CSRD Rpt. No. 734

Submitted to **1988 International Conference on Parallel Processing**, St. Charles, IL, August 1988.

REGULAR PROCESSOR ARRAYS

Allen D. Malony

January 1988

Center for Supercomputing Research and Development
University of Illinois
305 Talbot – 104 South Wright Street
Urbana, IL 61801–2932
Phone: (217) 333–6223

This work was supported in part by the National Science Foundation under Grant No. US NSF MIP–8410110, the U. S. Department of Energy under Grant No. US DOE–DE–FG02–85ER25001, the U. S. Air Force Office of Scientific Research under Grant No. AFOSR–F49620–86–C–0136, and the IBM Donation.

1. Introduction

Perhaps 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. Numerous processor interconnection topologies have been devised each with advantages and disadvantages.

Mesh connected processor arrays were among the first processor interconnection structures proposed for parallel processing [BBKK68] [KaLW68] [vonN68] [YaAm69]. 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. Parallel algorithms for numerical problems [Kuck68] and graph problems [LeKa72] as well as algorithms for permuting [Orcu76], sorting [ThKu77] [NaSa79], and switching [KaLW68] were developed, primarily for orthogonal connected processor arrays.

However, 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 [Witt81] [ReSc83]. 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

Center for
Supercomputing Research and Development

REGULAR PROCESSOR ARRAYS

Allen D. Malony[1]

January 18, 1988

University of Illinois at Urbana–Champaign
104 S. Wright Street
Urbana, Illinois 61801

[1] Center for Supercomputing Research and Development, University of Illinois at Urbana–Champaign.

REGULAR PROCESSOR ARRAYS

Allen D. Malony[1]

January 18, 1988

Abstract

Regular is an often used term to suggest simple and uniform structure of a parallel processor's organization 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. We also show how algorithms developed for one regular processor array might be transferred to another regular array using matrix multiplication and LU decomposition as examples.

---

[1]Center for Supercomputing Research and Development, University of Illinois at Urbana–Champaign.

This work supported in part by the National Science Foundation under Grant No. US NSF MIP–8410110, the U.S. Department of Energy under Grant No. US DOE– DE–FG02–85ER25001, the U.S. Air Force Office of Scientific Research under Grant No. AFOSR–F49620–86–C–0136, and the IBM Donation.

the standard bus, crossbar, ring, and tree architectures [Witt81]. 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. Besides, other processor interconnection structures also claimed various degrees of regularity as well as flexibility, such as containing the mesh topologies as special cases.

Systolic array research approached the problem of designing processor arrays by concentrating on requirements for an effective VLSI implementation of a parallel algorithm [KuLe80] [Kung79] [Kung82]. Chip area, time and power required to implement an algorithm in VLSI are dominated by the communication geometry of the algorithm [SuMe77]. 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 [MeCo80] [Thom79]. The regularity requirement imposed on interconnection structures, in a broad sense, deals with the layout of the communication geometry in a two–dimensional area [Sava81]. 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 generalized 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 [Mazu86] [RaJo87]. In addition, there has been much work done on making regular mesh arrays more flexible through reconfigurability [ChFi82] [GoGr84] [Snyd82] [Venk85], graph embedding [GoKS84], and algorithm mapping [BeSn84] [Bokh81] [KoSi83] [MeSi87] [MoFo86] [ReAP87].

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). We also show how algorithms developed for one regular processor array can be transferred to another regular array using matrix multiplication and LU decomposition as examples.

2. Regularity

Intuitively, the term *regular* implies simplicity and uniformity in space. A more quantitative geometrical description is required, however, if we are to form a meaningful classification of processor array topologies. Luckily, there is a wealth of mathematical

literature that comes to bear to help us construct a definition of regularity. [Borr68] [Crit69] [Dumn71] [Gard72] [GrSh77] [GrSh80] [Kers68] [Lave31] [Loeb76] [Subn16] [Zalg69].

In this section, we describe processor interconnection geometries as *graphs* with the standard association being that vertices correspond to nodes and edges represent links in the processor array. Although regularity can be defined for multiple dimensions [Borr68][Crit69][Loeb76][Zalg69], 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. Although Malony considered regular graphs with vertices of possibly different degrees [Malo82], we restrict the definition here to apply only to graphs with vertices of equal degree. 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. The final requirement that we place on regular graphs is that they be plane filling. That is, the infinite graph completely covers the entire two–dimensional plane.

The requirements we place on regular graphs are not without mathematical precedence. The justification comes from the old geometrical problem of determining those convex polygon figures that tessellate the plane [Gard72][GrSh77][GrSh80][Kers68]. 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 $3 \leq r \leq 6$ [GrSh77].

Although there are eighty–one types of isohedral tilings in the plane [GrSh77], there are ONLY eleven topologically distinct types of *Laves nets* [Lave31] (also called *regular* or *Subnikov nets* [Subn16]) 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 entire geometry of the distinct tessellation topologies can be described from this simple vertex valency syntax.

So, what does this have to do with regular processor interconnection geometries? The point is that the tessellation structures embody the requirements that we 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, therefore, 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 [GrSh77]. These regular graph are shown in Figure 2 (the same notation is used except the numbers refer to the edge valency of the faces incident on a vertex). 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.

We are now ready for the definition of a regular graph and a regular processor array.

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.

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 [Malo82]. The strongly regular processor arrays are well known in the parallel processing literature because of their simple structure and geometrical symmetry which is important for regular algorithm data flows and for physical VLSI layout [Kung79] [FoKu80].

3. 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 communication geometry [BeSn84] [Bokh81] [KoSi83] [MeSi87] [MoFo86], and the ability of the array to reconfigure its communication geometry to that of the algorithms or other array topologies [AmEp74] [ChFi82] [GoGr84] [GoKS84] [Snyd82] [Venk85] [YaAm71]. In this section, we evaluate the regular processor arrays based on their ability to emulate other regular arrays.

3.1. 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 realize 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.

3.2. Space Emulation

One immediate conclusion quickly drawn regarding space emulation among regular arrays is that the hexagonal array is the most efficient due to its higher node degree. However, it is not as clear whether the triangular array is the least efficient just because its node degree is three, although we suspect this is true. Nor can one safely say that optimal hexagonal array space emulations can be achieved in all cases. Also, we would like to know the relative differences in space emulation performance between different host arrays.

We begin by defining a performance measure for space emulation. Clearly, an optimal space emulation scheme should minimize 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.

Definition: A *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 realize 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. First, we show that a topological hierarchy is formed among the three strongly regular networks with respect to space emulation.

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 3.1 and Figure 3.8.

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 shown in Figure 4.1 and Figure 4.11.

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 5.8 and Figure 5.11.

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 Figures 3.1–11 for the triangular host array, Figures 4.1–11 for the orthogonal host array, and Figures 5.1–11 for the hexagonal host array [Malo82]. For each scheme, the underlying host array is shown as nodes connected by dashed links. If a host link is solid, the link is active. Node groupings for the triangular host array are shown as contained within solid line boundaries; one of the groupings has been shaded in each emulation scheme. For the orthogonal array, node groupings are shown using rectangles. We have also left the internal connections between nodes within a grouping as dashed to help emphasize the grouping structure.

In some cases, it is seen that a host node has none of its links drawn in, i.e. solid; see Figure 3.9, Figure 4.4 and Figure 5.2. This means that this host node does not participate in the space emulation.

Space Emulation Efficiency

The space emulation efficiencies of the schemes presented for the strongly regular arrays are shown in Tables 1, 2 and 3. As expected, the hexagonal 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 realize 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 realizes 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 used to realize 3.4.6.4. This would result in an emulation efficiency of $2\frac{2}{3}$ instead of 3.

Observations

Space emulations among the regular arrays are interesting for several reasons. First, it provides a simple measure of cost, $S_M(N)$, for comparing the versatility of the different regular arrays. Second, it helps us determine the pay back for adding additional links to the array. Finally, it allows for algorithms to be designed for one particular regular array with the assurance that it can execute on another with bounded performance degradation.

3.3. 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. However, along with some simple eye–balling, we employed some convenient shortcuts that allowed us to develop a collection of time emulations for the strongly regular arrays as target topologies.

After the definition of time emulation efficiency, we make some simple observations.

> **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 realize 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 $T_M(O)\leq t_1 * t_2$.

We begin constructing time emulations by looking at the space emulations with efficiency one. Since these already give us a one to one node mapping, we can easily devise an optimal time emulation of the host array in the space emulation by the target array in the space emulation and calculate its efficiency 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 of Table 4. Optimal time emulations of the orthogonal array were also constructed for 4.8^2 and 6^3. The time emulation efficiencies in 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 $3^3.4^2$.

