REGULAR PROCESSOR ARRAYS

Allen D. Malony

Center for Supercomputing Research and Development
University of Illinois at Urbana–Champaign.

ABSTRACT

Regular is an often used term to suggest simple and uniform structure of a parallel processor's organisation or a parallel algorithm's operation. However, a strict definition is long overdue. In this paper, we define regularity for processor array structures in two dimensions and enumerate the eleven distinct regular topologies. Space and time emulation schemes among the regular processor arrays are constructed to compare their geometric and performance characteristics. The hexagonal array is shown to have the most efficient emulation capabilities.

Keywords: regularity, processor arrays, emulation, interconnection networks

INTRODUCTION

The most widely debated topic in parallel processing research is how to interconnect multiple processors. The arguments take place across many different cost/performance criteria such as algorithm mapping, scalability, reconfigurability, communication efficiency, graph embedding, fault tolerance, and VLSI implementation.

Mesh connected processor arrays were among the first processor interconnection structures proposed for parallel processing [1] [7]. Their distinguishing feature is the connection of processors only to immediate neighbors where the connection degree is uniform throughout the array. The original motivation for mesh topologies came from their ability to easily represent the natural data flow patterns found in many algorithms [8] [7] [13] [18] [17] [27].

The thought of interconnecting thousands of processors brought on a wave of new processor interconnection structures aimed at providing cost–effective solutions to certain key scalability issues such as mean internode distance, communication traffic density, connections per node, link visit ratios, and fault tolerance [21] [28]. The processor arrays proposed included the torus, X–tree, chordal ring, R–ary N–cube, cube–connected cycles, spanning bus hypercube, and dual bus hypercube, in addition to the standard bus, crossbar, ring, and tree architectures [28]. Although favored for their regular geometry, uniform communication and simple extension, the mesh connected processor arrays were generally less desired because of the fact that internode communication delays increase as the square root of the number of nodes in the system.

Systolic array research approached the problem of designing processor arrays by concentrating on requirements for an effective VLSI implementation of a parallel algorithm [11] [9] [10]. Chip area, time and power required to implement an algorithm in VLSI are dominated by the communication geometry of the algorithm [25]. The effects of the area and time parameters of VLSI can be reduced to a large degree if very simple and regular patterns of interconnections between elements are used [16] [26]. The regularity requirement imposed on interconnection structures, in a broad sense, deals with the layout of the communication geometry in a two–dimensional area [22]. Simple and regular interconnection geometries that are two–dimensional and plane filling lead to cheap implementations and high chip density. Also, parallel algorithms with simple and regular communication and data flows are more appropriate for VLSI implementation and will result in higher performance.

The choice of processor array design to achieve good generalised communication performance conflicted with the simple processor arrays favored for specialized VLSI systems. If only the more sophisticated communication topologies were implemented in VLSI, then their communication efficiencies could be combined with the faster VLSI speeds. However, several recent results suggest that mesh–connected arrays have comparable, if not better, general communication efficiency and performance when implemented in VLSI as compared to other networks [15] [19]. In addition, there has been much work done on making regular mesh arrays more flexible [3] [23] [4] [2] [20].

In this paper, we consider the question of what are the simple and regular processor array topologies? The primary contribution of this work is the enumeration and analysis of the "regular" two–dimensional processor array topologies using a geometric definition of regularity. Several topologies are shown that have not appeared in the computer science literature previously. Our analysis of the regular processor arrays is based on their ability to emulate the other members of the class. We consider both space emulation (processors of the host array are combined into "logical" nodes of the target array) as well as time emulation (the interconnection geometry of the target array is provided by time–multiplexing the links).

