)$.

In this section we assume $G(N)$ be a connected graph, with vertex set $\{0, 1, ..., N-1\}$ and arbitrary edge set. For an arbitrary factor graph $G(N)$, vertex labels can define the ascending order of data when sorted. However we need to define an order in the nodes of $PG_r(N)$, which will determine the final location of the sorted keys. The order defined is known as "snakelike" order. For one dimension it is simply the order defined in $G(N)$, from 0 to $N-1$. Given an order for the nodes of $PG_{k-1}(N)$, the order for $PG_k(N)$ is defined as follows: if $u < v$ then any node in the uth subgraph $PG_k^k(N)$ precedes any node in the vth subgraph $PG_k^k(N)$. If $u = v$ then the uth subgraph $PG_k^k(N)$ has the same order defined as $PG_{k-1}(N)$ if u is even, and reverse order if u is odd.

The order defined guarantees several properties:

- Any two consecutive nodes always belong to a common $G(N)$ subgraph.
- Let $x = x_r...x_{i+1}x_ix_{i-1}...x_1$, where $x_i < N-1$, be a vertex of $PG_r(N)$ and let $p_j(x) = \sum_{k=j}^{r} x_k$. Then, x precedes $x_r...x_{i+1}(x_i+1)x_{i-1}...x_1$ if and only if $p_{i+1}(x)$ is even.
- Two consecutive $PG_r^{r-k,...,r}(N)$ subgraphs of $PG_r(N)$, for $k = 1, ..., r-1$, have reverse orders.

In the rest of this section we present the sorting algorithm for $PG_r(N)$. The heart of the algorithm resides in the multiway-merge process that takes N sorted sequences, placed in the N subgraphs $PG_k^1(N)$ and combines them into one sorted sequence in $PG_k(N)$. To do so, recursive calls to the merge process are used when necessary.

Once the merge process is available, the sorting is done by initially sorting sequences of N^2 elements placed in the $PG_r^{1,...,r-2}(N)$ subgraphs and iteratively merging groups of N sequences in larger sequences until only one sorted sequence remains. The reason for starting the iteration with sequences of length N^2 instead of N is in the nature of the merge process, since sorting them is faster than applying the merge step once more.

Implementation of the Multiway-Merge Algorithm in $PG_r(N)$

Now we present in detail the implementation of the multiway-merge algorithm in $PG_r(N)$. The initial scenario is N sequences, of N^{r-1} keys each, sorted in the N subgraphs $PG_r^1(N)$.

Step 1 of the merge process is done as follows. Reverse the order of each dimension-2 $G(N)$-subgraph such that $p_3(x)$ is odd. This makes all the dimension-2 $G(N)$-subgraphs sorted in non-decreasing order. After this process each $PG_r^1(N)$ subgraph is a snakelike-order sorted sequence of N-key sequences, each sorted in non-decreasing order. Then, the sequence can be divided in N subsequences by just fixing the second dimension to a value j, for $j = 0, ..., N-1$. The sequence $B_{i,j}$ is then contained in the (i,j)th subgraph $PG_r^{1,2}(N)$ and is already sorted in snakelike order.

One of the odd-even transpositions will not affect this distribution, while the other is going to move zeroes from the second sequence to the first and ones from the first to the second. Depending on whether there are more zeroes than ones or vice-versa in these two sequences, after these two steps H_i is filled with zeroes or H_{i+1} is filled with ones, respectively (see Figure 5.6.(c).) Therefore, only one sequence contains zeroes and ones combined. The last step of sorting will sort this sequence and the whole sequence J will be sorted (see Figure 5.6.(d).) ■

5.1.3 Sorting Algorithm

Using the above algorithm, and an algorithm to sort sequences of length N^2, it is very simple to obtain a sorting algorithm to sort a sequence of length N^r, for $r \geq 2$.

First divide the sequence in subsequences of length N^2 and sort each subsequence using the known algorithm. Then, iteratively apply the following process until only one sequence remains:

- Group all the sorted sequences obtained in sets of N sequences.
- Merge the sequences in each set into a larger sorted sequence using the algorithm shown in the previous section.

In the next section we show how to implement this algorithm in any homogeneous product network and we study the time complexity of the resultant general sorting algorithm.

5.1.4 Implementation in Homogeneous Product Networks

The purpose of this section is to obtain a general result of the form: "if the graph $G(N)$ can sort N keys in $f(N)$ time, then $PG_r(N)$ can sort N^r keys in $g(N)$ time." Once we obtain such a general result, we will then be able to tune the general algorithm for specific instances of product networks. Thus, it is reasonable to initially assume that there exists a sorting algorithm for $G(N)$. For example, since the hypercube Q_r is nothing but the r-dimensional product of 2-node linear arrays, the assumed sorting algorithm consists of a single step of compare-exchange operation. For other factor graphs, such as products of de Bruijn graphs, we can use the well-known Batcher's algorithm to sort the N values. In the rest of this paper, the time complexity of this sorting algorithm will be denoted as $\mathcal{S}(N)$. Similarly, we assume the existence of a permutation routing algorithm that takes $\mathcal{R}(N)$ time steps to execute any permutation in $G(N)$.

Before the sorting algorithm starts, each node of $PG_r(N)$ holds one of the keys to be sorted, and during and after the sorting only one key will be held in each node. The time taken to sort in $PG_r(N)$ will be denoted $\mathcal{S}_r(N)$.

for a window of keys of length at most $(N-1)N$.

Proof: Let z_i be the number of zeroes in sequence A_i, for $i = 0, ..., N-1$. The rest of elements in A_i are ones. Step 1 breaks each sequence A_i into N subsequences $B_{i,j}$, $j = 0, ..., N-1$. Given the nature of this process, the number of zeroes in a subsequence $B_{i,j}$ is $\lfloor z_i/N \rfloor + f_j(z_i)$, where $f_j(x) = 1$ if and only if $x \ mod \ N > j$. Observe that $f_{N-1}(x) = 0$ for any x.

After recombining the subsequences as defined in step 2, each row i, for $i = 0, ..., N-1$, has one subsequence from each of the original sequences. Also in each row there is a sequence of the form $B_{j,N-1}$, whose associated f function has a value $f_{N-1}(z_j) = 0$. Hence, the total number of zeroes in the sequences of one row is $\lfloor z_1/N \rfloor + \lfloor z_2/N \rfloor + ... + \lfloor z_N/N \rfloor + g_i$, where g_i is the summation of the values of f functions of the subsequences, and therefore can vary from 0 to $N-1$. Step 3 places all these zeroes at the beginning of a new sequence C_i.

In step 4 we interleave the N sorted sequences into D by taking one key from each sequence C_i at the time. Any two sequences C_i can differ in at most $N-1$ zeroes, since g_i can only vary from 0 to $N-1$, for $i = 0, ..., N-1$. Since the interleaving starts taking one key from the C_0 and ends with C_{N-1}, the worst case will occur when $g_0 = 0$ and $g_{N-1} = N-1$. In this case we will have a distance of $(N-1)N$ between the first 1 and the last 0, which defines the unsorted (dirty) area in the sequence D. ■

The worst case after step 4 is shown in Figure 5.5. In this figure, after $z = \lfloor z_1/N \rfloor + \lfloor z_2/N \rfloor + ... + \lfloor z_N/N \rfloor$ columns of zeroes we see that C_0 has $g_0 = 0$, while the last sequence C_{N-1} has $g_{N-1} = N-1$. That yields a dirty area that spans along $N-1$ columns of N keys each, as shown.

Now we can show how the last step actually cleans the dirty area in the sequence.

Lemma 5.2 *The sequence J, obtained after the completion of step 5, is sorted.*

Proof: We know that the dirty area of the sequence D, obtained in step 4, has at most length $(N-1)N$. If we divide the sequence D in consecutive subsequences of N^2 keys, E_i, the dirty area can either fit in exactly one of these subsequences or be distributed between two adjacent subsequences.

If the dirty area fits in one subsequence E_i, after the initial sorting and the odd-even transpositions the sequence H_j contains exactly the same keys than the sequence E_j, for $j = 0, ..., N^{r-2}$. Then, the last sorting in each sequence H_j and the final concatenation yield a sorted sequence J.

However, if the dirty area is distributed between two adjacent subsequences E_i and E_{i+1}, we have two subsequences with zeroes and ones combined. Figure 5.6.(a) presents an example of this initial situation. After the first sorting, the zeroes are located in one side in sequence F_i and in the other side in sequence F_{i+1} (see Figure 5.6.(b).)

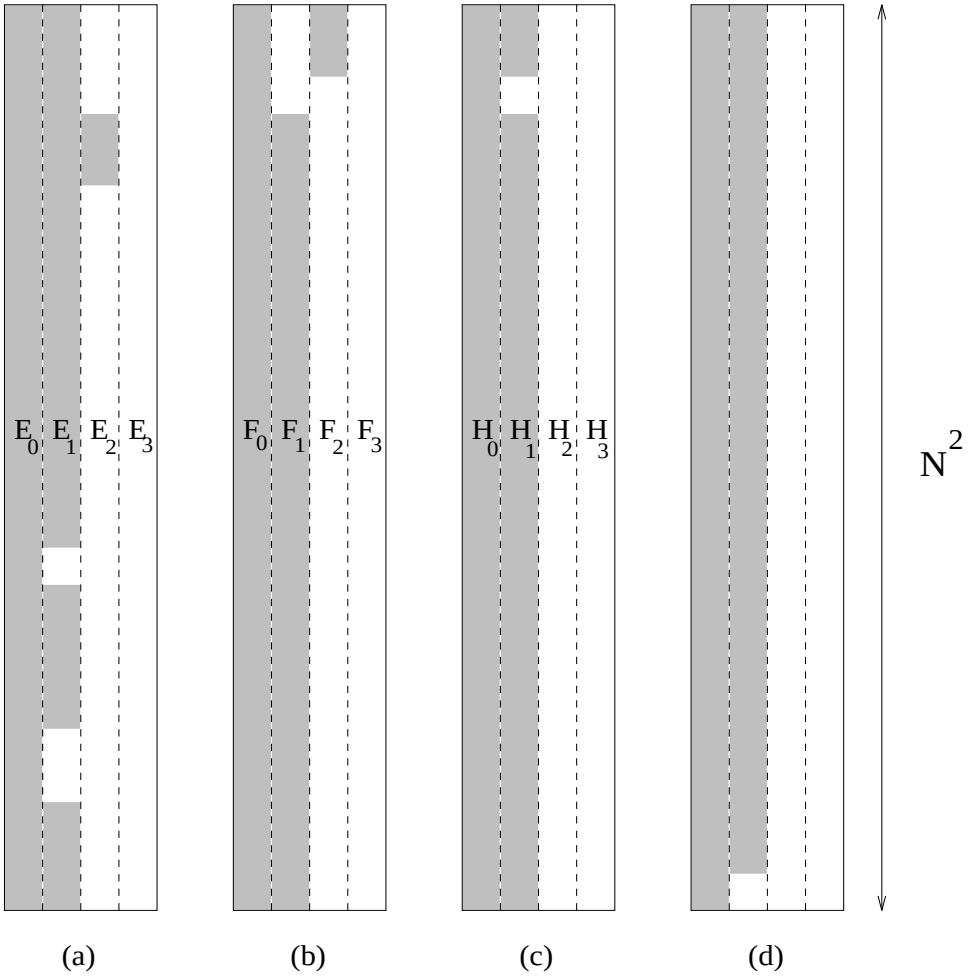

Figure 5.6: Clearing of the dirty area.

This step is illustrated in Figure 5.6. In this figure we take groups of N adjacent columns in Figure 5.5 and place them in single columns as initial situation. Then, each column in Figure 5.6.(a) is one subsequence E_i, for $i = 0, ..., N^{r-2} - 1$. In the figure we present the dirty area divided between two columns. Figure 5.6.(b) presents the sequences F_i obtained after sorting the columns in alternate orders. The situation after the steps of odd-even transposition is shown in Figure 5.6.(c), and Figure 5.6.(d) presents the final sorted sequence.

We need to show that the described process actually merges the sequences. To do so we use the zero-one principle which allows us to assume that the domain of keys to be sorted is $\{0, 1\}$, and to generalize the observed properties to any domain of keys.

The first property is stated as a lemma.