Observations

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.

1. Mapping Algorithms Among Regular Processor Arrays

Emulating a target regular processor array by some other regular host array either in space or time is one way of making an algorithm originally designed for the target array execute on the host. As we have seen, however, there will probably be some penalty paid in array size reduction or increased algorith execution time. Alternatively, one can try making minor changes to the data flow of an algorithm without affecting its overall structure so that the algorithm will execute on a different processor array [Malo82].

1.1. Matrix Multiplication on a Triangular Array

Kung and Leiserson developed a matrix multiplication algorithm for execution on a hexagonal processor array [KuLe80]. Because data flows in only three directions in the algorithm, we suspect that a triangular processor array might be able to perform the same steps of the computation.

The problem is one of multiplying two $n \times n$ matrices. The matrix product $C = [c_{ij}]$ of $A = [a_{ij}]$ and $B = [b_{ij}]$ can be computed by the recurrences:

$$c_{ij}^{(1)} = 0$$

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik} * b_{kj}$$

$$c_{ij} = c_{ij}^{(n+1)}$$

We consider A and B to be $n \times n$ band matrices of width $w1$ and $w2$, respectively. The matrix multiplication algorithm can be evaluated by the above recurrences by pipelining

the a_{ij}, b_{ij} and c_{ij} through an array of $2*w1*w2$ triangularly–connected processors.

Consider the matrix multiplication problem shown in Figure 6. The triangular processor array used for this problem is the diamond shaped array shown in Figure 7 with the arrows indicating the direction of data flow of the matrices A, B and C.

The bands of the matrices move in three directions synchronously through the array. As each c_{ij} passes through, it accumulates its necessary terms as defined by the recurrences. The single step process of the computation is define as follows:

Stage 1: The a_{ij}, b_{ij} and c_{ij} move in their respective directions to the next cell in the network.

Stage 2: The recurrence equation is computed.

Stage 3: The a_{ij}, b_{ij} and c_{ij} move in their respective directions to the next cell in the network.

The single step computation process is a *move–compute–move* process. Figure 8 shows several four steps in the execution of the algorithm.

The data flows of the matrix multiplication algorithm executing on the triangular processor array are exactly the same as when it is executed on the hexagonal array except for the need of an extra move in the single step computation. This move is required to insure that the a_{ij}, b_{ij} and c_{ij} are in their correct positions at the correct time. Although this results in an increase in computation time of 50%, it is much less than the four and three times increase using a space and time emulation of the hexagonal processor array, respectively.

1.2. LU Decomposition on a Triangular Processor Array

Kung and Leiserson's LU decomposition algorithm also easily transfers to the triangular processor array [KuLe80]. We require, however, that bi-directional data movement across links be allowed.

LU decomposition is the problem of factoring a matrix A into lower and upper triangular matrices L and U. If we assume that the A has the property that its LU decomposition can be derived by Gaussian elimination without pivoting, the triangular matrices $L = [l_{ij}]$ and $U = [u_{ij}]$ are evaluated according to the following recurrences:

$$a_{ij}^{(1)} = a_{ij}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k+1)} + l_{ik} * (-u_{kj})$$

$$l_{ik} = 0 \text{ if } i<k$$

$$l_{ik} = 1 \text{ if } i=k$$

$$l_{ik} = a_{ik}^{(k)} * u_{kk}^{-1} \text{ if } i>k$$

$$u_{kj} = 0 \text{ if } k>j$$

$$u_{kj} = a_{kj}^{(k)} \text{ if } k \leq j$$

These recurrences can be pipelined on a triangular processor array with bi-directional links.

Consider the LU decomposition of the band matrix shown in Figure 9. The triangular processor array, shown in Figure 10, is constructed as follows. $2*w^2$ processors are needed for a band matrix of width w. These processors are arranges in a diamond shape with the shaped processor at the top representing a special processor. This

processor computes the reciprocal of its input and passes the result southwest and the same input northward unchanged. The inner processors below the upper boundaries are basic processors. The processors on the upper boundaries are also basic processors but the left boundary processors have an orientation rotated 120 degrees clockwise and the right boundary processors are oriented 120 degrees counterclockwise.

As in the matrix multiplication algorithm, the single step computation is a *move-compute-move* process. The data flows in the network as indicated by the arrows. Figurer 11 shows four steps in the execution of the LU decomposition algorithm on the triangular array. Again, the performance is better than if a space or time emulation scheme was used.

2. CONCLUSION

Processor interconnection topologies incorporating communication and spatial regularity will become increasingly important as VLSI dimensions continue to decrease. The organization of VLSI cells will become much more structured with simple, regular, and uniform geometries being more efficient. Although mesh processor arrays have known scalability limitations with respect to communication [Witt81] [Reed83], several recent reports suggest that the communication efficiency of two dimensional meshes is greater than other interconnection topologies when compared for VLSI implementation [Mazu86] [RaJo87].

The regular processor arrays described in this paper have an aesthetic appeal of simplicity and ordered structure. They 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. The emulations efficiencies provide measures for evaluating the regular processor arrays.

A natural extension of the work presented here concerns regular three dimension organizations [Zalg69]. The exact geometrical interpretation of regularity in three dimensional processor arrays is not clear. However, the volume–filling nearest neighbor networks may be an appropriate place to begin analyzing such topologies. The increased spatial flexibility in three dimensions makes the problem significantly harder. Nevertheless, research in this area will become more important and necessary as VLSI begins to offer three dimensional interconnects.

REFERENCES

[AmEp74] Serafino Amoroso and Irving J. Epstein. *Maps Preserving the Uniformity of Neighborhood Interconnection Patterns in Tesselation Structures.* **Info. and Control**, Vol. 25, 1974, pp. 1–9.

[BBKK68] 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.

[BeSn84] F. Berman and L. Snyder. *On Mapping Parallel Algorithms into Parallel Architectures. Proc. of the 1984 Inter. Conf. on Parallel Processing*, Aug. 1984, pp. 307–309.

[Bokh81] S.H. Bokhari. *On the Mapping Problem.* **IEEE Trans. Comput.**, Vol. C–30, March 1981, pp. 207–214.

[Borr68] J. Borrego. *Space Grid Structures.* MIT Press, Mass., 1968.

[ChFi82] T.L. Chang and P. David Fisher. *Programmable Systolic Arrays.* **Proc. 9th Symp. on Comput. Arch.**, 1982, pp. 48–53.

[Comp87] **Computer** (special issue on Systolic Arrays), Vol 20, No. 7, July 1987, pp. 12–17.

[Crit69] K. Critchlow. *Order in Space.* Viking Press, New York, 1969.

[Dumn71] J.A. Dumn. *Tesselation with Pentagons.* **Math Gazette**, Vol. 55, No. 394, Dec. 1971, pp. 366–369.

[FoKu80] M.J. Foster and H.T. Kung. *The Design of Special–Purpose VLSI Chips.* **Computer**, Jan. 1980, pp. 26–40.

[Gard72] M. Gardner. *On Tesselating the Plane with Convex Polygon Tiles.* **Scientific American**, Vol. 56, No. 398, Dec. 1972, pp. 332–335.

[GoGr84] Naga S. Gollakota and F.Gail Gray. *Reconfigurable Cellular Architecture.* **Proc. 11th Int. Symp. Computer Arch.**, 1984, pp. 377–379.

[GoKS84] 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.

[GrSh77] 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.

[GrSh80] 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.

[KaLW68] 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.

[Kers68] R.B. Kershner. *On Paving the Plane.* **Amer. Math Monthly**, Vol. 75, No. 8, Oct. 1968, pp. 839–844.

[KoSi83] I. Koren and G.M. Silberman. *A Direct Mapping of Algorithms onto VLSI Processor Arrays Based on the Data Flow Approach.* **Proc. of the Inter. Conf. on Parallel Proc.**, Aug. 1983, pp. 335–337.

[Kuck68] D.J. Kuck. *ILLIAC IV Software and Applications Programming.* **IEEE Trans. Comp.**, Vol. C–17, No. 8, Aug. 1968, pp. 758–770.

[Kung79] H.T. Kung. *Let's Design Algorithms for VLSI Systems.* **Caltech Conf. on VLSI**, Jan. 1979, pp. 65–90.

[Kung82] H.T. Kung. *Why Systolic Architectures?.* **Computer**, Vol. 15, No. 1, Jan. 1982, pp. 97–107.

[KuLe80] 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.

[Lave31] F. Laves. *Ebenenteilung und Koordinationszahl.* **Z. Kristallogr.**, Vol. 78, 1931, pp. 208–241.