REGULARITY

Intuitively, the term *regular* implies simplicity and uniformity in space. A more quantitative geometrical definition of regularity can be formulated from the extensive mathematical literature on graphs [5] [6] [12] [24]. Although regularity can be defined for multiple dimensions, our discussion is restricted to graphs that are two–dimensional, i.e. planar. A second requirement is that the graph have a simple description and be uniformly extensible following a basic set of construction rules. By the graph being uniformly extensible, we mean that the properties of the vertices and edges do not change as the number of nodes is increased; e.g., the length of an edge. Another requirement for regular graphs is that the vertices have equal degree. The final requirement is that regular graphs be plane filling. That is, the infinite graph completely covers the two–dimensional plane.

The requirements placed on regular graphs are not without mathematical precedence. Justification comes from the old geometrical problem of determining those convex polygon figures that tessellate the plane [5] [6]. In particular, the problem is to construct tilings of the plane where a single convex polygon of r sides is used. Based on Euler's theorem $v - e + f = 1$ (v vertices, e edges and f faces of a polygonal network of tiles) and basic Diophantine analysis, it is a simple consequence that $3 \leq r \leq 6$ [5].

Although there are eighty–one types of isohedral tilings in the plane [5], there are ONLY eleven topologically distinct types of *Laves*

This work was supported in part by the National Science Foundation under Grants No. US NSF DCR84–10110, the U. S. Department of Energy under Grant No. US DOE–DE–FG02–85ER25001, the U.S. Air Force Office of Scientific Research Grant No. AFOSR–F49620–86–C–0136, and the IBM Donation.

nets [12] (also called *regular* or *Subnikov nets* [24]) which are the "skeleton" graphs consisting of tile "vertices" (where three or more tiles meet), and tile "edges" where two tiles intersect. Figure 1 shows the eleven Laves nets along with symbols denoting the valences of the vertices as the tessellating r–gon is traced; e.g., $3^2.4.3.4$ describes a pentagon tessellation where the pentagon meets 3 other tiles, then 3, 4, 3, and finally, 4 other tiles. The geometry of the distinct tessellation topologies can be described from this simple vertex valency syntax.

Tessellation structures embody the requirements set forth for regular graphs: they are two–dimensional, they have a simple description (tile vertex valency syntax), all tiles used in a tessellation have the same number of edges (r–gon), they are uniformly extensible, and they are plane filling. If we associate a tile to a processor array node and the links to tile intersections (tile edges), the resultant interconnection topology will embody the same regular properties.

The regular processor interconnection graphs can be generated by taking the *dual* of the Laves nets, i.e. the faces (tiles) are mapped to vertices, the tile vertices are mapped to faces, and tile edges map to edges between the new vertices [5]. Because the graphical duality mapping is isomorphic, there are exactly eleven distinct regular processor array topologies. These topologies are also known as the familiar *nearest neighbor* topologies because all vertices are of equal degree and each vertex connects to that many of its nearest neighbors.

Definition: A graph is regular if it is two dimensional, all vertices have equal degree and the dual of the graph is a tessellation.

Definition: A processor array is regular if its interconnection topology is a regular graph.

In the next section, we consider emulations among the regular processor arrays. In particular, we focus on the triangular (6^3), the orthogonal (4^4) and the hexagonal (3^6) topologies. These have been defined to be *strongly regular* because they form a set closed under duality: the triangular graph is the dual of the hexagonal and vice versa, and the orthogonal graph is the dual of itself [14].

REGULAR PROCESSOR ARRAY EMULATION

Although the number of regular processor arrays is finite, it would be cost inefficient to include each array in a parallel processing system and use an array only when there is an appropriate match between an algorithm's communication geometry and that array's topology. Instead, we would like to design the system with a single processor array that offers good performance across a wide range of algorithms. The versatility of a processor array is measured not only by the range of algorithms for which it is specifically suited but also by the ease to which other algorithms can be mapped to its communi cation geometry [2], and the ability of the array to reconfigure it communication geometry to that of the algorithms or other array topologies [3] [4] [23]. We evaluate the regular processor arrays based on their ability to emulate other regular arrays.