Lemma 5.1 *The sequence D, obtained after the completion of step 4, is sorted except*

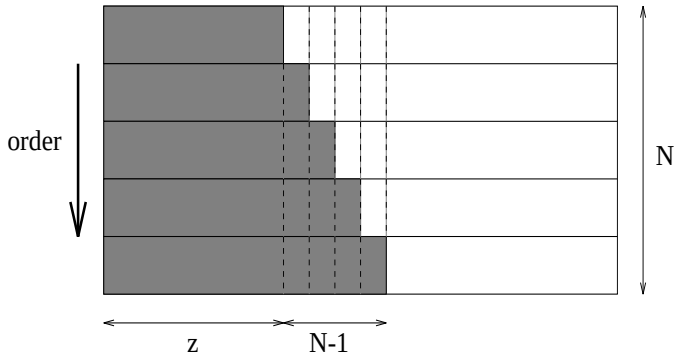

Figure 5.5: Sequence D obtained after interleaving. The order goes from left to right taking each column from top to bottom. The shaded area is filled with zeroes and the white area with ones. The boundary area has at most $N-1$ columns, as shown.

sorted sequences from top to bottom in Figure 5.4.

We prove below that D is almost sorted, since there is a potential dirty area (window of keys not sorted) of length at most $N(N-1)$. This situation is shown in Figure 5.5, where having completed step 3, the sorted sequences are interleaved by following the vertical dimension one column at a time from left to right.

5. Clean the dirty area. To do so we start by dividing the whole sequence $D = \langle d_0, d_1, ..., d_{N^r-1} \rangle$ into N^{r-2} subsequences of N^2 consecutive keys each. If we denote these subsequences as E_i, for $i = 0, ..., N^{r-2}-1$, the ith subsequence has the form $E_i = \langle d_{iN^2}, d_{iN^2+1}, ..., d_{iN^2+N^2-1} \rangle$.

   Then, we sort the subsequences in alternate orders, *i.e.* E_i is transformed into a sequence F_i, where F_i contains the keys of E_i sorted in non-decreasing order if i is even or in non-increasing order if i is odd, for $i = 0, ..., N^{r-2}-1$.

   Now, we apply two steps of odd-even transposition between the sequences F_i, for $i = 0, ..., N^{r-2}-1$. In the first step, each pair of sequences F_i and F_{i+1}, for i even, is transformed into two sequences $G_i = \langle min\{f_{i,0}, f_{i+1,0}\}, min\{f_{i,1}, f_{i+1,1}\}, ..., min\{f_{i,N^2-1}, f_{i+1,N^2-1}\} \rangle$ and $G_{i+1} = \langle max\{f_{i,0}, f_{i+1,0}\}, max\{f_{i,1}, f_{i+1,1}\}, ..., max\{f_{i,N^2-1}, f_{i+1,N^2-1}\} \rangle$. In the second step, each pair of sequences G_i and G_{i+1}, for i odd, is transformed into two sequences $H_i = \langle min\{g_{i,0}, g_{i+1,0}\}, min\{g_{i,1}, g_{i+1,1}\}, ..., min\{g_{i,N^2-1}, g_{i+1,N^2-1}\} \rangle$ and $H_{i+1} = \langle max\{g_{i,0}, g_{i+1,0}\}, max\{g_{i,1}, g_{i+1,1}\}, ..., max\{g_{i,N^2-1}, g_{i+1,N^2-1}\} \rangle$.

   Finally, we sort each sequence H_i in non-decreasing order, generating sequences I_i, for $i = 0, ..., N^{r-2}-1$, and concatenate all these sequences into a single sorted sequence $J = \langle i_{0,0}, i_{0,1}, ..., i_{0,N^2-1}, i_{1,0}, i_{1,1}, ..., i_{1,N^2-1}, ..., i_{N^{r-2}-1,0}, i_{N^{r-2}-1,1}, i_{N^{r-2}-1,N^2-1} \rangle$.

$B_{0,0}$	$B_{0,1}$	$B_{0,2}$	...	$B_{0,N-1}$
$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	...	$B_{1,N-1}$
$B_{2,0}$	$B_{2,1}$	$B_{2,2}$	...	$B_{2,N-1}$
...	...	...	...	...
$B_{N-1,0}$	$B_{N-1,1}$	$B_{N-1,2}$	...	$B_{N-1,N-1}$

Figure 5.2: Situation after step 1: each sequence A_i, $i = 0, ..., N-1$, has been distributed into N subsequences $B_{i,j}$, $j = 0, ..., N-1$. Each of the subsequences is still sorted.

$B_{0,0}$	$B_{N-1,1}$	$B_{N-2,2}$	...	$B_{1,N-1}$
$B_{1,0}$	$B_{0,1}$	$B_{N-1,2}$	...	$B_{2,N-1}$
$B_{2,0}$	$B_{1,1}$	$B_{0,2}$	...	$B_{3,N-1}$
...	...	...	...	...
$B_{N-1,0}$	$B_{N-2,1}$	$B_{N-3,2}$	...	$B_{0,N-1}$

Figure 5.3: Situation after the recombination of subsequences done in step 2.

sorting algorithm for sequences of length N^2 can be used, because a recursive call to the merge process would be more costly in time. The situation after this step is illustrated with Figure 5.4.

4. Interleave the obtained sorted sequences $C_i = \langle c_{i,0}, c_{i,1}, ..., c_{i,N^{r-1}-1} \rangle$, for $i = 0, ..., N-1$, into one single sequence D by alternatively taking one key from each sequence. Then, the sequence D will have the form $\langle c_{0,0}, c_{1,0}, ..., c_{N-1,0}, c_{0,1}, c_{1,1}, ..., c_{N-1,1}, ..., c_{0,N^{r-1}-1}, c_{1,N^{r-1}-1}, ..., c_{N-1,N^{r-1}-1} \rangle$. This is equivalent to reading the

C_0
C_1
C_2
. . . . .
C_{N-1}

Figure 5.4: Situation after merging the subsequences in each row.

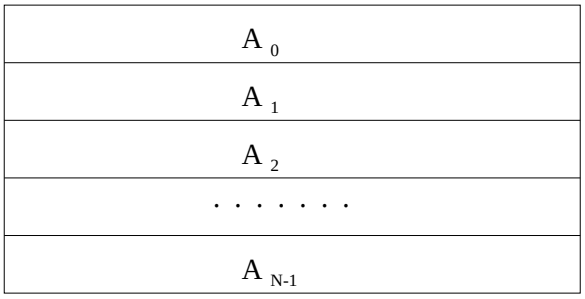

Figure 5.1: Initial situation before the merge process starts. Each sorted sequence is represented as a horizontal block (a row.)

To show the correctness of the proofs we will use the zero-one principle due to Knuth [36]. The zero-one principle states that if an algorithm based on compare-exchange operations is able so sort any sequence of 0's and 1's, then it sorts any arbitrary sequence of keys. Thus, we will assume that we are only dealing with sequences of zeroes and ones.

5.1.2 Multiway-Merge Algorithm

In this section we describe the algorithm to merge N sorted sequences, $A_i = \langle a_{i,0}, a_{i,1}, ..., a_{i,N^{r-1}-1} \rangle$, for $i = 0, ..., N-1$, into a large sorted sequence. The initial situation is pictured in Figure 5.1.

The merge process is implemented in the following steps:

1. Distribute the keys in each sorted sequence A_i into N sorted subsequences $B_{i,j}$, for $i = 0, ..., N-1$ and $j = 0, ..., N-1$. Subsequence $B_{i,j}$ has the form $\langle a_{i,j}, a_{i,j+N}, a_{i,j+2N}, ..., a_{i,j+(N^{r-2}-1)N} \rangle$, for $i = 0, ..., N-1$ and $j = 0, ..., N-1$. Note that the obtained subsequences are sorted, since all the keys in one subsequence $B_{i,j}$ come from the same sorted sequence A_i and are placed in the subsequence in the same order. In Figure 5.2 we illustrate the situation after the completion of this process. Each of the N rows contains N sorted subsequences.

2. Recombine the subsequences in the rows, so that each row has exactly one subsequence from each original sequence and exactly one subsequence of the form $B_{i,N-1}$. This can be done, for example, by cyclicly rotating the jth column of subsequences in Figure 5.2 by j positions, for $j = 0, ..., N-1$. As a results, the ith row, for $i = 0, ..., N-1$, will contain the subsequences $B_{i,0}, B_{(i-1) \bmod N,1}, ..., B_{(i-N+1) \bmod N,N-1}$. The result of this process is illustrated in Figure 5.3.

3. Merge the N subsequences in each row i into a single sorted sequence C_i, for $i = 0, ..., N-1$. This is done with a recursive call to the multiway-merge process if the number of keys in the row, N^{r-1}, is at least N^3. If the number of keys is N^2 a

affect only in a small amount to the final result, and will not affect the asymptotic analysis at any point. However, in order to obtain exact values when applying the algorithms to concrete networks, we will consider this fact and count the steps accurately.

Finally, in many algorithms we will assume that a node can simultaneously manipulate messages through all its communication links. This model of communication is known as the *multiport model*, and will be assumed in the rest of this chapter.

5.1 Sorting Algorithm

In this section we develop an algorithm to merge N sorted sequences into a single sorted sequence. We call this operation a "multiway merge." From the multiway-merge operation we derive a sorting algorithm, and we show how to use it to obtain an efficient sorting algorithm for any homogeneous product network. The obtained algorithms run very efficiently on product networks since their underlying structure is very well-suited for product networks.

We start by giving some specific definitions and notation for this section only. Then, we present our multiway merge algorithm and show how to use it for sorting. We continue by showing how to implement the multiway merge sorting algorithm in any homogeneous product network and analyze its time complexity.

5.1.1 Definitions and Notation

A sorted sequence is defined as a sequence of keys $\langle a_0, a_1, ..., a_{n-1} \rangle$ such that $a_0 \leq a_1 \leq ... \leq a_{n-1}$. The multiway-merge algorithm combines N sorted sequences $A_i = \langle a_{i,0}, a_{i,1}, ..., a_{i,n-1} \rangle$, for $i = 0, ..., N-1$, into a single sorted sequence $J = \langle j_0, j_1, ..., j_{nN-1} \rangle$. For simplicity, we assume n to be a power of N, N^{r-1}, where $r > 2$.

In order to build an intuitive understanding of the basic idea of the merge operation, we assume that the keys to be sorted are placed in a two dimensional area, as shown in Figure 5.1. Then, in the proof we can meaningfully use the terms row and column when referring to groups of keys. For instance, we initially assume that each sorted sequence A_i, for $i = 0, ..., N-1$, is in a different row (see Figure 5.1.)

To merge N sequences of N^{r-1} keys each, we initially assume the existence of an algorithm which can sort N^2 keys. We make no assumption in regards to the efficiency of this algorithm as yet. We might note here that our assumption is not due to a limitation of the proposed approach, because we can recursively apply the same algorithm to sequences of length N^2 taking a new of number of sequences to be merged (for instance $M = \sqrt{N}$) until the sequence length reduces to $O(1)$ keys which will then take constant time to sort on every network. The purpose of this assumption is to maintain the generality of discussions independent of the method used to implement this process. For instance, in our networks we assume this operation as basic, and obtain the time complexity based on the time of performing this sorting.

Chapter 5

Algorithms

In this chapter we present several algorithms that run efficiently in homogeneous product networks. One of the most important properties of all these algorithms is that they only depend on a few characteristics of the factor graph. All of them can be executed in any homogeneous product network without modification.

We start by presenting a sorting algorithm based on Batcher's odd-even merge sorting algorithm. Here we do a non-trivial generalization of its odd-even merge to obtain an algorithm very well-suited to product networks.

We follow by recalling two simple known routing algorithms, that will be used in other algorithms developed here. Then, an algorithm to compute the summation of several values is presented. This algorithm can be easily modified to solve other problems with similar structure.

Then, we derive a matrix-multiplication algorithm that uses the broadcasting and summation algorithms mentioned. Finally, we generate a minimum-weight spanning-tree algorithm, that uses an algorithm to find the minimum of a set of values trivially derived from the summation algorithm.

After the application of the algorithms generated to several product networks we find that they perform very efficiently in most of the cases, reaching the asymptotic time complexity of the best practical algorithms known.

One execution step in an algorithm involves the input of some values into each node through the links incident to it, some computation done in the node, and the output of some values through the node links. Of course, the number of values input or output in one step is bounded, and the size of each is also bounded. Therefore, the time taken by the step is considered constant.

In many of the algorithms we start with simple algorithms for the factor network and apply them in order to compose the desired algorithm. Note that the last step of one algorithm and the first of the following can be combined and counted as a single step in the overall algorithm. In order to simplify the expressions we will not consider this fact when obtaining the general expressions of the time complexity of the algorithm. This will

Corollary 4.4 *$PR_r(N)$ is a subgraph of $PD_r(N)$, $PC_r(N)$, or $PB_r(N)$.*

The same theorem can be used to show this corollary, from the existence of a hamiltonian path in the shuffle-exchange graph [25].