[LeKa72] K.N. Levitt and W.H. Kautz. *Cellular Arrays for the Solution of Graph Problems.* **CACM**, Vol.15, No. 9, 1972, pp. 789–801.

[Loeb76] A.L. Loeb. *Space Structures.* Addison–Wesley, Reading, Mass., 1976.

[Malo82] Allen D. Malony. *Regular Interconnection Networks.* Master's Thesis, Univ. of California, Los Angeles, August 1982.

[Mazu86] Pinaki Mazumder. *Evaluation of Three Interconnection Networks for CMOS VLSI Implementation.* **1986 ICPP**, Aug. 1986, pp. 200–207.

[MeCo80] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison–Wesley, Reading, Mass., 1980.

[MeSi87] Bilha Mendelson and Gabriel M. Silberman. *Mapping Data Flow Programs on a VLSI Array of Processors.* ?, 1987, pp. 72–80.

[MoFo86] D.I. Moldovan and J.A.B. Fortes. *Partitioning and Mapping Algorithms Into Fixed–Size Systolic Arrays.* **IEEE Trans. Comput.**, Vol. C–35, No. 1, Jan. 1986, pp. 1–12.

[NaSa79] D. Nassimi and S. Sahni. *Bitonic Sort on a Mesh–Connected Parallel Computer.* **IEEE Trans. Comput.**, Vol. C–27, Jan. 1979, pp. 2–7.

[Orcu76] 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.

[RaJo87] 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.

[ReAP87] 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.**.

[ReSc83] 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.

[Sava81] 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.

[Snyd82] L. Snyder. *Introduction to the Configurable, Highly Parallel Computer.* **Computer**, Vol 15, No. 1, Jan. 1982, pp. 47–56.

[Subn16] A. Subnikov. *K voprosu o stroenii Kristallov.* **Bulletin Acad. Imp. Sci.**, Ser. 6, Vol. 10, 1916, pp. 755–779.

[SuMe77] I.E. Sutherland and C.A. Mead. *Microelectronics and Computer Science.* **Scientific American**, Vol. 237, Sept. 1977, pp. 210–228.

[Thom79] C.D. Thompson. *Area–Time Complexity for VLSI.* **Proc. Caltech Conf on VLSI**, Jan. 1979, pp. 405–508.

[ThKu77] C.D. Thompson and H.T. Kung. *Sorting on a Mesh–Connected Parallel Computer.* **CACM**, Vol. 20, April 1977, pp. 263–271.

[Ullm84] J.D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[Venk85] N. Venkateswaran. *PA^9 Arrays for Supercomputers.* **Proc. 1985 IEEE Supercomputer Conf.**, 1985, pp. 20–30.

[vonN68] J. von Neumann. *The Theory of Self–Reproducing Automa*, Univ. of Illinois at Urbana–Champaign, 1968.

[Witt81] L.D. Wittie. *Communications Structures for Large Networks of Microcomputers.* **IEEE Trans. Comput.**, Vol. C–30, No. 4, April 1981, pp. 264–273.

[YaAm69] H. Yamada and S. Amoroso. *Tessellation Automata.* **Info. and Control**, Vol. 14, 1969, pp. 299–317.

[YaAm71] H. Yamada and S. Amoroso. *Structural and Behavorial Equivalences of Tesselation Automata.* **Info. and Control**, Vol. 18, 1971, pp. 1–31.

[Zalg69] Viktor A. Zalgaller. *Convex Polyhedra with Regular Faces.* in **Seminars in Mathematics**, Vol. 2, V.A. Steklov (editor), Math. Inst., Leningrad, Consultants Bureau, New York, 1969, pp. 1–10.

Triangular Host Topology – Space Emulation Efficiency		
Target Topology	Optimal Emulation (host nodes / target node)	Emulation Efficiency (host nodes / target node)
3^6	4	4
$3^4.6$	3	3
$3^3.4^2$	3	3
$3^2.4.3.4$	3	3
3.4.6.4	2	3
3.6.3.6	2	2
3.12^2	1	2
4^4	2	2
4.6.12	1	$2\frac{2}{3}$
4.8^2	1	2
6^3	1	1

Table 1. Space Emulation Efficiency for Triangular Host Array

Orthogonal Host Topology – Space Emulation Efficiency		
Target Topology	Optimal Emulation (host nodes / target node)	Emulation Efficiency (host nodes / target node)
3^6	2	2
$3^4.6$	2	$2\frac{1}{3}$
$3^3.4^2$	2	2
$3^2.4.3.4$	2	$2\frac{1}{4}$
3.4.6.4	1	$1\frac{1}{3}$
3.6.3.6	1	$1\frac{1}{3}$
3.12^2	1	2
4^4	1	1
4.6.12	1	$1\frac{1}{3}$
4.8^2	1	1
6^3	1	1

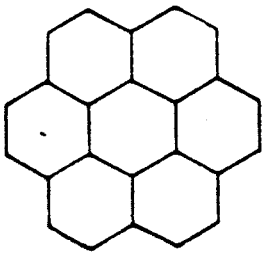
Table 2. Space Emulation Efficiency for Orthogonal Host Array

Hexagonal Host Topology – Space Emulation Efficiency		
Target Topology	Optimal Emulation (host nodes / target node)	Emulation Efficiency (host nodes / target node)
3^6	1	1
$3^4.6$	1	$1\frac{1}{6}$
$3^3.4^2$	1	1
$3^2.4.3.4$	1	1
3.4.6.4	1	1
3.6.3.6	1	$1\frac{1}{3}$
3.12^2	1	1
4^4	1	1
4.6.12	1	1
4.8^2	1	1
6^3	1	1

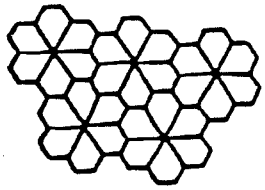
Table 3. Space Emulation Efficiency for Hexagonal Host Array

Time Emulation Efficiency of Strongly Regular Arrays			
Host Topology	Emulation Efficiency (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)

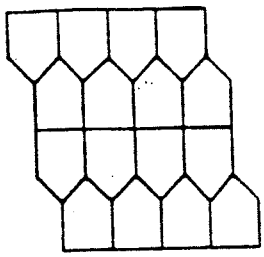
Table 4. Time Emulation Efficiency of Regular Target Arrays