Emulation Philosophy

The goal of emulating a *target* regular array by a *host* regular array is to reproduce the communication properties of the target array in the host. The emulation can take place either in space or in time. *Space emulation* structurally maps the host array to the target array by physically grouping host nodes into logical target nodes and activating the appropriate host links such that the communication topology of the target array is realized. If the target array cannot be embedded in the host array with a one to one node mapping, the space emulation will necessarily result in a reduction of the effective size of the emulated target array.

Time emulation realizes the communication properties of the target array by time multiplexing the host array links. Once a one to one node mapping is made between the target and the host, the maximum number of host "minor" communication time cycles needed to realise the communication connectivity of one "major" target time cycle can be determined. If the target array cannot be embedded in the host array with a one to one link mapping (we already assume a one to one node mapping), the time emulation will necessarily result in an increase in the number of time cycles needed to execute an algorithm on the emulated target array.

Space Emulation

An optimal space emulation scheme should minimise the average number of host nodes used to emulate a node in the target array.

Definition: The *space emulation efficiency* $S_M(N)$ of a space emulation scheme used by regular array M to emulate regular array N is the average number of nodes of M required to emulate one node of N. If N contains n nodes, the size of the emulated target array will be n / $S_M(N)$ nodes. A lower bound on $S_M(N)$ can be determined by calculating the number of host array nodes needed to match the node degree of the target array. An *optimal* space emulation scheme achieves the lower bound of the average number of host nodes required for a node of the emulated target array. That is, no more than the number of host nodes needed to meet the node degree requirements of the emulated target array are used.

The process followed to construct an emulation scheme begins by grouping adjacent nodes together to form logical nodes of the emulated target array. "Active" host links are then selected to realise the target communication geometry. During operation, nodes within a group coordinate their actions to correctly communicate data across the active links. Instead of enumerating all space emulations for all regular host arrays ad nauseam, we instead concentrate on the strongly regular arrays.

Theorem 1: The triangular array can optimally emulate the hexagonal array with a space emulation efficiency of four and the orthogonal array with a space emulation efficiency of two.
Proof: The emulation schemes are shown in Figure 2.

Theorem 2: The orthogonal array can optimally emulate the hexagonal array with a space emulation efficiency of two and the triangular array with a space emulation efficiency of one.
Proof: The emulation schemes are in [14].

Theorem 3: The hexagonal array can optimally emulate the orthogonal array with a space emulation efficiency of one and the triangular array with a space emulation efficiency of one.
Proof: The emulation schemes are shown in Figure 3.

Theorem 4: $S_{triangular}(R) \leq S_{orthogonal}(R) \leq S_{hexagonal}(R)$ where R is a regular array.
Proof: Any space emulation scheme used by the triangular array to emulate another regular array can also be used by the orthogonal and hexagonal arrays since only one node is required by the orthogonal and hexagonal arrays to emulate a node in the triangular array. Therefore, any emulation scheme used by the orthogonal and hexagonal arrays for emulating another array must be at least as efficient as the optimal scheme that would be used by the triangular array. A similar argument is applied to show $S_{orthogonal}(R) \leq S_{hexagonal}(R)$.

Space Emulation Schemes

The space emulation schemes for the regular processors arrays using the strongly regular arrays are shown in Figure 2 for the triangular host array and Figures 3 for the hexagonal host array [14]. Node groupings for the triangular host array are shown shaded. The dashed lines in the hexagonal host array indicate inactive links.

Space Emulation Efficiency

The space emulation efficiencies of the schemes presented for the strongly regular arrays are shown in Tables 1. As expected, the hex-

agonal array shows the best efficiencies with nine of the schemes using an optimal emulation of one. The inability to achieve optimal schemes for $3^4.6$ and 3.6.3.6 is attributed to the rigid structure of those topologies.

Notice that the triangular array was able to achieve more optimal space emulations than the orthogonal array. In part, this has to do with the orthogonal array's inability to realise triangular interconnection paths present in some of the arrays such as $3^4.6$, $3^2.4.3.4$ and 3.6.3.6.