Corollary 4.5 *$PL_r(N)$ is a subgraph of $PS_r(N)$.*

Since we know that the cube-connected cycles, $C(N)$, is a subgraph of the wrapped butterfly, $B(N)$, by using the same theorem, we obtain that $PC_r(N)$ is a subgraph of $PB_r(N)$.

There are embeddings between $S(N)$ and $D(N)$ with dilation 2 and congestion 2. Then, the use of corollaries 4.1 and 4.2 yields the following.

Corollary 4.6 *$PS_r(N)$ can be embedded onto $PD_r(N)$, and vice-versa, with dilation 2 and congestion 2.*

This corollary shows that the products of these networks are computationally equivalent, the same way the factor networks are. Let us consider now the product of complete binary trees, $PT_r(N)$. Corollary 4.3 gives us that the torus can be embedded onto this network with dilation 3 and congestion 2. If we map the nodes of $T(N)$ into the nodes of $D(N+1)$ following the labeling shown in Definition 2.8 and Figure 2.6, we see that the former is a subgraph of the latter. Therefore, $PT_r(N)$ trees is a subgraph of $PD_r(N+1)$.

Observe that embeddings imply other embeddings. For instance, this last mentioned fact implies that $PT_r(N)$ can be embedded into $PS_r(N+1)$, since $PD_r(N+1)$ can.

We want to note that sometimes it is interesting to work with instances of product networks and consider them as factor graphs to obtain some results. This is very useful, for instance, if we are dealing with the hypercube.

Corollary 4.7 *$PT_r(N)$ can be embedded into the $(r\log(N+1))$-dimensional hypercube with dilation 2 and congestion 1.*

This result is a direct consequence of the existence of a dilation-2 congestion-1 embedding of the complete binary tree into the hypercube [31].

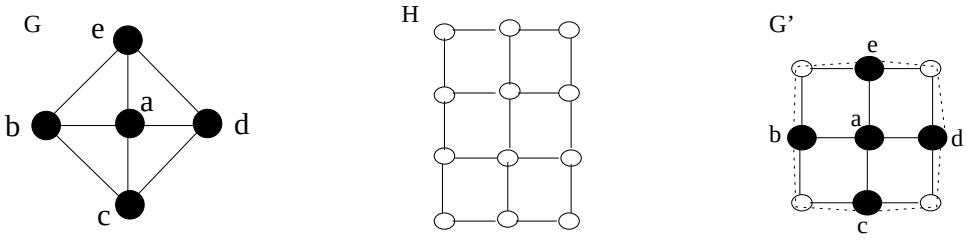

Figure 4.1: Obtaining $G'(N_{G'})$ from $G(N_G)$

Since $G'(N_{G'})$ obtained this way is a subgraph of $H(N_H)$, apply the above theorem. ■

Corollary 4.2 *If $G(N_G)$ can be embedded into $H(N_H)$ with congestion c, then $PG_r(N_G)$ can be embedded into $PH_r(N_H)$ with congestion c.*

Proof: An embedding of $G(N_G)$ into $H(N_H)$ with congestion cost c directly induces the claimed embedding for $PG_r(N_G)$ in $PH_r(N_H)$. ■

Note that the expansion of the embedding between factor graphs is N_H/N_G. The presented embeddings have, then, expansion $(N_H/N_G)^r$, that grows exponentially with the number of dimensions. However, in most of the embeddings found in this research the expansion of the embeddings between factor graphs is practically 1, and the resulting expansion for product graphs is also practically 1.

The above results can be used to obtain very powerful results for any homogeneous product graph. For instance, we can use the embedding of the N-node ring onto any N-node connected graph presented in Theorem 3.15 of [42] to show the following corollary.

Corollary 4.3 *If $G(N)$ is connected, then the N^r-node r-dimensional torus can be embedded onto $PG_r(N)$ with dilation 3 and congestion 2.*

Observe that this result presents that practically any product network can efficiently emulate the torus. This gives an idea of the power that any product network has just by belonging to this class.

4.2 Application to Specific Networks

We apply now the above results to prove embeddings between product networks. Many others can be presented, but we do not attempt to make the list exhaustive. This compilation simply shows the power of the above simple results.

Since the de Bruijn, cube-connected cycles, and wrapped butterfly graphs contain a hamiltonian cycle, it is directly implied by Theorem 4.1 that they contain the torus as subgraph.

Chapter 4

Embedding Properties

The embedding results of this research are among the most important results since they show a way of emulating one network by another. In the context of product networks, the utility of embedding results is further emphasized by the fact that many of the existing popular architectures can be modeled as product networks.

We start by showing simple but powerful results. Subsequently, we apply them to prove several embedding results involving specific homogeneous product networks.

4.1 General Results

The following is one of the most important results of this section.

Theorem 4.1 *$PG_r(N_G)$ is a subgraph of $PH_r(N_H)$ if and only if $G(N_G)$ is a subgraph of $H(N_H)$.*

Proof: The sufficient condition has been shown before (Lemma 3.3 in [42].) Therefore, we only focus on the necessary condition. If $G(N_G)$ is not a subgraph of $H(N_H)$, then there must be at least one edge (u, v) in $G(N_G)$ which cannot be mapped to any edge in $H(N_H)$. Since $PG_r(N_G) = G(N_G) \otimes PG_{r-1}(N_G)$, we can write an edge of $PG_r(N_G)$ as (ux, vx) where x is a vertex in $PG_{r-1}(N_G)$. Similarly, $PH_r(N_H) = H(N_H) \otimes PH_{r-1}(N_H)$, and the edge (ux, vx) cannot exist in $PH_r(N_H)$ since (u, v) is not an edge in $H(N_H)$. ■

This theorem and its extensions have many significant implications. In particular, the next two results have a wide variety of possible applications.

Corollary 4.1 *If $G(N_G)$ can be embedded into $H(N_H)$ with dilation d, then $PG_r(N_G)$ can be embedded into $PH_r(N_H)$ with dilation d.*

Proof: Modify $G(N_G)$ to obtain $G'(N_{G'})$, such that whenever an edge of $G(N_G)$ is mapped to a path of $H(N_H)$, replace it for the path it is mapped to (see Figure 4.1.)

Factor netw.	Nodes	Edges	Diameter	Conn.	δ, Δ
$L(N)$	N^r	$r(N-1)N^{r-1}$	$r(N-1)$	r	r, $2r$
$T(N)$	N^r	$r(N-1)N^{r-1}$	$2r(\log(N+1)-1)$	r	r, $3r$
$S(N)$	N^r	$3rN^r/2$	$r(2\log N-1)$	r	$3r$
$D(N)$	N^r	$2rN^r$	$r\log N$	$2r$	$4r$
$B(N)$	N^r	$2rN^r$	$\Theta(r\log N)$	$4r$	$4r$
$C(N)$	N^r	$3rN^r/2$	$\Theta(r\log N)$	$3r$	$3r$
Q_1	2^r	$r2^{r-1}$	r	r	r
$P(10)$	10^r	$15r10^{r-1}$	$2r$	$3r$	$3r$
$K(N)$	N^r	$r(N-1)N^r/2$	r	$(N-1)r$	$(N-1)r$

Factor netw.	Partitionability	Max. cong.	Bis. width	Cr. Number
$L(N)$	$\lfloor N/i \rfloor$, $i=1,...,N$	$O(N^{r+1})$	$\Theta(N^{r-1})$	$\Omega(N^{2(r-1)})$
$T(N)$	2^i, $i=0,...,\log(N+1)$	$N^{r-1}(N^2-1)/2$	$\Theta(N^{r-1})$	$\Omega(N^{2(r-1)})$
$S(N)$	-	$O(N^r\log N)$	$\Theta(N^r/\log N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$
$D(N)$	-	$O(N^r\log N)$	$\Theta(N^r/\log N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$
$B(N)$, $N=n2^n$	2^i, $i=0,...,n$	$O(N^r\log N)$	$\Theta(N^r/\log N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$
$C(N)$, $N=n2^n$	2^i, $i=0,...,n$	$O(N^r\log N)$	$\Theta(N^r/\log N)$	$\Omega(\frac{N^{2r}}{\log^2 N})$
Q_1	-	2^r	2^{r-1}	$\Omega(2^{2(r-1)})$
$P(10)$	-	10^r	$10^r/2$	$\Omega(10^{2(r-1)})$
$K(N)$	$\lfloor N/i \rfloor$, $i=1,...,N$	$2N^{r-1}$	$\Theta(N^{r+1})$	$\Omega(N^{2(r+1)})$

Table 3.3: Structural parameter of several homogeneous product networks obtained by application of the presented results.

width is 1 and the value B' for the case N odd is also 1. We use these to obtain the rest of the properties of the grid.

A complete binary tree $T(N)$, has $N-1$ edges, diameter of $2(\log(N+1)-1)$, connectivity of 1, minimum vertex degree of 1, maximum vertex degree of 3, and it is 2^i-partitionable, for $i = 0, ..., \log(N+1)$. The maximal congestion is $(N^2-1)/2$, the bisection width is 1, and the value B' is 1.

In the shuffle-exchange graph $S(N)$, every vertex has a degree of 3, although some edges are self loops. Any two nodes are connected by at least one path, although many pairs of vertices are connected by up to three vertex disjoint paths. The diameter is $2\log N - 1$ and the bisection width is $\Theta(N/\log N)$. From the analysis conducted in [42] to obtain its bisection width we can conclude that its maximal congestion is $O(N \log N)$.

Every vertex of the de Bruijn graph $D(N)$ has degree 4. Every pair of nodes are connected by at least two vertex disjoint paths (since de Bruijn graph contains a hamiltonian cycle), although most pairs of vertices are connected by up to four vertex-disjoint paths. The diameter is $\log N$ and the bisection width is $\Theta(N/\log N)$. The maximal congestion is $O(N \log N)$. Neither this network nor the shuffle-exchange are partitionable.

We now consider the cube-connected cycles, $C(N)$, and the wrapped butterfly, $B(N)$, were $N = n2^n$. Both have diameter $\Theta(\log N)$, are 2^i-partitionable, for $i = 0, ..., n$ (n is the number of levels), have bisection width $\Theta(N/\log N)$, and maximal congestion of $O(N \log N)$. $C(N)$ has vertex degree of 3 in each node and connectivity of 3, while the wrapped butterfly $B(N)$ has vertex degree and connectivity of 4.

The Petersen graph $P(10)$ has 10 nodes and 15 edges. Its diameter is 2, connectivity is 3, and vertex degree is 3. The bisection width is 5 and the maximal congestion is 10.

The complete graph $K(N)$ has $N-1$ edges, diameter of 1, connectivity of $N-1$, and vertex degree of $N-1$, it is $\lfloor N/i \rfloor$-partitionable, for $i = 1, ..., N$, has bisection width $\Theta(N^2)$, and has maximal congestion of 2.

All these values have been used to compose Table 3.3 by direct application of the general results presented in this chapter.

Note the interesting structural properties that each single product network presents. If the factor network has logarithmic diameter the product network also has logarithmic diameter. When the diameter is linear in the factor network (see the linear array), in the product version it tends to be logarithmic for a large enough number of dimensions.

The connectivity of all the product networks is at least r. The vertex degree is also a function of r. If we maintain the value of r bounded by a large constant number we have the advantages of a network with bounded vertex degree and large connectivity.

It can be noted also that the bisection width of the networks is large if r is reasonably large. The results presented allow us to obtain the exact value of the bisection width for two of the cases (actually, for N even, we also obtain exact values for grids and products of complete graphs.)

Property	Value in $PG_r(N)$
Maximal congestion	At least CN^{r-1}
Bisection width	At most BN^{r-1} (N even)
	At most $B + B/2\sum_{i=1}^{r-1}(N^i+1) + B'/2\sum_{i=1}^{r-1}(N^i-1)$ (N odd)
	At least $N^{r+1}/2C$ (N even)
	At least $(N^{2r}-1)/2CN^{r-1}$ (N odd)
Crossing number	At least $(N^r-1)(N^r-2)(N^r-3)/(20C^2N^{r-2}) - (rEN^{r-1})/2$

Table 3.2: Advanced structural properties of $PG_r(N)$ obtained from the values of the maximal congestion, C, the bisection width, B, and B' of $G(N)$.

We know from Theorem 3.6 that the maximal congestion of $PG_r(N)$ is at most CN^{r-1} and we have seen earlier in this chapter that it has rEN^{r-1} edges. Using these, we obtain the claimed lower bound on the crossing number of $PG_r(N)$. ■

The bounds obtained with this theorem are the same for the hypercube as those obtained in [76], but our result is obtained much simpler than those in [76]. For several networks we show that the bounds we obtain are asymptotically tight, since we are able to generate layouts for these networks with the same asymptotical layout area (see Chapter 6.)