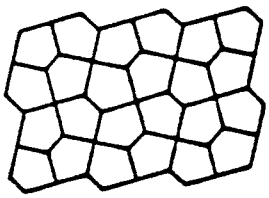
3^6



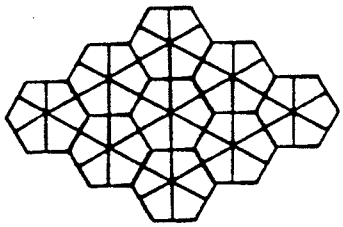
$3^4.6$



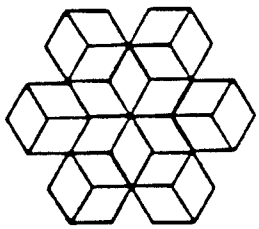
$3^3.4^2$



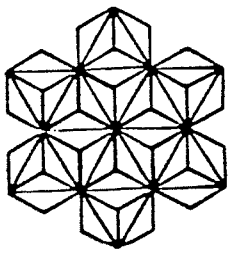
$3^2.4.3.4$



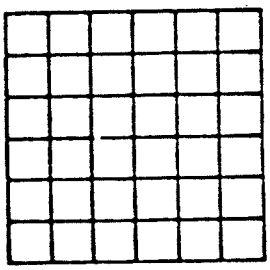
3.4.6.4



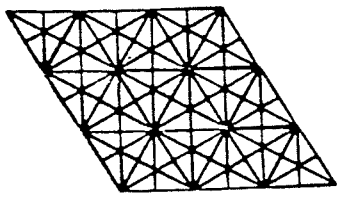