An interesting observation from the table is that $S_{triangular}(3.4.6.4) = 3$, yet $S_{triangular}(4^4) = 2$ and $S_{orthogonal}(3.4.6.4) = 1\frac{1}{3}$. One quickly realises that a better space emulation scheme could be achieve for 3.4.6.4 using the triangular array if the orthogonal array was first emulated and then its emulation scheme applied to realise 3.4.6.4. This would result in an emulation efficiency of $2\frac{2}{3}$ instead of 3.

Space emulation analysis allows a simple measure of cost, $S_M(N)$, to be used for comparing the versatility of the different regular arrays. Additionally, the pay back for adding additional links to the array can be easily discerned. Finally, algorithms be designed for one particular regular array can executed on another array with bounded performance degradation.

Time Emulation

Time emulations among the regular processor arrays are more complex to construct because a mapping from nodes of the host array to nodes of the target array must first be devised. We employed some convenient shortcuts that allowed us to develop a collection of time emulations for the strongly regular arrays as target topologies.

Definition: The *time emulation efficiency* $T_M(N)$ of a time emulation scheme used by regular array M to emulate regular array N is the number of communication time cycles required in M to realise the data transfer between nodes possible in one cycle in N. Assuming the processor array speeds are equal, if N completes an algorithm in t time cycles, the time emulation scheme used by M will finish in $T_M(N) * t$ time cycles.

Notice that if $S_M(N)=1$, $T_M(N)=1$. We can make use of this fact to compute bounds on time emulations based time emulations already known. That is, if $T_M(N)=t_1$ and $T_N(O)=t_2$, then $T_M(O)\leq t_1 * t_2$.

Initial time emulations can be constructed by looking at the space emulations with efficiency one. Since these already give a one to one node mapping, an optimal time emulation of the host array (in the space emulation) by the target array (in the space emulation) can be devised and its efficiency calculated by visually following the shortest path to establish the single link connections of the underlying host array. For instance, we compute $T_{3.4.6.4}(3^6)$ to be three by looking at the space emulation scheme of 3.4.6.4 by 3^6 and following the shortest path between connected hexagonal nodes using only the 3.4.6.4 links.

Following the above procedure, we were able to construct optimal time emulations of the hexagonal array for most regular arrays. The time emulation efficiencies are shown in Table 2. The parentheses indicate upper bounds determined by applying the above formulas to these optimal hexagonal and orthogonal time emulations. Other entries in the table come from visually mapping one processor array onto another as in the case of the square array onto $3^2.4.3.4$ and $3^3.4^2$.

The interest in time emulations comes from the fact that the emulated target array is not reduced in size. Instead, a more complex routing of data in multiple time steps is required to emulate the target array's communication properties. However, we cannot ignore the time needed to route data in a space emulation scheme. In fact, we see that there are cases where a time emulation will be actually faster than a space emulation; a time emulation of a hexagonal array using a triangular array will take three time cycles whereas the space emulation requires four. In other cases, the opposite is true; consider the triangular array emulating the orthogonal array.

CONCLUSION

Processor interconnection topologies incorporating communication and spatial regularity will become increasingly important as VLSI dimensions continue to decrease. Although mesh processor arrays have known scalability limitations with respect to communication [28], several recent reports suggest that the communication efficiency of two dimensional meshes is better than other interconnection topologies when compared for VLSI implementation [15] [19].

The regular processor arrays described in this paper are geometrically defined based on nearest neighbor connections and space–filing properties. Interestingy, only eleven processor arrays of regular topology exist in two dimensions. We have enumerated these arrays as well as presented space and time emulation schemes. A natural extension of the work presented here concerns regular three dimension organizations. Research in this area will become more important and necessary as VLSI begins to offer three dimensional interconnects.

REFERENCES

[1] G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnik, R.A. Stokes. *The ILLIAC IV Computer.* **IEEE Trans. Comp.**, Vol. c–17, 1968, pp. 746–757.