The results of this section have been collected in Table 3.2.

3.3 Application to Specific Networks

Here we apply all the obtained results to several specific networks, identified by their factor graph. The results for the product networks are obtained from the properties of their factor graphs. The results are summarized in Table 3.3.

We have shown that every product graph is N^i-partitionable, for $i = 0, ..., r$, then we do not include this property in the table. Only specific partitionability characteristics are included there. In order to simplify some entries, we also express some results in asymptotic notation. Specifically, the bisection width and the crossing number are mostly expressed in this notation in the table.

We start with the linear array, $L(N)$. It has $N-1$ edges, diameter of $N-1$, connectivity of 1, minimum vertex degree of 1, maximum vertex degree of 2, and it is $\lfloor N/i \rfloor$-partitionable, for $i = 1, ..., N$ ($\lfloor N/i \rfloor$ is the number of subarrays of i nodes that can be obtained from the linear array.) These properties yield the first set of structural properties of the grid. The maximal congestion is at most $N^2/2$, and is found in the central edges of the array once we map the directed complete graph edges. The bisection

is $f(1) = B$. The solution of this recurrence yields the claim. ■

However, to obtain lower bounds on the bisection width is not that easy. The value of the bisection width of $G(N)$ does not carry enough information to be able to derive a lower bound on the value of the bisection width of $PG_r(N)$. Therefore we need to use a stronger parameter. This will be the the maximal congestion of $G(N)$.

The use of the maximal congestion allows us to derive the following result. The proof technique is similar to that used in [42] to obtain lower bounds on the bisection width of other networks.

Theorem 3.9 *If the maximal congestion of $G(N)$ is C, then $PG_r(N)$ has bisection width at least $\frac{N^{r+1}}{2C}$ if N is even, and at least $\frac{N^{2r}-1}{2CN^{r-1}}$ if N is odd.*

Proof: The bisection width of the N^r-node directed complete graph is $N^{2r}/2$ if N is even, and $(N^{2r}-1)/2$ if N is odd. Since we can embed it onto $PG_r(N)$ with congestion at most CN^{r-1}, the bisection width of $PG_r(N)$ has to be at least $\frac{N^{2r}/2}{CN^{r-1}} = N^{r+1}/2C$ if N is even or $(N^{2r}-1)/2CN^{r-1}$ if N is odd, because otherwise we could bisect the embedded directed complete graph by removing less edges than its bisection width, which is a contradiction. ■

Observe that the lower bound just presented and the upper bound presented in Theorem 3.7 are the same (and therefore both tight) if $B = N^2/2C$. However, when N is odd we can not guarantee that the bounds obtained are tight. To obtain the exact value of the bisection width of $PG_r(N)$ when N is odd is a difficult task, since it is not even known for such a simple network as the multidimensional grid [42]. For $PL_2(N)$, when N is odd, it is known that the lower bound obtained by Theorem 3.9 is not tight, while the upper bound obtained by Theorem 3.8 is.

3.2.3 Crossing Number

We now investigate the crossing number of homogeneous product networks. Since we only use the value of this parameter to derive lower bounds on the layout area, we only derive lower bounds on its value. The method used is similar to the method introduced in [41], and uses the maximal congestion. Again, we could not generate these bounds from the value of the crossing number of $G(N)$.

Theorem 3.10 *If $G(N)$ has E edges and its maximal congestion is C, then the crossing number of $PG_r(N)$ is at least $\frac{(N^r-1)(N^r-2)(N^r-3)}{20C^2N^{r-2}} - \frac{rEN^{r-1}}{2}$*

Proof: From the results in [34, 35, 39] it was shown in [76] that if an n-node graph has e edges and its maximal congestion is c then its crossing number is at least $\frac{n(n-1)(n-2)(n-3)}{20c^2} - \frac{e}{2}$.

3.2.2 Bisection Width

In this section we obtain bounds on the bisection width of $PG_r(N)$. We start by presenting upper bounds simply obtained from the value of the bisection width of $G(N)$.

Theorem 3.7 *If the bisection width of $G(N)$ is B and N is even, then the bisection width of $PG_r(N)$ is at most BN^{r-1}.*

Proof: The bisection of each $G(N)$-subgraph in a given dimension i bisects the whole graph $PG_r(N)$. Let $G(N)$ be divided into two subgraphs with vertex sets U and V, respectively, when bisected. Then, the bisection of each dimension-i $G(N)$-subgraph divides $PG_r(N)$ into two subgraphs with nodes of the form $x_r...x_{i+1}ux_{i-1}...x_1$ and $x_r...x_{i+1}vx_{i-1}$ $...x_1$, respectively, where $u \in U$, $v \in V$, and x_j, for $j = 1, ..., r$ and $j \neq i$, is a node of $G(N)$. Since $|U| = |V|$, both graphs have the same number of nodes and $PG_r(N)$ has been bisected. ■

The upper bound on the bisection width of a product graph $PG_r(N)$ when N is odd is a little more complicated. When we bisect $G(N)$ we obtain two subgraphs with a difference of one in the number of nodes. Therefore, we can not use the method presented above in this case.

Let us assume that the bisection of $G(N)$ yields subgraphs with node sets U and V, and that $|V| = |U| + 1$. We can define B' to be the minimum number of edges that need to be removed from $G(N)$ to bisect it into two subgraphs with vertex sets $V - \{v\}$ and $U \cup \{v\}$, for some $v \in V$. Clearly, if there are several ways to bisect $G(N)$ by removing B edges, B' can be taken as the minimum of the corresponding possible values.

With these assumptions we can show that.

Theorem 3.8 *If the bisection width of $G(N)$ is B, N is odd, and B' is as defined, then the bisection width of $PG_r(N)$ is at most $B + \frac{B}{2}\sum_{i=1}^{r-1}(N^i + 1) + \frac{B'}{2}\sum_{i=1}^{r-1}(N^i - 1)$.*

Proof: To bisect the product graph we apply an inductive process. $PG_r(N)$ is bisected by choosing a dimension (say r) and first bisecting the vth $PG_r^r(N)$ subgraph, where v is the node of $G(N)$ presented above. This partition divides the vth $PG_r^r(N)$ subgraph into two subgraphs disconnected from each other in each dimension and connected to the rest of $PG_r(N)$ by the rth dimension. One subgraph contains $(N^{r-1} + 1)/2$ nodes and the other contains $(N^{r-1} - 1)/2$ nodes.

We can now bisect each dimension-r $G(N)$-subgraph to finish the process. A $G(N)$-subgraph that contains a node from the large subgraph of vth $PG_r^r(N)$ subgraph is bisected by removing B edges, while the rest of the $G(N)$-subgraphs are bisected by removing B' edges. This bisects the graph $PG_r(N)$.

If we denote $f(r)$ the number of edges removed to bisect $PG_r(N)$, from the above process we obtain $f(r) = f(r-1) + B(N^{r-1}+1)/2 + B'(N^{r-1}-1)/2$. The initial condition

an upper bound on it, we derive lower bounds on the bisection width and the crossing number of the product graph.

These lower bounds will be applied in Chapter 6 to obtain lower bounds on the VLSI layout area and maximum wire length for product networks. We can observe that the bounds obtained from the bisection width and from the crossing number are asymptotically the same. These two parameters are the two known approaches to this problem, and no link between them was known. Therefore, the maximal congestion appears to be such a link.

3.2.1 Maximal Congestion

In this section we study the maximal congestion of product graphs. Since we do not really need the exact value of the maximal congestion, we will show how to obtain upper bounds on the maximal congestion of $PG_r(N)$ given the value of the maximal congestion of $G(N)$.

To our knowledge, this is the first time the maximal congestion is explicitly identified as an important structural property of a graph. In the following sections we will use the maximal congestion to obtain lower bounds on properties of homogeneous product graphs that, contrary to the above results, cannot be derived from the same property of the factor graph.

Theorem 3.6 *If the maximal congestion of $G(N)$ is C, then the maximal congestion of $PG_r(N)$ is at most CN^{r-1}.*

Proof: We show a mapping of the edges of the N^r-node directed complete graph into paths of $PG_r(N)$. We first map the nodes of the directed complete graph onto the nodes of $PG_r(N)$ one-to-one. Then, we map the directed edge from node $x = x_r...x_1$ to node $y = y_r...y_1$ to the path

$$x \rightarrow y_r x_{r-1}...x_1 \rightarrow ... \rightarrow y_r...y_2 x_1 \rightarrow y$$

The ith arrow represents the path in the corresponding $G(N)$-subgraph from x_i to y_i, for $i = 1, ..., r$. By definition of maximal congestion, these paths imply at most congestion C in the $G(N)$ subgraph.

Let $(z_r...z_i...z_1, z_r...z'_i...z_1)$ be a dimension-i edge of $PG_r(N)$. If this edge is traversed by a path from x to y as described, then it must be $y_r = z_r$, ..., $y_{i+1} = z_{i+1}$ and $x_{i-1} = z_{i-1}$, ..., $z_1 = z_1$. Since the edge (z_i, z'_i) of $G(N)$ is traversed by at most C paths between two nodes of $G(N)$, there are at most C possible combinations of the values of x_i and y_i. Each other x_j, for $j = i+1, ..., r$, and y_k, for $k = 1, ..., i-1$, can take N possible values. Therefore, the edge can be traversed by at most CN^{r-1} paths. ■

	$G(N)$	$PG_r(N)$
Nodes	N	N^r
Edges	E	ErN^{r-1}
Diameter	d	rd
Connectivity	κ	$r\kappa$
Min. vertex degree	δ	$r\delta$
Max. vertex degree	Δ	$r\Delta$
Partitionability	-	N^i, for $i = 0...r$
	k	k^r

Table 3.1: Structural properties of $PG_r(N)$ obtained from similar properties of $G(N)$.

3.1.5 Partitionability

The ability to recursively partition a graph into distinct copies of its smaller versions is another important property, since it allows assigning the parts of a recursive computation to different subnetworks, or shows a way to share the system between many users.

As we already mentioned, product graphs contain a variety of subgraphs which are isomorphic copies of product graphs with fewer dimensions. We have shown in Chapter 2 that by removing the edges belonging to k dimensions we obtain a set of disjoint copies of $PG_{r-k}(N)$. Hence the next theorem follows.

Theorem 3.4 *$PG_r(N)$ is N^i-partitionable, for $i = 0, ..., r$.*

Furthermore, if the factor graph $G(N)$ is already partitionable into k disjoint isomorphic M-node graphs of its same family, by applying this partition to each $G(N)$-subgraph in all the dimensions we obtain a partition of $PG_r(N)$ into k^r disjoint subgraphs isomorphic to $PG_r(M)$.

Theorem 3.5 *If $G(N)$ is k-partitionable, then $PG_r(N)$ is k^r-partitionable.*

The structural results presented until this point have been compiled in Table 3.1. All these results can be characterized by the fact that the properties of the product graph involved are easily obtained from the same properties of the factor graph. In the rest of the chapter we obtain non-trivial properties of product graphs.

3.2 Advanced Results

In this section we obtain properties that are not trivially derived from the value of the same property in the factor graph. We initially study the maximal congestion and, from

Theorem 3.1 *If $G(N)$ has diameter d, then $PG_r(N)$ has diameter rd.*

The key to the proof of Theorem 3.1 is the observation that there exist at least one pair of nodes in $PG_r(N)$ which differ in every symbol position and each differing symbol pair correspond to a distance as much as the diameter of $G(N)$. Finding such a pair of nodes yields both a lower bound and an upper bound for the diameter of the product graph.

3.1.3 Connectivity

The connectivity of a network has important implications on its communication bandwidth and its fault tolerance. The connectivity of product networks was investigated in [24] where a lower bound was obtained.

Theorem 3.2 *If $G(N)$ has connectivity κ, then $PG_r(N)$ has connectivity at least $r\kappa$.*

The theorem can be simply proven with an inductive statement, where it is shown that if $PG_{r-1}(N)$ has connectivity $(r-1)\kappa$, then $PG_r(N)$ has connectivity $(r-1)\kappa+\kappa$. The increased number of paths is due to the alternatives added by the introduced dimension.

3.1.4 Vertex Degree

The vertex degree is important mainly in two aspects of a network. First, the maximum vertex degree has strong implications on the cost of the implementation of the network. Second, the minimum vertex degree determines an upper bound on the connectivity of the network.

The maximum vertex degree of product networks has been previously studied [24, 82, 86], while we are not aware of any result on the minimum vertex degree. We join results on both properties in the following theorem.