3.6.3.6



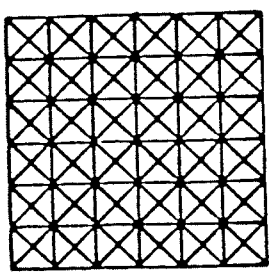
3.12^2



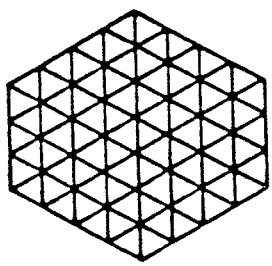
4^4



4.6.12



4.8^2



6^3

Figure 1. The Eleven Laves Nets

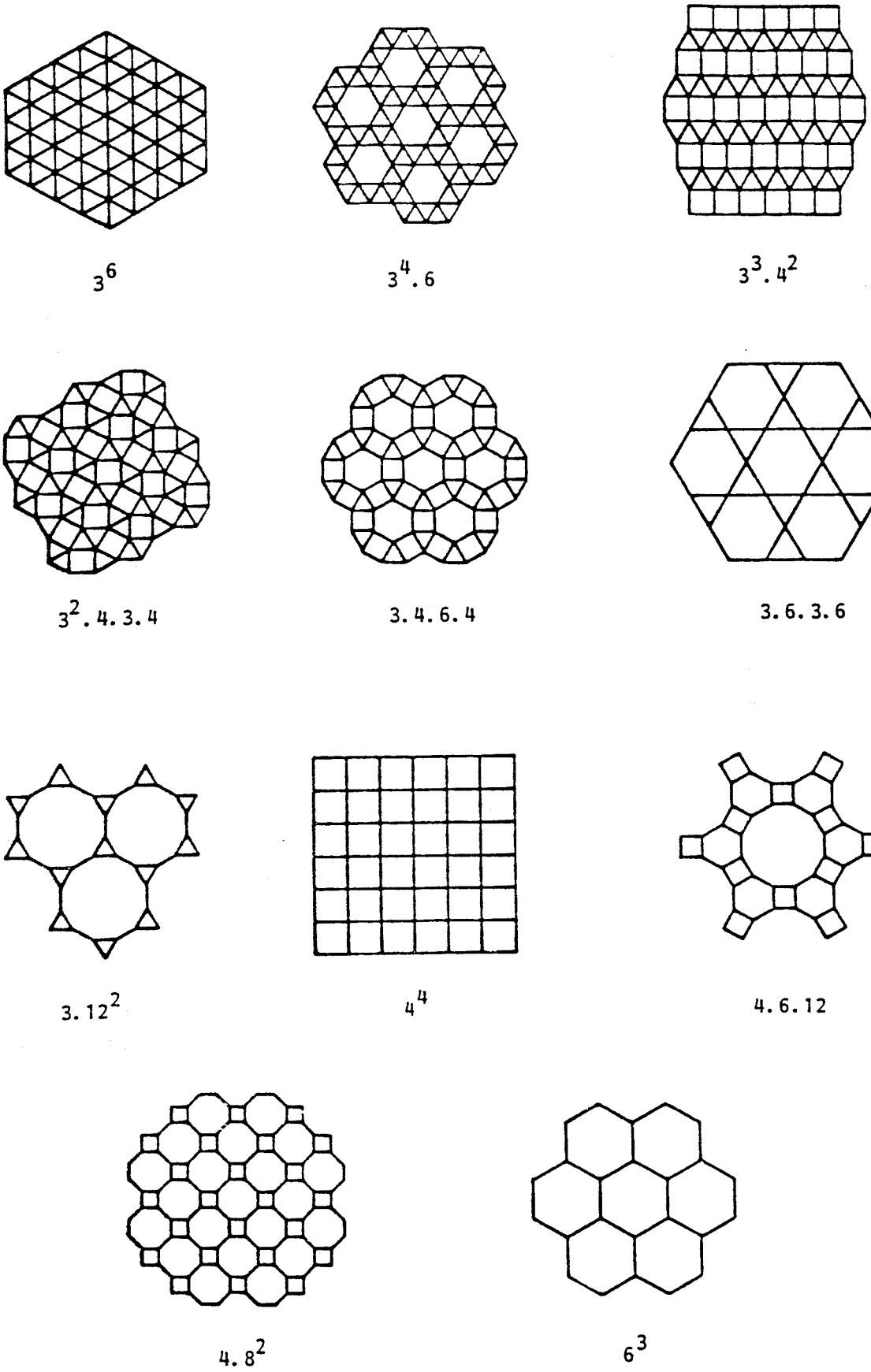


Figure 2. The Eleven Regular Processor Arrays

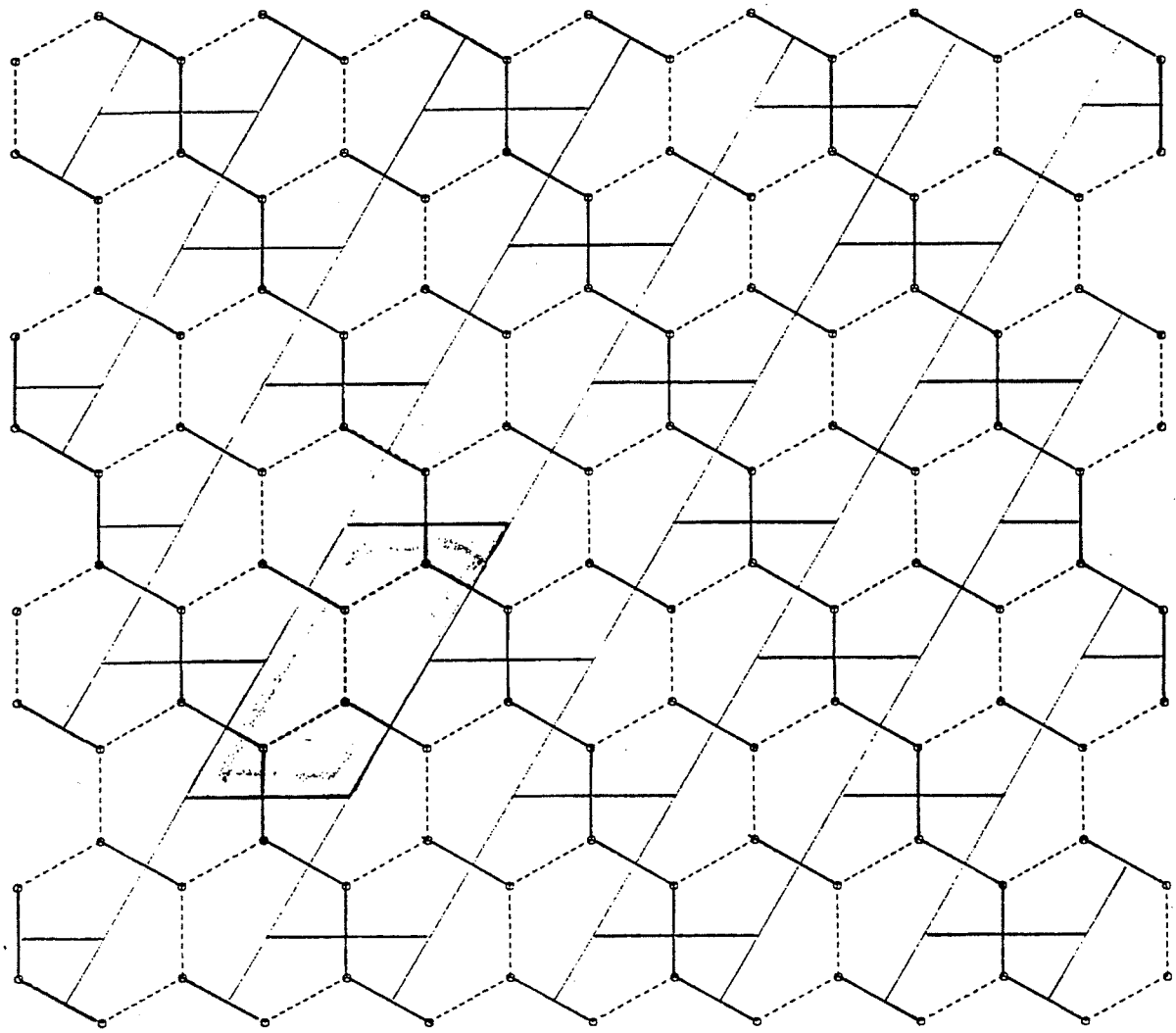