[2] S.H. Bokhari. *On the Mapping Problem.* **IEEE Trans. Comput.**, Vol. C–30, March 1981, pp. 207–214.

[3] Naga S. Gollakota and F.Gail Gray. *Reconfigurable Cellular Architecture.* **Proc. 11th Int. Symp. Computer Arch.**, 1984, pp. 377–379.

[4] D. Gordon, I. Koren and G. Silberman. *Embedding Tree Structures in VLSI Hexagonal Arrays.* **IEEE Trans. Comput.**, Vol. C–33, Jan. 1984, pp. 104–107.

[5] B. Grunbaum and G.C. Shephard. *The Eighty-one Types of Isohedral Tilings in the Plane.* **Math. Proc. of the Cambridge Phil. Soc.**, Vol. 82, Sept. 1977, pp. 177–196.

[6] B. Grunbaum and G.C. Shephard. *Tilings with Congruent Tiles.* **Bulletin of the Amer. Math. Soc.**, Vol. 3, No. 3, Nov. 1980, pp. 951–973.

[7] W.H. Kautz, K.N. Levitt and A. Waksman. *Cellular Interconnection Arrays.* **IEEE Trans. Comp.**, Vol. C–17, No. 5, May 1968, pp. 443–451.

[8] D.J. Kuck. *ILLIAC IV Software and Applications Programming.* **IEEE Trans. Comp.**, Vol. C–17, No. 8, Aug. 1968, pp. 758–770.

[9] H.T. Kung. *Let's Design Algorithms for VLSI Systems.* **Caltech Conf. on VLSI**, Jan. 1979, pp. 65–90.

[10] H.T. Kung. *Why Systolic Architectures?.* **Computer**, Vol. 15, No. 1, Jan. 1982, pp. 97–107.

[11] H.T. Kung and C.E. Leiserson. *Algorithms for VLSI Processor Arrays.* in **Introduction to VLSI Systems**, by C. Mead and L. Conway, Addison–Wesley, 1980, pp. 271–292.

[12] F. Laves. *Ebenenteilung und Koordinationszahl.* **Z. Kristallogr.**, Vol. 78, 1931, pp. 208–241.

[13] K.N. Levitt and W.H. Kautz. *Cellular Arrays for the Solution of Graph Problems.* **CACM**, Vol.15, No. 9, 1972, pp. 789–801.

[14] Allen D. Malony. *Regular Interconnection Networks.* Master's Thesis, Univ. of California, Los Angeles, August 1982.

[15] Pinaki Mazumder. *Evaluation of Three Interconnection Networks for CMOS VLSI Implementation.* **1986 ICPP**, Aug. 1986, pp. 200–207.

[16] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison–Wesley, Reading, Mass., 1980.

[17] D. Nassimi and S. Sahni. *Bitonic Sort on a Mesh-Connected Parallel Computer.* **IEEE Trans. Comput.**, Vol. C–27, Jan. 1979, pp. 2–7.