Theorem 3.3 *If $G(N)$ has maximum vertex degree Δ and minimum vertex degree δ, then $PG_r(N)$ has maximum vertex degree $r\Delta$ and minimum vertex degree $r\delta$.*

Note from Figure 2.2 that each time we add a new dimension to the product network, we add at least δ and at most Δ to the vertex degrees. Then, there is a vertex $x = x_r...x_1$ where each x_i, for $i = 1, ..., r$, corresponds to a vertex of $G(N)$ with degree δ. This means that x is a node with the minimum vertex degree of $r\delta$. Similarly, there is a vertex $y = y_r...y_1$ where each y_i, for $i = 1, ..., r$, corresponds to a vertex of $G(N)$ with degree Δ. Then y must have the maximum vertex degree $r\Delta$. Trivially, by construction, no node can have vertex degree smaller than $r\delta$ or larger than $r\Delta$.

Chapter 3

Structural Properties

In this chapter we compile some known structural properties of product networks and we add to the collection with new non-trivial results on other properties not studied previously.

3.1 Direct Results

In this section we present general results on several structural properties of homogeneous product networks. The properties presented here are directly derived from the value of the same property in the factor graph.

3.1.1 Number of Nodes and Links

Among the first questions we ask about a new interconnection network is the number of nodes and links in it. We can easily observe that, if $G(N)$ has N vertices and E edges, then $PG_r(N)$ has N^r vertices and ErN^{r-1} edges. The statement about the number of vertices follows directly from Definition 2.2. To compute the number of edges, it is possible to observe in Figure 2.2 that $PG_r(N)$ contains all the edges of N copies of $PG_{r-1}(N)$ plus the edges of N^{r-1} copies of $G(N)$ (since there are N^{r-1} columns in the figure.) Thus, we can derive a recurrence $f(r) = Nf(r-1) + EN^{r-1}$ for the number of edges in $PG_r(N)$. The solution of this, with the initial condition $f(1) = E$, gives the desired result.

3.1.2 Diameter

The diameter of a network is another important property. In general, computation of exact diameter for a given graph may be very difficult, but for homogeneous product graphs we are able to state simple rules to calculate the diameter. The next result has been presented in several papers independently [4, 24, 82, 86].

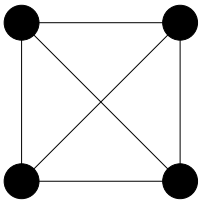

Figure 2.12: The 4-node complete graph, $K(4)$.

number of dimensions. We denote the Petersen graph as $P(10)$ and its r-dimensional product as $PP_r(10)$, which is obtained as $PP_r(10) = P(10) \otimes PP_{r-1}(10)$.

Finally, we define the complete graph as follows.

Definition 2.14 *The N-node* complete graph, *denoted $K(N)$, is the graph where each of its nodes is connected with an edge to all the other $N-1$ nodes.*

Figure 2.12 presents the 4-node complete graph $K(4)$. Observe that the vertex degree of $K(N)$ increases with N.

The r-dimensional product of $K(N)$ is denoted $PK_r(N)$, and obtained as $PK_r(N) = K(N) \otimes PK_{r-1}(N)$. Since the 2-node linear array, $L(2)$, is isomorphic to $K(2)$, the hypercube can be considered as the multidimensional product of $K(2)$.

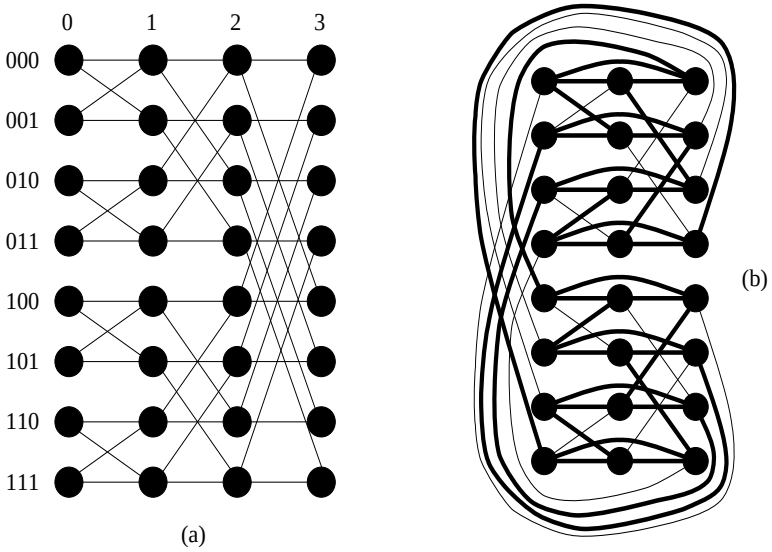

Figure 2.9: The 3-level butterfly and wrapped butterfly, $B(24)$.

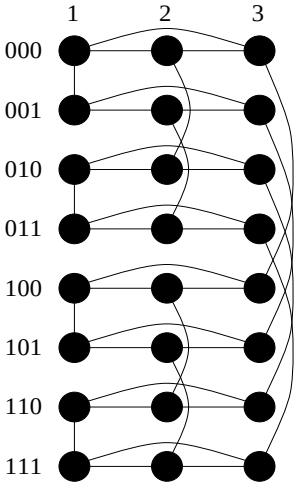

Figure 2.10: The 3-dimensional cube-connected cycles, $C(24)$.

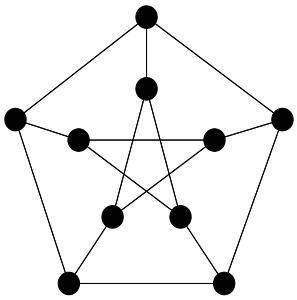

Figure 2.11: The Petersen graph, $P(10)$.

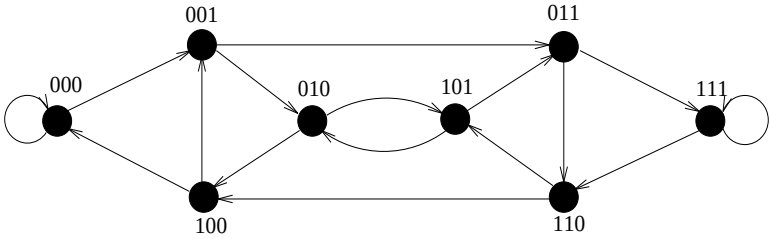

Figure 2.8: The 8-node de Bruijn graph, $D(8)$.

$(\langle u,i\rangle,\langle v,j\rangle)$ is an edge of the butterfly if and only if $j = i+1$ and either (a) $u = v$ (straight edge) or (b) u and v differ only in the ith bit (cross edge.)

If we collapse the nodes $\langle u,0\rangle$ and $\langle u,n\rangle$ into one single node, for $u = 0,...,2^n-1$, we obtain the *wrapped butterfly*. Figure 2.9.(a) shows the 3-level butterfly and Figure 2.9.(b) the 3-level wrapped butterfly.

As a rule, in the rest of the dissertation we only study the properties of the $N = n2^n$-node wrapped butterfly, which we denote as $B(N)$. This decision is justifiable by the fact of both networks being almost the same. For instance, it is known that both butterflies can emulate each other with constant slowdown. Furthermore, observe that both have asymptotically the same number of nodes $\Theta(n2^n)$, since the n-level butterfly has $(n+1)2^n$ nodes and the n-level wrapped butterfly has $n2^n$ nodes. Most of the results obtained for the butterfly are expressed in asymptotical notation and, hence, are applicable to both networks.

The r-dimensional product of (wrapped) butterflies is denoted as $PB_r(N)$, and can be obtained as $PB_r(N) = B(N) \otimes PB_{r-1}(N)$.

Definition 2.13 *The n-dimensional* cube-connected cycles *is the graph obtained from the hypercube Q_n by replacing each node of the hypercube with a n-node cycle, so that each node of the cycle is connected to one of the edges incident to the original node.*

The n-dimensional cube-connected cycles has $N = n2^n$ nodes and will be denoted $C(N)$. Figure 2.10 presents the 3-dimensional cube-connected cycles, $C(24)$. It is easy to see that the cube-connected cycles and the butterfly are very similar. In fact, the n-dimensional cube-connected cycles is a subgraph of the n-level wrapped butterfly (in Figure 2.9.(b) the darker edges represent the edges of the 3-dimensional cube-connected cycles.) Similarly, there is a constant-congestion constant-dilation embedding of the wrapped butterfly onto the cube-connected cycles.

The r-dimensional product of $C(N)$ is denoted $PC_r(N)$ and obtained as $PC_r(N) = C(N) \otimes PC_{r-1}(N)$.

The next network to be considered is the *Petersen graph,* shown in Figure 2.11. It is a fixed-size graph and, therefore, its product version can only grow by changing the

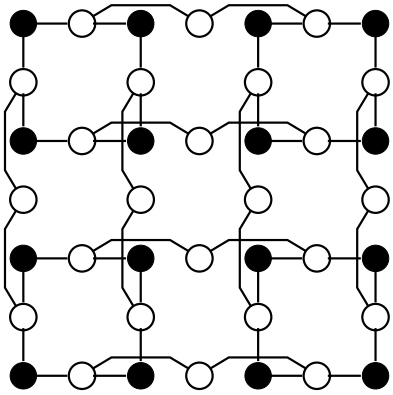

Figure 2.6: The 2-dimensional mesh of 4-leaf trees.

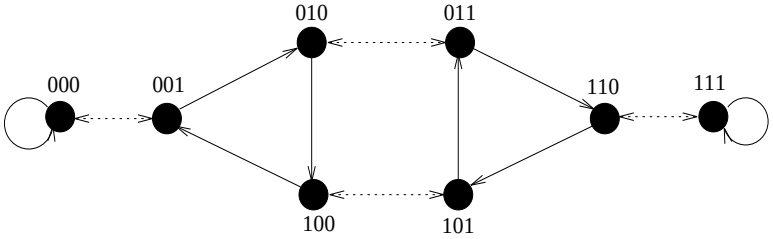

Figure 2.7: The 8-node shuffle-exchange graph, $S(8)$.

The r-dimensional product of N-node shuffle-exchange graphs will be denoted $PS_r(N)$, and it is obtained as $PS_r(N) = S(N) \otimes PS_{r-1}(N)$.

Definition 2.11 *The N-node* de Bruijn graph, *denoted $D(N)$, where $N = 2^h$, is the graph with vertex set $0, ..., N-1$ (in binary), and whose nodes u, v, and w are connected by edges (u, v) and (u, w) if v can be obtained from u by a cyclic left shift and w differs from v in the rightmost bit only.*

Note that, although for simplicity of definition we use directed edges to describe the graphs $S(N)$ and $D(N)$, once the construction in done the resulting graph is considered undirected.

An 8-node de Bruijn graph is shown in Figure 2.8. Observe that, whenever (u, v) is a shuffle-edge in $S(N)$, it is also an edge in $D(N)$. Additionally, whenever (u, v, w) is a path in $S(N)$ such that (u, v) is a shuffle edge and (v, w) is an exchange edge, (u, w) is an edge in $D(N)$.

The r-dimensional product of N-node de Bruijn graphs will be denoted $PD_r(N)$, and it is obtained as $PD_r(N) = D(N) \otimes PD_{r-1}(N)$.

Definition 2.12 *The n-level* butterfly *is the graph with vertex set $\langle u, i \rangle$, where i is the level of the node, $0 \leq i \leq n$, and u is the row of the node, $0 \leq u \leq 2^n - 1$ in binary.*

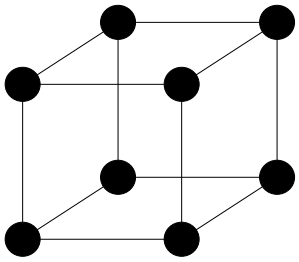

Figure 2.4: The 3-dimensional hypercube, Q_3.

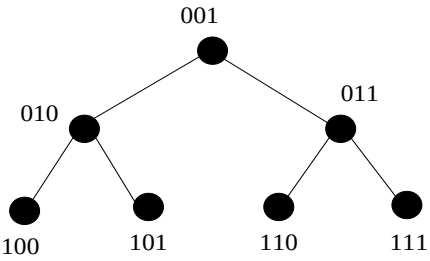

Figure 2.5: The 7-node complete binary tree, $T(7)$.

1, is the root of the tree. Figure 2.5 presents the 7-node complete binary tree, which has 3 levels and 4 leaves.

The r-dimensional product of $T(N)$, using the defined notation, will be denoted as $PT_r(N)$. It can be obtained as $PT_r(N) = T(N) \otimes PT_{r-1}(N)$. In Figure 1.1 we have presented the 2-dimensional product of 7-node complete binary trees, $PT_2(7)$.