Figure 3.1 Triangular Space Emulation of 3^6

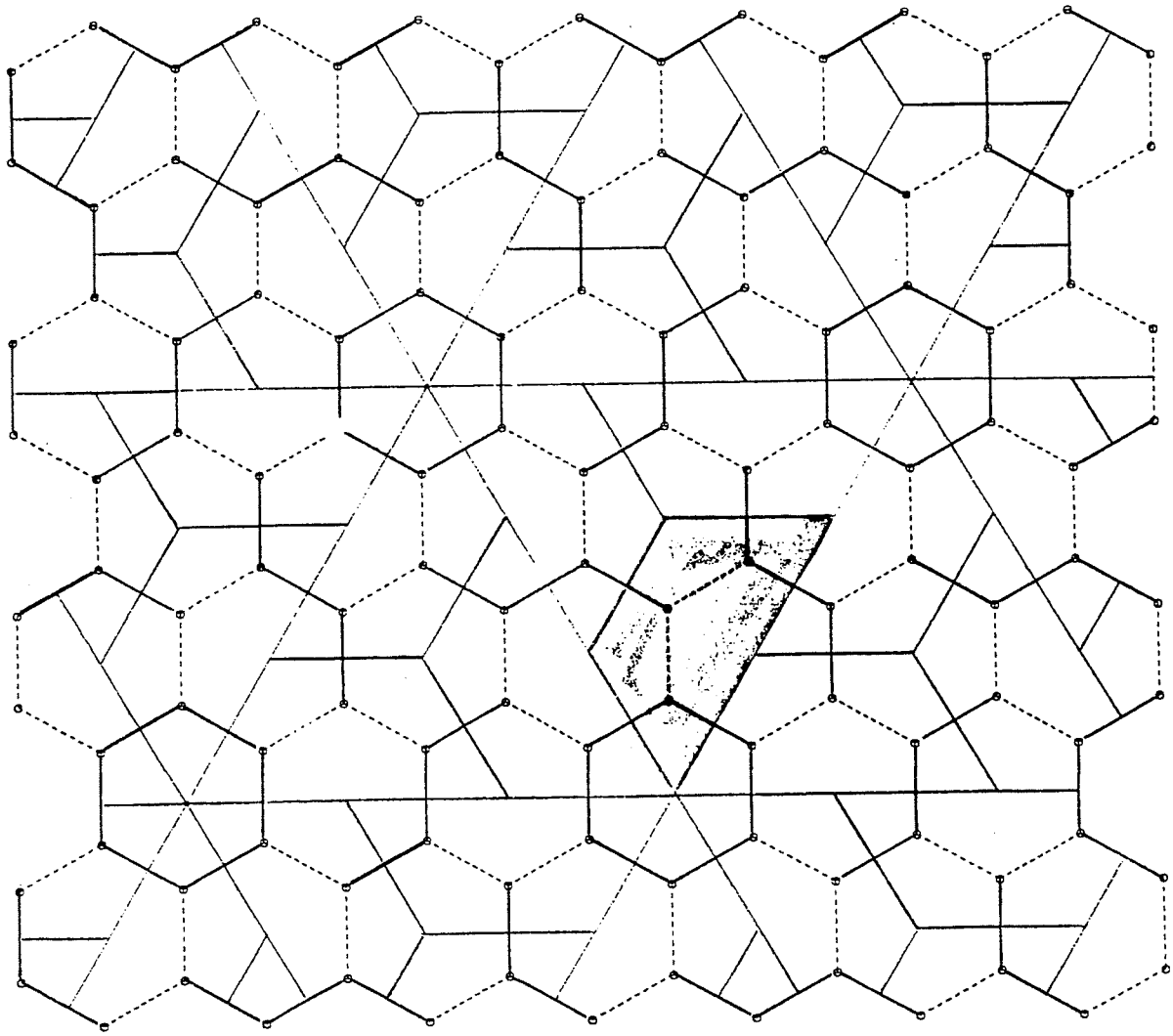


Figure 3.2 Triangular Space Emulation of $3^4.6$

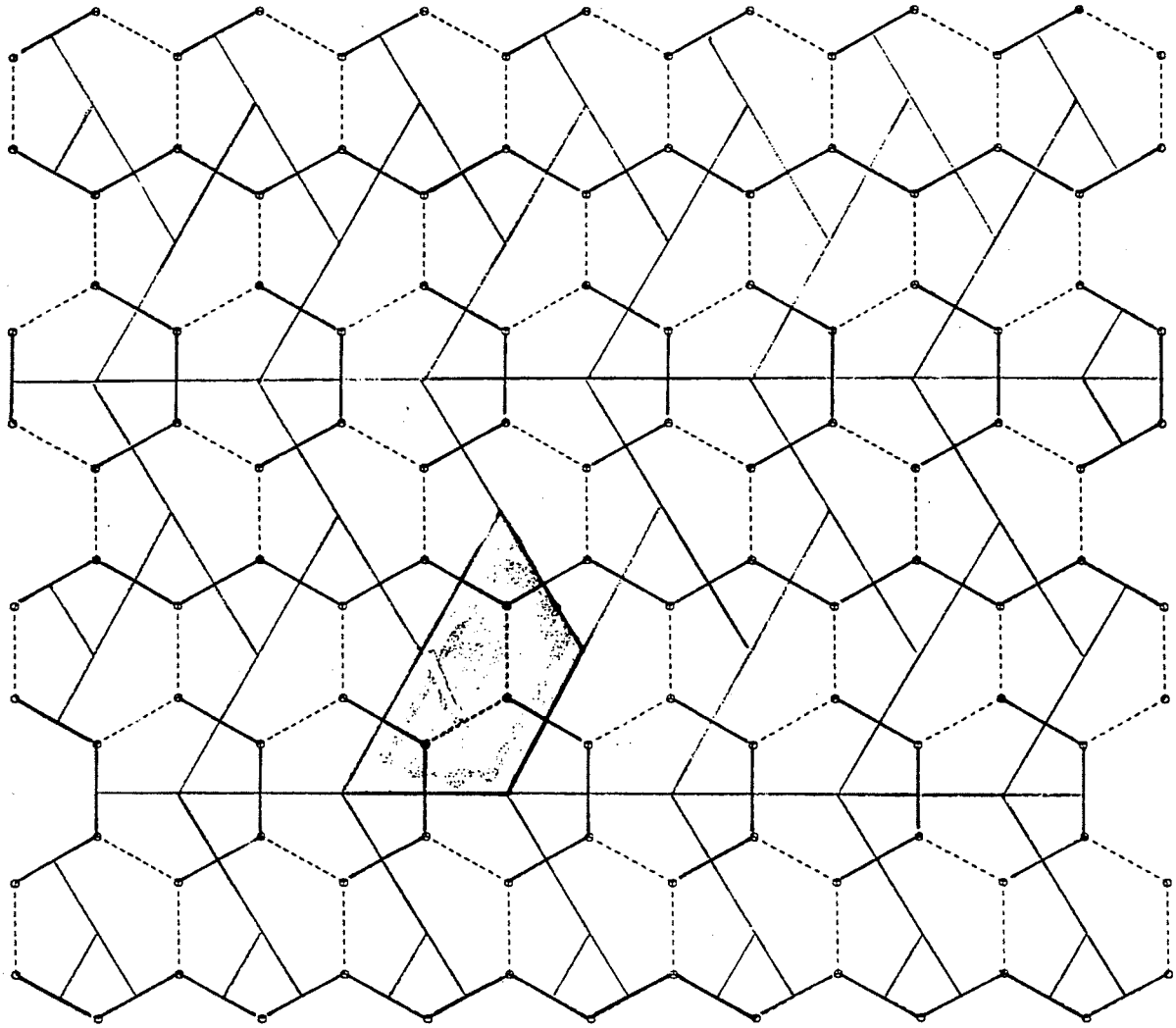


Figure 3.3 Triangular Space Emulation of $3^3.4^2$

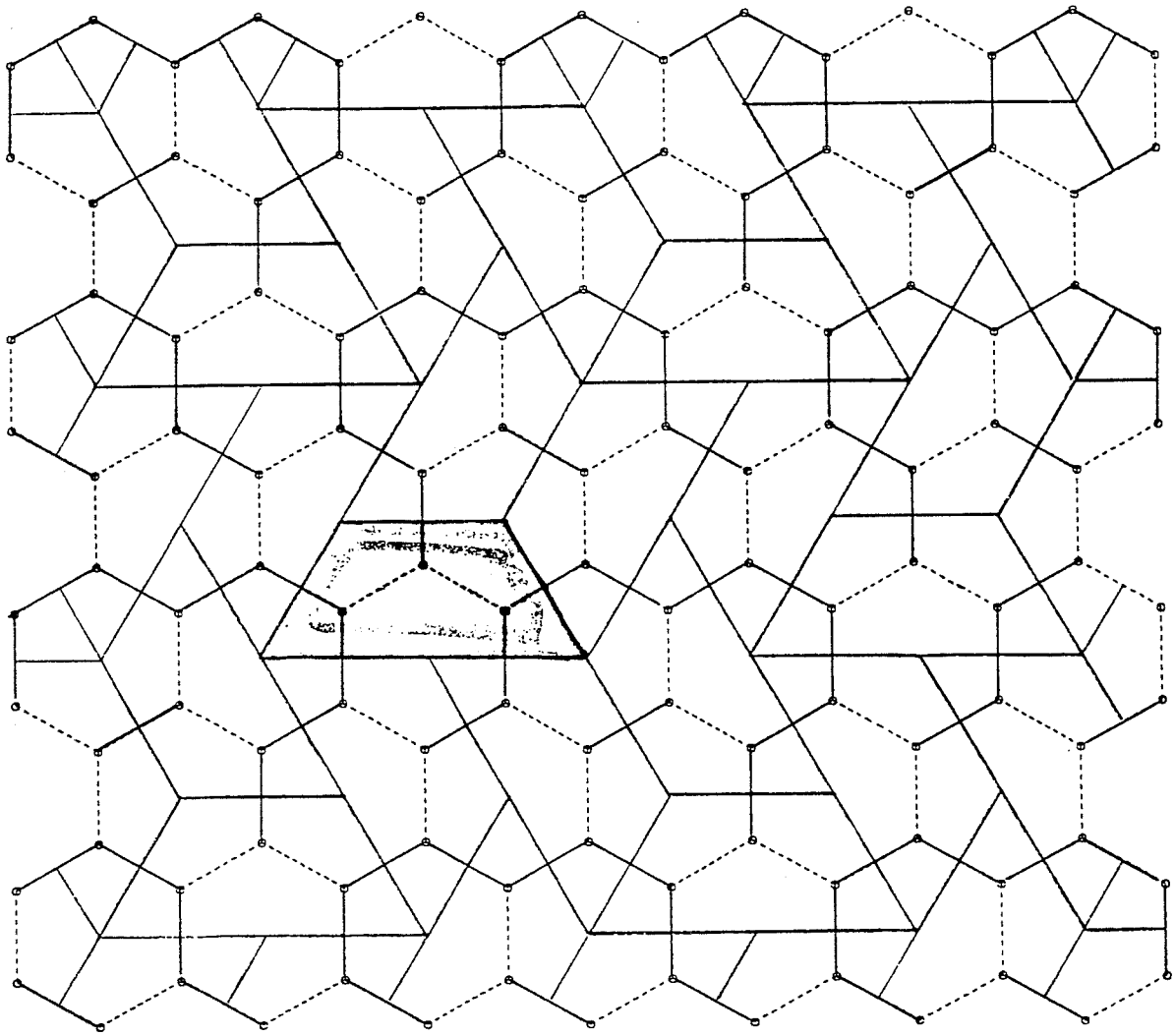