[18] S.E. Orcutt. *Implementation of Permutation Functions in ILLIAC IV-Type Computers.* **IEEE Trans. Comp.**, Vol. C-25, No. 9, Sept. 1976, pp. 929–936.
[19] A.G. Ranade and S.L. Johnson. *The Communication Efficiency of Meshes, Boolean Cubes and Cube Connected Cycles for Wafer Scale Integration.* **Proc. 1987 Inter. Conf. on Parallel Proc.**, Aug. 1987, pp. 479–482.
[20] D.A. Reed, L.M. Adams and M.L. Patrick. *Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems.* to appear in **IEEE Trans. Comput.**.
[21] D.A. Reed and H.D. Schwetman. *Cost-Performance Bounds for Multi-microcomputer Networks.* **IEEE Trans. Comput.**, Vol. C-32, No. 1, Jan. 1983, pp. 83–95.
[22] J.E. Savage. *Planar Circuit Complexity and the Performance of VLSI Algorithms.* **Proc. of the CMU Conf. on VLSI Systems and Computations**, 1981, pp. 61–67.
[23] L. Snyder. *Introduction to the Configurable, Highly Parallel Computer.* **Computer**, Vol 15, No. 1, Jan. 1982, pp. 47–56.
[24] A. Subnikov. *K voprosu o stroenii Kristallov.* **Bulletin Acad. Imp. Sci.**, Ser. 6, Vol. 10, 1916, pp. 755–779.
[25] I.E. Sutherland and C.A. Mead. *Microelectronics and Computer Science.* **Scientific American**, Vol. 237, Sept. 1977, pp. 210–228.
[26] C.D. Thompson. *Area-Time Complexity for VLSI.* **Proc. Caltech Conf on VLSI**, Jan. 1979, pp. 405–508.
[27] C.D. Thompson and H.T. Kung. *Sorting on a Mesh-Connected Parallel Computer.* **CACM**, Vol. 20, April 1977, pp. 263–271.
[28] L.D. Wittie. *Communications Structures for Large Networks of Microcomputers.* **IEEE Trans. Comput.**, Vol. C-30, No. 4, April 1981, pp. 264–273.

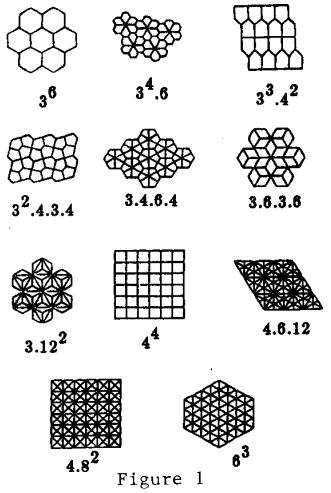

Figure 1

Target Topology	Optimal Emulation (host/target)			Emulation Efficiency (host/target)		
	Tri.	Ortho.	Hex.	Tri.	Ortho.	Hex.
3^6	4	2	1	4	2	1
$3^4.6$	3	2	1	3	2 1/3	1 1/6
$3^3.4^2$	3	2	1	3	2	1
$3^2.4.3.4$	3	2	1	3	2 1/4	1
3.4.6.4	2	1	1	3	1 1/3	1
3.6.3.6	2	1	1	2	1 1/3	1 1/3
3.12^2	1	1	1	2	2	1
4^4	2	1	1	2	1	1
4.6.12	1	1	1	2 2/3	1 1/3	1
4.8^2	1	1	1	2	1	1
6^3	1	1	1	1	1	1

Table 1. Space Emulation Efficiency

Host Topology	Emulation Efficiency in Cycles		
	Hexagonal	Orthogonal	Triangular
3^6	1	1	1
$3^3.4^2$	2	1	1
$3^2.4.3.4$	2	1	1
3.4.6.4	3	(3)	(3)
3.12^2	6	(6)	(6)
4^4	2	1	1
4.6.12	5	(5)	(5)
4.8^2	4	3	(3)
6^3	3	3	(3)

Table 2. Time Emulation Efficiency

Figure 2
TRIANGULAR SPACE EMULATION

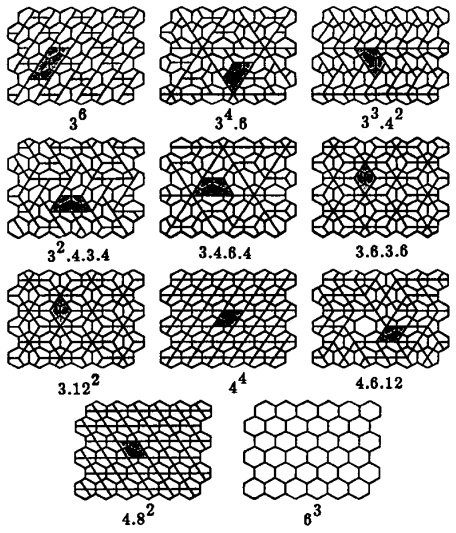

Figure 3
HEXAGONAL SPACE EMULATION