Definition 2.9 *The r-dimensional N^r-leaf* mesh of trees, *or r-dimensional mesh of N-leaf trees, is the graph obtained from the N^r-node r-dimensional grid by substituting the linear connections along each dimension by N-leaf complete binary trees. The leaves of the trees are the original nodes of the grid, and additional nodes are introduced to obtain the internal nodes of the trees.*

Figure 2.6 presents the 2-dimensional 16-leaf mesh of trees (or 2-dimensional mesh of 4-leaf trees.) In this figure the nodes of the original grid are shown as dark nodes and the additional nodes introduced are shown as empty nodes.

Definition 2.10 *The N-node* shuffle-exchange graph, *denoted $S(N)$, where $N = 2^h$, is the graph with vertex set $0, ..., N-1$ (in binary), and whose nodes u and v are connected by an edge (u, v) if either (a) u and v differ in the rightmost bit only (denoted as a "exchange" edge) or (b) v can be obtained from u by a cyclic left shift (denoted as a "shuffle" edge.)*

An 8-node shuffle-exchange graph is shown in Figure 2.7. In this figure shuffle edges are shown as solid lines and exchange edges are shown as dotted lines.

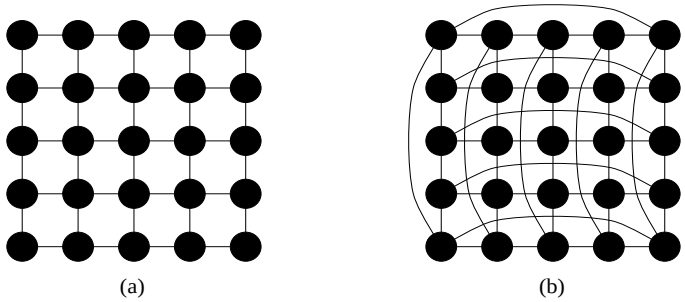

Figure 2.3: The 25-node 2-dimensional grid, $PL_2(5)$, and the 25-node 2-dimensional torus, $PR_2(5)$, respectively.

2.4 Networks of Interest

We start by defining several networks which will be often referred to in the dissertation and non-product networks whose r-dimensional products will be studied

Definition 2.7 *The N^r-node r-dimensional* grid *(resp.* torus*) is the graph whose vertices comprise all the r-tuples $x = x_r...x_1$, such that $x_i \in \{0, ..., N-1\}$, for $i = 1, ..., r$, and whose edges connect any pair of nodes x and y if and only if x and y differ in exactly one index position i and $x_i = y_i + 1$ (resp. $x_i = (y_i + 1)$ mod N.)*

As can be observed, the N^r-node r-dimensional torus is the r-dimensional product of the N-node ring and the N^r-node r-dimensional grid is the r-dimensional product of the N-node linear array. Clearly, the N^r-node r-dimensional grid is a subgraph of the N^r-node r-dimensional torus.

We will denote the N-node linear array as $L(N)$, and the N-node ring (or cycle) as $R(N)$. Then, by using the product notation, the graph $PL_r(N)$ (resp. $PR_r(N)$) is the N^r-node r-dimensional grid (resp. torus.) From Definition 2.2 they can be obtained as $PL_r(N) = L(N) \otimes PL_{r-1}(N)$ and $PR_r(N) = R(N) \otimes PR_{r-1}(N)$, respectively.

The r-dimensional *hypercube* is simply the 2^r-node r-dimensional grid, $PL_r(2)$. For the sake of brevity, we often denote the r dimensional hypercube as Q_r, whose factor graph is Q_1.

Figure 2.3 presents the 25-node 2-dimensional grid, $PL_2(5)$, and the 25-node 2-dimensional torus, $PR_2(5)$. Figure 2.4 presents the 3-dimensional hypercube (or 8-node 3-dimensional grid), Q_3.

Definition 2.8 *The N-node* complete binary tree, *denoted $T(N)$, where $N = 2^h - 1$, is the graph whose vertices comprise the set $\{1, ..., N\}$ and whose edges connect each vertex $u < 2^{h-1}$ with the vertices $2u$ and $2u + 1$.*

$T(N)$ has h levels of nodes, where the ith level contains the nodes 2^{i-1} to $2^i - 1$. The nodes at level h are called the "leaves" of the tree, and the single node at level 1, labeled

In order to obtain lower bounds on the value of these structural properties, we introduce a new structural property that is of great interest for several results in our work.

Definition 2.5 *The* maximal congestion *of a N-node graph, denoted C, is the congestion for any embedding of the N-node directed complete graph onto it.*

The definitions of embedding and congestion of an embedding are given in the next section. The maximal congestion is an intrinsic parameter of a graph just like the chromatic number, crossing number, etc. are intrinsic parameters of a graph. Although it looks somehow strange and difficult to obtain, for all the studied networks it has been enough to have a tight upper bound of its value. Such a bound can be simply obtained for an arbitrary network by applying a routing algorithm between each pair of nodes and counting the congestion of each edge of the network.

2.3 Embedding Properties

We start by giving a formal definition of embedding:

Definition 2.6 *An* embedding *of a "guest" graph $G(N_G)$ into a "host" graph $H(N_H)$ is a mapping f of the vertices of G into the vertices of H and a mapping g of the edges of G into paths in H, such that if (u, v) is an edge of G, then $g((u, v))$ is a path connecting $f(u)$ and $f(v)$ in H.*

The main cost measures used in embedding efficiency are:

- The *load* of the embedding is the maximum number of vertices of the guest graph mapped to any vertex of the host graph.
- The *dilation* of an embedding is the maximum path length in the host graph representing an edge of the guest graph.
- The *congestion* of an embedding is the maximum number of paths (that correspond to the edges of the guest graph) that share any edge of the host graph.
- The *expansion* of an embedding is the ratio N_H/N_G of the host and guest graphs sizes.

It has been shown in [37] that if G can be embedded into H with load l, dilation d, and congestion c, H can emulate t steps of a computation running on G in $O(l + d + c)t$ steps. If the values l, d, and c are constant, the slowdown introduced by this emulation is also constant. We consider an embedding efficient if the cost measures are bounded, *i.e.* they are $O(1)$.

This notation can be extended, since by removing the edges of k different dimensions in $PG_r(N)$ we obtain N^k disjoint copies of $PG_{r-k}(N)$. If we remove the edges in dimensions $i_1, ..., i_k$, we denote the $PG_{r-k}(N)$ subgraph containing the node $x = x_r...x_1$, as the $(x_{i_1}, ..., x_{i_k})$th $PG_r^{i_1,...,i_k}(N)$ subgraph of $PG_r(N)$.

2.2 Structural Properties

In this section we define several structural properties and establish their notation for the rest of the document.

Let the *distance* between two nodes in a network be the minimum number of edges that need to be traversed to go from one node to the other, then the *diameter* of a network is the maximum distance between any pair of nodes of the network, and it will be denoted as d. The diameter of a network is an upper bound on the time that any node-to-node communication in the network will take.

The *connectivity* of a network is the minimum number of vertex-disjoint paths connecting any two nodes of the network, and will be denoted as κ. This value is the same as the minimum number of nodes to be removed from the network to disconnect it. The connectivity of a network is related with the fault-tolerance in the sense that, if the connectivity of the network is κ, then the network can tolerate up to $\kappa - 1$ faults in nodes and edges.

The *vertex degree* of a node is the number of edges incident to it. We are specially interested on the maximum vertex degree of a network, *i.e.* the maximum of the vertex degrees of its nodes, denoted as Δ. This value determines its maximum connectivity and has implications in its VLSI layout. We will also study the minimum vertex degree, denoted as δ. The vertex degree of an arbitrary node u will be denoted as δ_u

A graph is said to be *k-partitionable* if it contains as subgraphs k disjoint isomorphic copies of a graph of its same family. The partitionability properties of a network are very closely related to the scalability of the network, or the ease of increasing the size of a network to another network of the same family. The partitionability of a network is very useful when implementing recursive algorithms, when working with different size problems, or when sharing the network between several users.

The above properties for a product graph are trivially obtained from the same properties of the factor graph. However, other structural properties are more difficult to be derived and have never been studied before. We concentrate on two such structural properties of product networks: the bisection width and the crossing number.

Definition 2.3 *The* bisection width *of a graph, denoted B, is the minimum number of edges that have to be removed from it to obtain two disjoint subgraphs with the same number of nodes (within one.)*

Definition 2.4 *The* crossing number *of a graph, denoted c, is the minimum number of edge crossings of any drawing of the graph in the plane.*

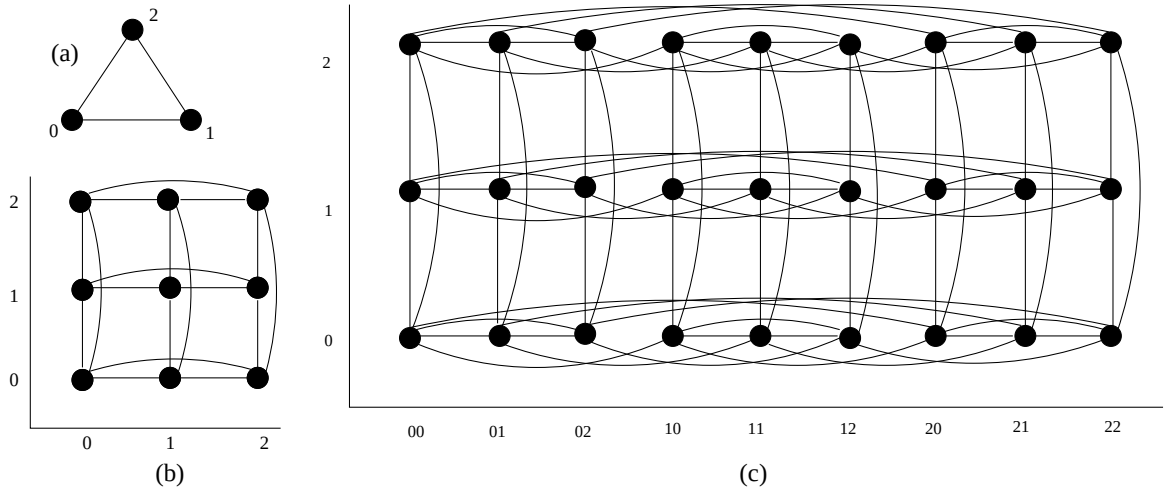

Figure 2.2: Recursive construction of multi-dimensional product networks.

From the intuitive description of the construction of $G \otimes H$ presented above, the construction of $PG_r(N)$ from $PG_{r-1}(N)$ and $G(N)$ is intuitively described in several simple steps: First, place the vertices of $PG_{r-1}(N)$ along a straight line. Draw N copies of this drawing of $PG_{r-1}(N)$ at the same vertical level in parallel columns. Associate a different vertex u of $G(N)$ to each copy of $PG_{r-1}(N)$ and extend the vertex labels of each node x to ux. Finally, connect two nodes ux and vx in the same column if and only if (u, v) is an edge in $G(N)$. Figure 2.2 shows how to obtain the 3-dimensional homogeneous product of the 3-node ring in two applications of this process.

From the definition, the edges of $PG_r(N)$ can be characterized as follows.

Observation 2.2 *If x and y are in the form $x = x_r...x_1$ and $y = y_r...y_1$, where x_i, y_i are nodes of $G(N)$, for $i = 1, ..., r$, then (x, y) is an edge in $PG_r(N)$ if and only if x and y differ in exactly one symbol position i, and the differing symbols (x_i, y_i) define an edge in $G(N)$.*

We say that an edge (x, y) belongs to dimension i if the nodes incident to it differ only in the ith index position. A $G(N)$-subgraph of $PG_r(N)$ is said to be a dimension-i subgraph if any two nodes in the subgraph differ only in the ith index position.

Observe that $PG_r(N)$ contains N disjoint copies of $PG_{r-1}(N)$, each with a different node of $G(N)$ associated. These copies can be obtained by removing the dimension-r edges from $PG_r(N)$. Similarly, from Observation 2.1 a similar set of disjoint subgraphs can be obtained by removing all the edges of any dimension i from $PG_r(N)$. We use the notation $PG_r^i(N)$ to refer to any of the disjoint subgraphs obtained by removing the dimension-i edges of $PG_r(N)$. Since each of the resulting subgraphs has a different node u of $G(N)$ associated, we can meaningfully talk about the uth $PG_r^i(N)$ subgraph of $PG_r(N)$. For every node x of the uth $PG_r^i(N)$ subgraph of $PG_r(N)$ we have that $x_i = u$.