Figure 3.4 Triangular Space Emulation of $3^2.4.3.4$

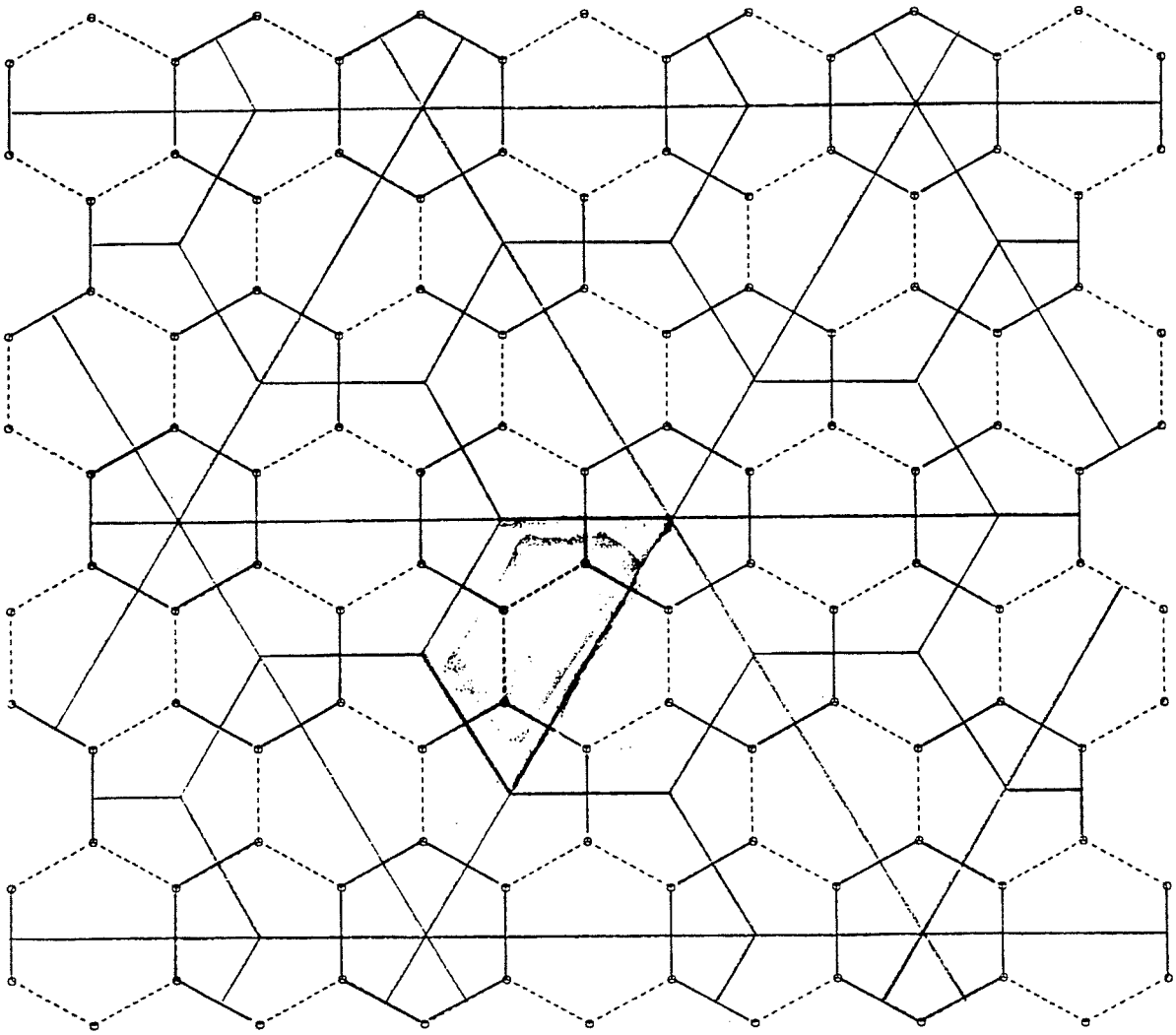


Figure 3.5 Triangular Space Emulation of 3.4.6.4

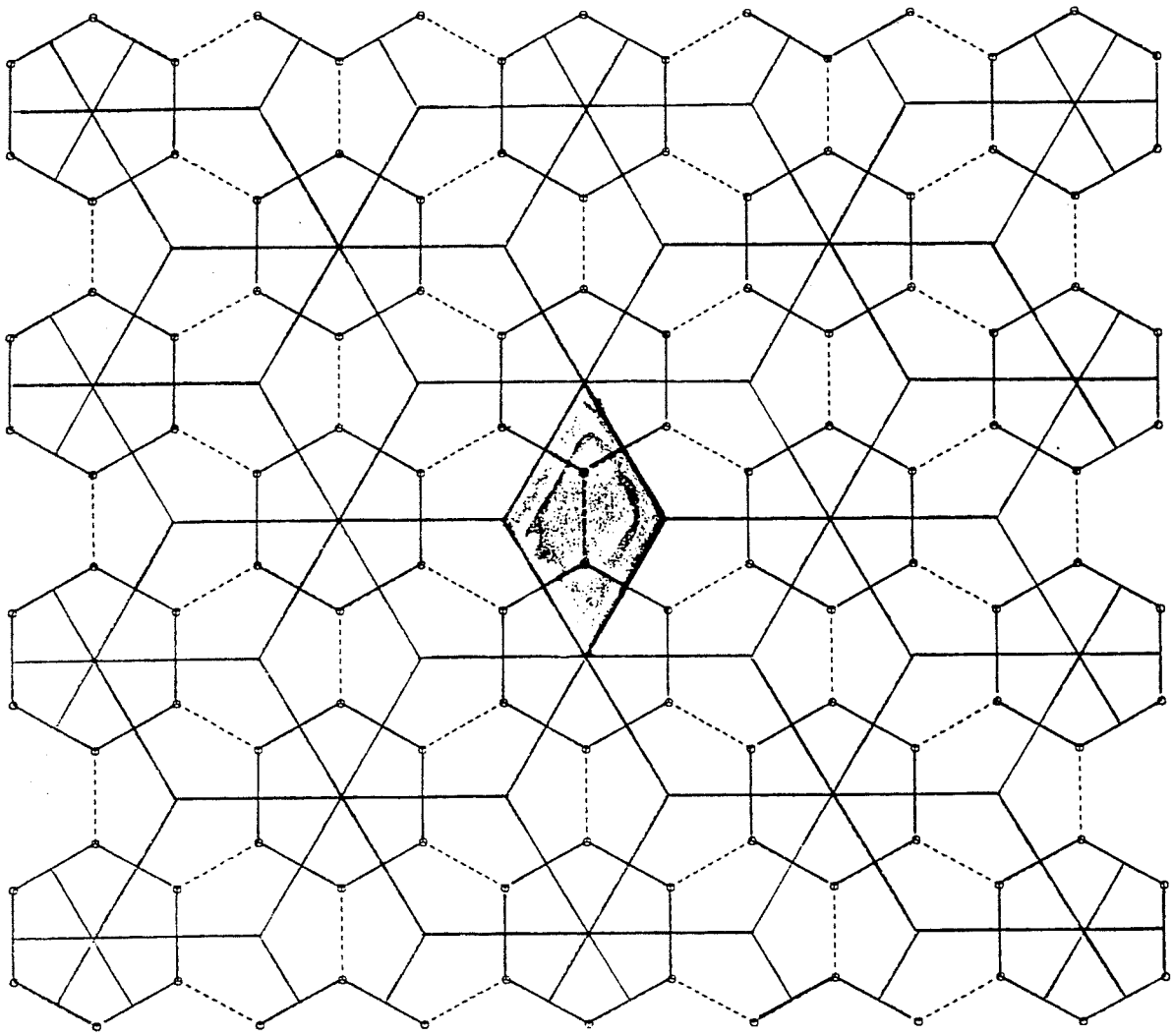


Figure 3.6 Triangular Space Emulation of 3.6.3.6

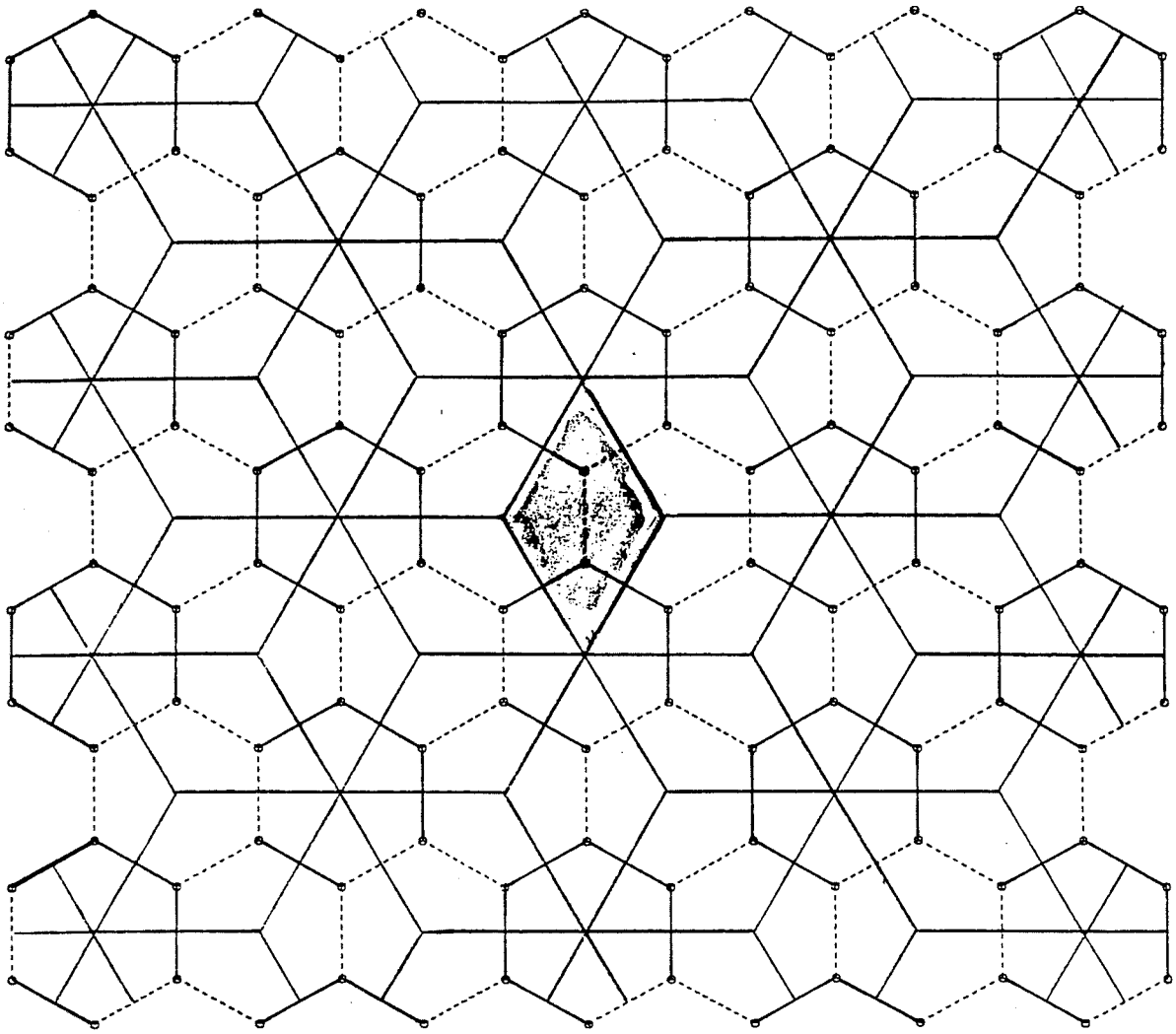


Figure 3.7 Triangular Space Emulation of 3.12^2