At a more intuitive level, the construction of $G \otimes H$ from G and H can be described as follows. First, place the vertices of H along a straight line as shown in Figure 2.1. Then, draw $|U|$ copies of H such that the vertices with identical labels fall in the same column. Next, extend the vertex labels by associating a different label $u \in U$ to each copy of H and changing each vertex label $v \in V$ of the copy to uv. Finally, connect the columns in the interconnection pattern of the labeled graph G, such that uv is connected to $u'v$ if and only if (u, u') is an edge in G.

From the symmetry in this definition, note that the product operator is commutative and associative:

Observation 2.1 *$G_1 \otimes G_2$ is isomorphic to $G_2 \otimes G_1$, and $G_1 \otimes (G_2 \otimes G_3) = (G_1 \otimes G_2) \otimes G_3$.*

Observe that $G_1 \otimes G_2$ and $G_2 \otimes G_1$ are not the same graph since, although in both of them each vertex is a pair of symbols, the order in the vertices of one is reverse than in the vertices of the other. However, if G_1 and G_2 are the same graph, then both product graphs are the same.

Note that, by construction, $G \otimes H$ contains $|U|$ disjoint copies of H as subgraphs, and each copy has a different label $u \in U$ associated. Similarly, from the above observation, $G \otimes H$ has $|V|$ disjoint copies of G, each with a different label $v \in V$ associated.

We say that a graph is a *product graph* if it can be obtained from a set of factor graphs by the application of the cartesian product operation. If all the factor graphs are isomorphic we have a *homogeneous product graph.* Otherwise, the product graph is *heterogeneous.*

It will often be important to indicate the number of vertices, so we use $G(N)$ to denote the N-node graph G. The r-dimensional product of $G(N)$ is denoted $PG_r(N)$, with the subscript r representing the number of dimensions.

Applying this notation, the formal definition of r-dimensional homogeneous product graph is given as follows:

Definition 2.2 *Given a graph $G(N)$, its r-dimensional homogeneous product, denoted $PG_r(N)$, is*

1. *a single vertex without any edges and no labels when $r = 0$,*
2. *$PG_r(N) = G(N) \otimes PG_{r-1}(N)$, when $r > 0$.*

In general, we let x, y, z denote the vertices of homogeneous product graphs obtained from $G(N)$. For the r-dimensional product graph $PG_r(N)$, the vertex labels x, y, z are tuples of r symbols where each symbol is drawn from the set of vertices of $G(N)$. We use u, v, w to denote single vertices of $G(N)$. When the vertex of $G(N)$ is part of the label of a node x of $PG_r(N)$, it is denoted as x_i, where the subindex i indicates its position in the label. For example, x is in the form $x = x_r ... x_i ... x_1$, where x_i, for $i = 1, ..., r$, is a vertex of $G(N)$.

Chapter 2

Definitions and Notation

In this chapter we present definitions and notation that will be used in the rest of the dissertation. In order to keep this chapter brief and to locate specific information faster, those definitions that are relevant to only one chapter have been placed in the specific chapter.

2.1 Homogeneous Product Networks

As a reminder to the reader, we start this section with the definition of the cartesian product, which is illustrated in Figure 2.1.

Definition 2.1 *The* cartesian product *of two "factor" graphs* $G = (U, E)$ *and* $H = (V, F)$ *is the graph* $G \otimes H$ *whose vertex set is* $U \times V$ *and whose edge set contains all the edges* $(uv, u'v')$ *such that* $\{u, u'\} \subseteq U$, $\{v, v'\} \subseteq V$, *and either* $u = u'$ *and* $(v, v') \in F$, *or* $v = v'$ *and* $(u, u') \in E$.

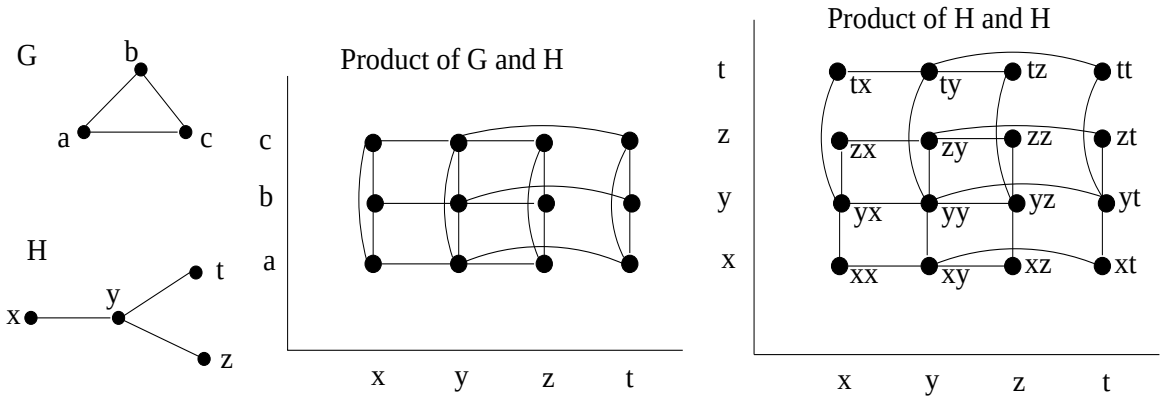

Figure 2.1: Definition of cartesian product.

network by combining collinear layouts for the factor graph.

After applying all these results to several networks we are able to obtain optimal-area layouts for all of them, with maximum wire lengths close to optimal.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 contains formal definitions and notation that will be used in the rest of the document. Those definitions and notation that are relevant to only one of the chapters have been placed in that specific chapter for locality reasons.

The following chapters present the results of our program of research. Chapter 3 presents the obtained results in the study of structural properties of product networks. Chapter 4 presents the results in embedding a product graph into another. Chapter 5 presents general algorithms for product networks. Chapter 6 presents the results in VLSI complexity obtained.

In Chapter 7 we concentrate on three specific instances of product networks and present further results for them.

Finally, in Chapter 8 we summarize conclusions of the work described here.

yields the same time complexity as the implementation in the mesh of trees, which is the fastest known.

1.2.5 VLSI Complexity

The VLSI layout model used in this research has been defined by Thompson [77, 78]. In this model the layout area is divided into square unit-area tiles where the nodes and wires are placed.

Thompson showed that the square of the bisection width is a lower bound on the area required by a network under this model. Similarly, Leighton [41] presented the crossing number of a graph as a lower bound on the area. We use these facts to derive lower bounds on VLSI complexity for product networks from the value of the bisection width and the crossing number. They are derived by using the maximal congestion, a new structural property of the factor network we present.

To obtain upper bounds we first use two traditional frameworks: separators and bifurcators.

Separators were initially used by Lipton and Tarjan [46, 47] to study planar graphs. Subsequently, they have been found to be useful to derive area-efficient layouts for arbitrary graphs by Floyd and Ullman [27], Leiserson [43, 44], and Valiant [81]. Furthermore, Bhatt and Leiserson [11] showed that they can be used to obtain layouts with short wires. All these results have been compiled by Leighton [41] and Ullman [80].

Bifurcators appeared as an alternative to separators, since they solve some of the restrictions presented by the separator framework. The initial papers defining and using bifurcators are due to Leighton [40] and Bhatt and Leiserson [10, 11], and these results have been compiled and improved by Bhatt and Leighton [9].

In this dissertation we show how to derive separators and bifurcators for the product graph from separators and bifurcators of the factor graph. Then, the results of the above references allow to obtain the desired upper bounds.

However, both separators and bifurcators are restricted by definition to networks with bounded degree. Sherlekar and JáJá [70, 72, 73] tried to solve this problem by defining stronger kinds of separators and bifurcators. However, they are so restrictive that we could not use them for our networks.

The last approach we investigate is based on the existence of efficient collinear layouts (layouts with all the nodes in one line) of the factor graph. Leiserson [44] showed how to obtain collinear layouts for a network with a given separator. Similarly, in [9] an upper bound is presented on the area of collinear layouts of bounded-degree networks. We also refer to graph-theory papers where labeling of graphs are studied. The problem of obtaining efficient collinear layouts is equivalent to the problem of finding a labeling of the graph with small bandwidth and small cutwidth [15, 16].

Besides these methods, we show how to obtain efficient collinear layouts for the factor graph from a regular layout. Then, we present how to obtain a layout for the product

matrices, and finding the minimum-weight spanning tree of a graph. These algorithms are easily modified to sove other problems with similar structure. For instance, it is trivial to modify the minimum-weight spanning-tree algorithm for determining the connected components, transitive closure, and shortest paths of graphs.

Our sorting algorithm is based on the odd-even merge sorting algorithm due to Batcher [2]. In this reference, Batcher presented two efficient sorting networks. Algorithms derived from these networks have been presented for a number of different parallel architectures, like the shuffle-exchange [75], the grid [50, 79], the cube-connected cycles [59], and the mesh of trees [51].

One of Batcher's sorting networks has, as main components, subnetworks that sort bitonic sequences. Sorting algorithms based on this method are generally called "bitonic sorters." A bitonic sequence is the concatenation of a non-decreasing sequence of keys with a non-increasing sequence of keys, or the rotation of such a sequence. Several papers have been devoted to generalizing bitonic sorters, generalizing Batcher's network [3, 48, 49]. Recently, Lee and Batcher [38] presented a new network to merge and sort k bitonic sequences.

The main components of the other sorting network proposed by Batcher are subnetworks that merge two sorted sequences into a single sorted sequence, denoted odd-even merge networks. To our knowledge, no result on generalizing the odd-even merge network to merge more than two sorted sequences exists.

In this dissertation we first develop a sorting algorithm for homogeneous product networks based on merging N sorted sequences into a single sorted sequence (we denote this operation multiway merge.) When we apply the sorting algorithm to specific networks we obtain optimal time complexity for some of them, and same complexity as Batcher's algorithm in the rest.

We continue by developing an algorithm to compute the summation of several values initially in the nodes of the network. The algorithm is based on the existence of an algorithm to compute summations in the factor network. If this basic algorithm is optimal, the derived algorithm is also optimal. The algorithm is easily adapted to similar problems. For instance, in the minimum-weight spanning-tree algorithm mentioned later we use an algorithm to obtain the minimum of a set of values that is structurally similar to the summation algorithm.

We develop a matrix-multiplication algorithm based on the algorithm presented by Preparata and Vuillemin [58] for the mesh of trees and its implementation by emulation in the hypercube. In this algorithm we use the broadcasting algorithm from [86] and the summation algorithm presented before. We implement the algorithm in several networks and for all of them it yields optimal time complexity.

The last algorithm we present computes the minimum-weight spanning tree of a graph. As we mentioned, this algorithm can be simply modified to solve similar problems. The algorithm is derived from algorithms developed by Hirschberg, Chandra, and Sarwate [33] and Shiloach and Vishkin [74]. When implemented in several networks, the algorithm

of the network to the other in one step. As Thompson [78] pointed out, the exact value of the bisection width of a graph is, in general, very difficult to obtain.

The crossing number was originally identified as an important parameter of an interconnection network by Leighton [41]. He showed that it defines a lower bound on the VLSI layout area of the network. Like the bisection width, it is not easy to obtain the exact value of this parameter, in general. For instance, there are not known exact values even for well-known networks like the hypercube or the cube-connected cycles [76].

In this dissertation we define a new structural property of graphs, which we call the *maximal congestion*, and we use the value of this property for the factor graphs to derive lower bounds on the value of the bisection width and the crossing number. Also upper bounds on the bisection width are derived from the bisection width of the factor graph. The application of these results to several networks shows that the obtained bounds are tight.

1.2.3 Embedding Properties

In the literature, it has been customary to formalize the notion of emulation with the notion of embedding [7, 61]. This comes from the property shown by Koch *et al* [37] that an efficient emulation between networks can be obtained from an efficient embedding. Therefore, to show that a network can efficiently emulate another network it is enough to show that the later can be efficiently embedded into the former. In fact, several researchers compared the computational power of interconnection networks by embedding-based emulations [1, 5, 6, 7, 29].

Simple results involving some cost measures of embeddings between product graphs have been presented in [42] and [86]. We strengthen these results and obtain new ones, which allow to derive embedding properties of the product networks from those of their factor graphs.

In general, the problem of finding efficient embedding is not simple. For instance, several papers have been published just on embedding the complete binary tree into the hypercube [8, 13, 14, 22, 23, 31, 45]. The inherent difficulty of embedding problems aside, it is very interesting that we can obtain efficient embeddings for the product graphs based on embeddings for the simpler factor graphs. In this context, any previous embedding result from the literature can be useful. For instance, we use the embedding proposed by Heckman *et al.* [32] to obtain an optimal-dilation embedding of the product of complete binary trees onto the grid.

1.2.4 Algorithms

General algorithms have been presented for broadcasting, point-to-point communication, off-line permutation, and fault-tolerant routing in product networks [4, 24, 56, 86]. We increase this collection with algorithms for sorting, computing summations, multiplying

detail these contributions in the following sections.

Examples of recently proposed heterogeneous (*i.e.* not homogeneous) product networks are the hyper-de Bruijn network proposed by Ganesan and Pradhan [28], the hyper-Petersen network proposed by Das and Banerjee [20], the banyan-hypercube network proposed by Youssef and Narahari [84], and the folded Petersen cube proposed by Öhring and Das [53]. All of them combine the hypercube with another factor network. They show that the resulting network has some advantages over the hypercube. Along these lines, Youssef [85, 86] combined the hypercube with several other networks and analyzed some of their properties.

Also, a few homogeneous product networks have been recently proposed. Rosenberg [61] introduced the two-dimensional product of de Bruijn graphs (which he calls the product-shuffle network) as a parallel architecture and has analyzed several of its computational properties[1]. He showed that this network contains rings, grids, complete binary trees, and meshes of trees as subgraphs. It can also emulate butterflies, shuffle-exchange, and de Bruijn graphs with constant slowdown. Panwar and Patnaik [57] proposed the two-dimensional product of shuffle-exchange graphs (which they call the modified shuffle-exchange network) as an alternative to the pure shuffle-exchange graph in solving linear systems. Öhring and Das [54] proposed the folded Petersen network, which is the multidimensional product of the Petersen graph.

Most of the general results obtained in this dissertation are applied to the above networks and to other homogeneous product networks never previously studied. We specially concentrate in the study of multidimensional products of complete binary trees, shuffle-exchange graphs, and de Bruijn graphs, and show that they are powerful interconnection networks.

1.2.2 Structural Properties

Several of the above mentioned references have explored some structural properties of product networks [4, 24, 82, 86]. These properties are trivially obtained from properties of the factor graphs. The properties covered by these references are the diameter, the connectivity, the vertex degree, and the partitionability of product networks.

However, other structural properties are more difficult to be derived and have never been studied before. In this dissertation we concentrate on two such structural properties of product networks: the bisection width and the crossing number.

The importance of the bisection width of a networks was pointed out by Thompson [77, 78], who showed that it implies an upper bound on the speed of certain computations in the network and a lower bound on the layout area of the network. In general, the bisection width gives an idea of how much information can be transferred from one side

[1]Rosenberg does not restrict the factor de Bruijn graphs to have same number of nodes. Hence, these networks are not strictly homogeneous.

1.2 Related Work

In this section we present work conducted by other researchers which, in one form or another, is relevant to our research. We also briefly present our contribution in each specific area.

Since we attempt to perform a comprehensive study of the different aspects of homogeneous product networks, references in many different areas have been collected here. We organize them in five main sections, following a structure similar to the document itself, in order to simplify the cross reference. The first subsection presents references about the cartesian product operation and about product networks as parallel architectures. In the rest of the subsections we refer to publications that influenced us in obtaining properties of homogeneous product networks in specific aspects of the study: structural properties, embedding properties, algorithms, and VLSI complexity.

1.2.1 Product Networks

The cartesian product operation is a very well-known operation in graph theory. The cartesian product combines two "factor" graphs into a "product" graph. Harary [30] has cited the works of Shapiro [69] and Sabidusi [63, 64] as early studies of the graph-theoretic properties of the cartesian product and product graphs. For instance, Sabidusi [64] showed that any graph has a decomposition into a unique set of "prime" factor graphs. Several other authors also studied the product operation from a graph-theory viewpoint [17, 52, 82].

Referring to interconnection networks, many widely-used networks are instances of product graphs, obtained by the multiple applications of the cartesian product operation. Examples of these are the grid, the torus, the hypercube, and the generalized hypercube [12]. This fact has been already used, for instance, to obtain and prove several properties of the hypercube [42, 62, 87].

The product operation has been seen as a unifying framework to the study of specific properties of interconnection networks by several authors. Youssef [86] compiled results on the structural, routing, and embedding properties of product networks. Baumslag and Annexstein [4] developed generalized off-line permutation routing algorithms for product networks and used them to create a general strategy for finding efficient permutation routes in arbitrary networks. El-Ghazawi and Youssef [24] studied the connectivity of product networks with respect to the connectivity of the factor network, obtained a lower bound on the connectivity of a product network, and developed fault-tolerant point-to-point routing algorithms. Öhring and Hohndel [56] presented fault-tolerant routing algorithms for broadcasting, gossiping, scattering, and total exchange, by finding spanning trees of the product graph. For a set of specific product networks, Öhring and Das [55] presented dynamic embeddings for dynamically-evolving trees and grids.

In this dissertation we compile some of these results and develop many others. We

research in defining new interconnection networks exists for cost reasons. The ideal interconnection pattern is the complete connection of each processor with each other. However, the cost of this interconnection scheme is only affordable when the number of processors is small. In general, research in designing new interconnection networks seeks to reach a compromise between cost and power.

We mainly evaluate the implementation cost of a network from its vertex degree and its VLSI layout complexity. The two VLSI layout parameters considered are the area and the length of the longest wire, since they determine the cost of the layout and the maximum speed of the system, respectively.

1.1 Applications

The proposed approach is specially suited to the study and implementation of special-purpose parallel architectures. The framework presented allows to evaluate and meaningfully compare different candidate architectures to solve a concrete problem, and choose the one that best fits the requirements and restrictions.

The most popular application of these special purpose architectures are embedded systems. These are subsystems of a larger system which are in charge of performing special tasks within the whole system. Examples are the Viterby decoder, based on the de Bruijn graph, developed by the Caltech's Jet Propulsion Laboratory to be used in the Galileo mission to Jupiter [18], or the network for template matching that is being developed in the University of South Florida [60].

On the other hand, emulation is one of the key operations in parallel architectures. In this context, a parallel system may consist of a very large number of very simple processors and specific interconnection schemes may be implemented by emulation. Therefore, the physical interconnection network must have powerful and flexible emulation capabilities. Most of the product networks that we present in this dissertation have this property and they could be interesting candidates for the task.

However, the results obtained in this research are not restricted to special-purpose SIMD architectures. Most of the properties derived are only dependent on the graph-theoretical model of the interconnection network, and not on the computational model of the system (*e.g.* SIMD, MIMD, etc.) Therefore, the obtained results on structural properties, embedding capabilities, and VLSI complexity are valid even if we are trying to design a general purpose architecture. Furthermore, although the presented algorithms assume a SIMD model of computation, they can still run on a general purpose parallel machine if it is appropriately programmed. Therefore, the results presented here are also practically applicable to existing general purpose parallel systems, such as the Intel Paragon and the Maspar whose interconnection networks are based on the grid, and the NCube and the iPSC/860 whose interconnection networks are based on hypercube, since these networks are instances of product networks.

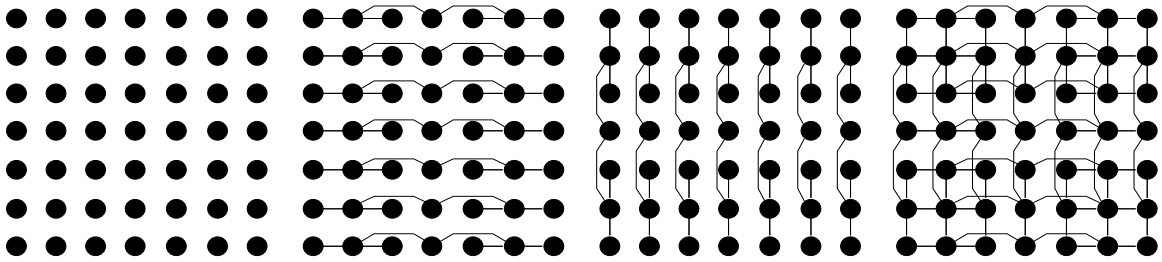

Figure 1.1: Construction of the 2-dimensional product of complete binary trees.

most of the cases. Among the instances evaluated there are several homogeneous product networks never previously proposed as interconnection networks. The evaluation of the properties of these networks presents them as very powerful and interesting candidates for future use as interconnection networks.

Surprisingly, to our knowledge, this is the first comprehensive study of the product notation as a unifying framework for the evaluation of interconnection networks. No previous reference has presented a collection of general results about product networks like the ones we show here. The results obtained allow to fully evaluate a new product network and compare its capabilities with other networks.

In this dissertation we study interconnection networks as "problem solvers." We study their powers as special-purpose architectures, implemented to solve specific computational problems. The whole network cooperates to efficiently perform this task under the SIMD model. This focus relieves us from studying properties that are only meaningful in the MIMD model of computation, like throughput, bandwidth, hot spots, etc.

With this in mind, the two main aspects to be evaluated in an interconnection network are its power and its implementation cost. The power of a network itself comes from its several properties. First, topological characteristics, like the diameter, bisection width, or connectivity, say much about the potential of the network as a parallel architecture.

Second, given our view of networks, the power of a network is mainly shown by developing fast-running algorithms for the network. Although, in general, it is not possible to use the running time of the algorithms to establish that a network is more powerful than another, since the structure of one or the other might be specially suited for particular problems, in some cases it gives a clear idea of the potential of the network.

Third, the power of a network can be also shown from its emulation capabilities. Efficient emulations transfer all the power of a network into another, giving a way to perform all the algorithms developed for the former in the later. A network H is considered to be at least as powerful as another network G if H can emulate any computation of G with constant slowdown. It is usual to formalize the notion of emulation with the notion of embedding, and assume H to be at least as powerful as G if there is an efficient embedding of G into H.

Besides the power, the cost is another important factor in a network. In fact, all

Chapter 1

Introduction

The *interconnection network* is one of the most important elements in a distributed-memory parallel architecture. This is because the interconnection scheme strongly determines the capabilities of a parallel architecture. For this reason, designing efficient interconnection networks has been at the forefront of parallel computing research.

In this dissertation we propose the *cartesian product operation* (or *product*, for short) as a unifying framework to study interconnection topologies. Several popular interconnection networks fall in the class of product networks (*e.g.* hypercube, grid, torus) and many others can be generated. The proposed framework will allow us to evaluate their properties and make meaningful comparisons between them.

Simply, we obtain the r-dimensional product of the N-node graph G from the r-dimensional $N \times ... \times N$ grid by replacing the linear connections of the grid by the interconnection pattern of G. For example, Figure 1.1 shows the construction of the 2-dimensional product of 7-node complete binary trees. In general, different factor graphs can be used in different dimensions and, hence, several different topologies can be generated. However, in this research we will concentrate on product networks with the same interconnection scheme for each dimension, which we denote *homogeneous product networks*, for which we can state stronger results.

The main results of this dissertation are expressed as rules that derive some property of a homogeneous product network from properties of its factor network. The framework allows a clean and simple notation to express and prove statements of the form "if G has the property A, then the r-dimensional product of G has the corresponding property $f(A)$." The statements themselves are independent of the specific graph G and, therefore, fully general. Specifically, the rules derived allow us to obtain structural properties, embedding capabilities, and the VLSI layout complexity of homogeneous product networks. We also develop general algorithms for several important problems, which are efficient for any homogeneous product network.

The application of these results to specific instances of product networks yields either exact values of the studied parameters or bounds on them, that are shown to be tight in

List of Figures

List of Tables

Contents

Acknowledgements

First, I would like to express my immense gratitude to my dissertation director, Dr. Kemal Efe, for his constant guidance and support. Most of the ideas included in this dissertation are the result of our intense discussions. Furthermore, he has always been a friend, and that friendship, I am sure, will go far beyond the end of this dissertation.

I would also like to thank Dr. Subrata Dasgupta, Dr. Henry Chu, and Dr. Nian-Feng Tzeng for being members of my committee and helping me to improve the quality of this dissertation with their comments.

Finally, I would like to recognize the role of my family in the realization of this work. Even from far away, they have been always there to help me, solve any problem that could arise, and give me their support.

I dedicate this dissertation to my mother and the memory of my father.

Homogeneous Product Networks for Processor Interconnection

Antonio Fernández

APPROVED:

Kemal Efe, Chairman Associate Professor of Computer Science	Subrata Dasgupta Professor of Computer Science
Chee-Hung Henry Chu Associate Professor of Computer Engineering	Nian-Feng Tzeng Associate Professor of Computer Engineering

Joan T. Cain
Dean, Graduate School

Homogeneous Product Networks for Processor Interconnection

A Dissertation
Presented to
The Graduate Faculty of
The University of Southwestern Louisiana
In Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

Antonio Fernández
Fall 1994