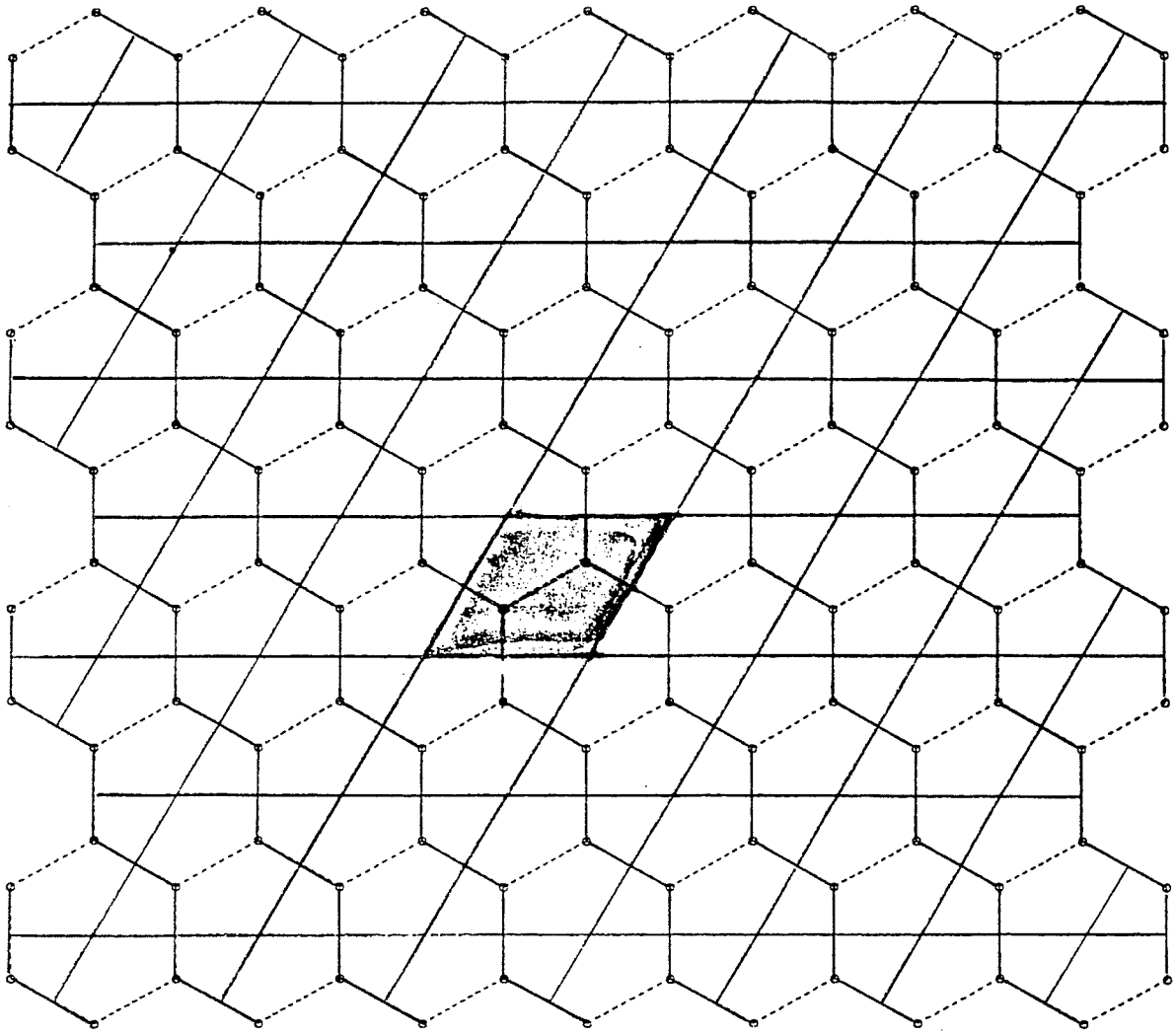


Figure 3.8 Triangular Space Emulation of 4^4

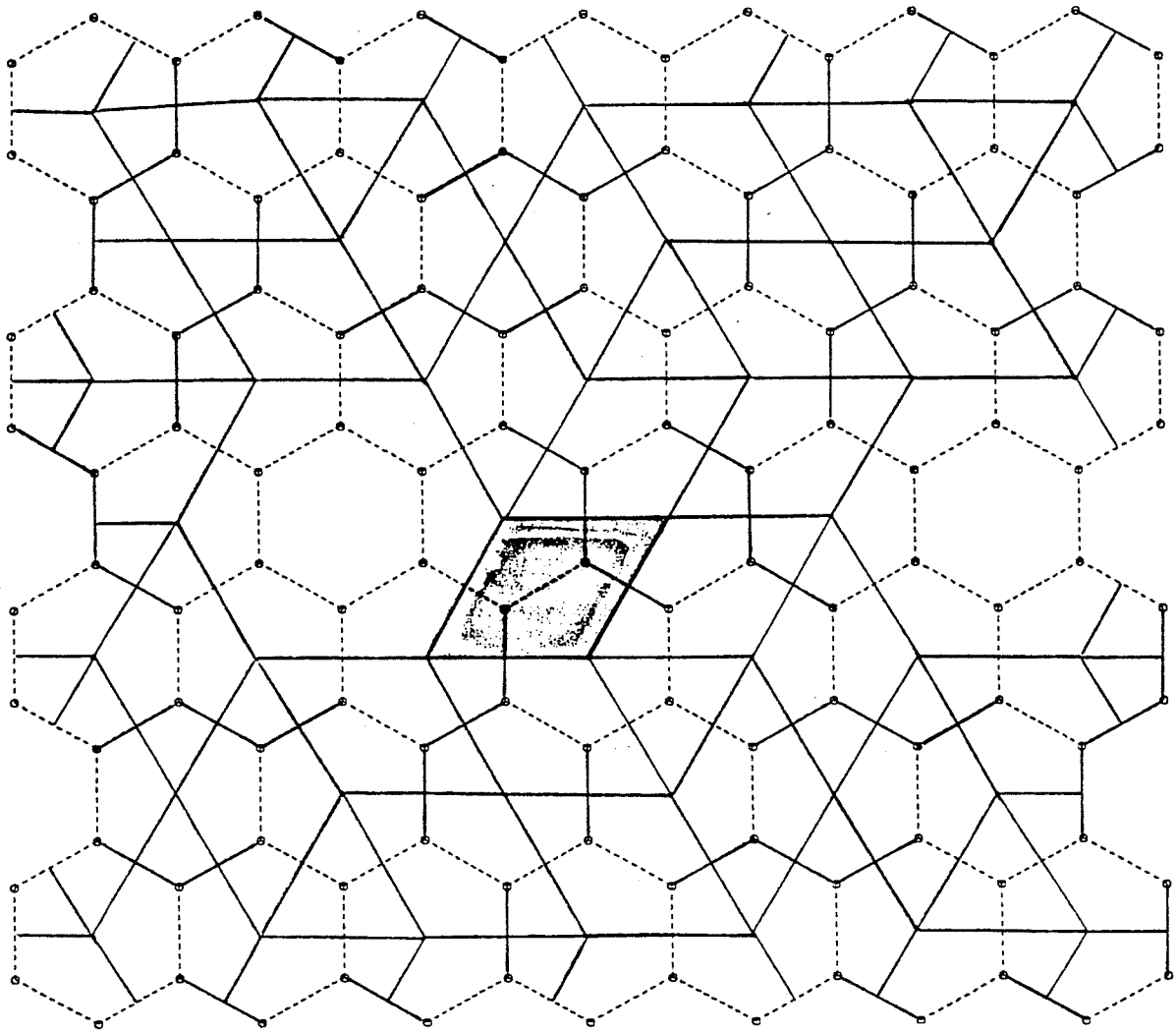


Figure 3.9 Triangular Space Emulation of 4.6.12

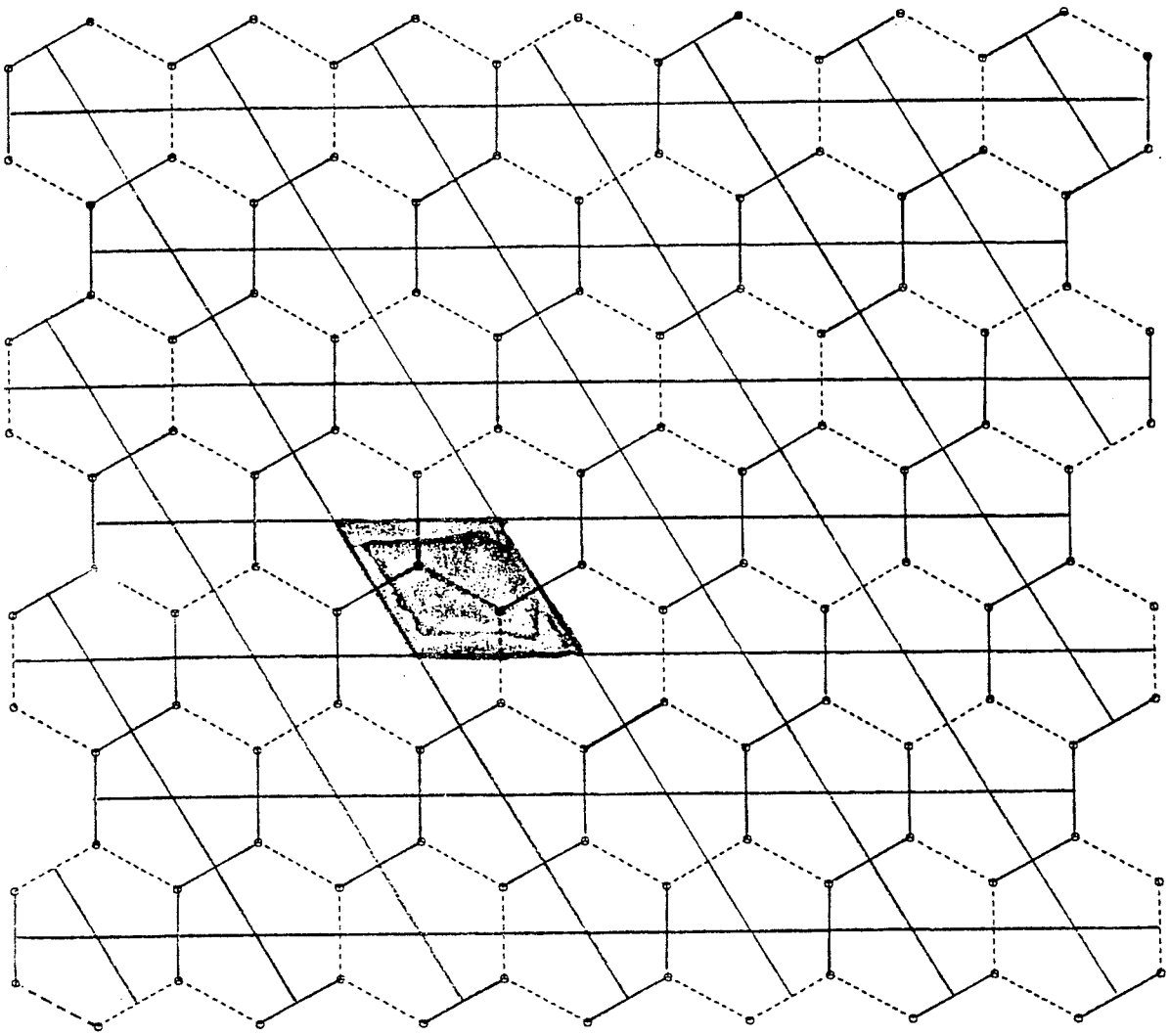


Figure 3.10 Triangular Space Emulation of 4.8^2

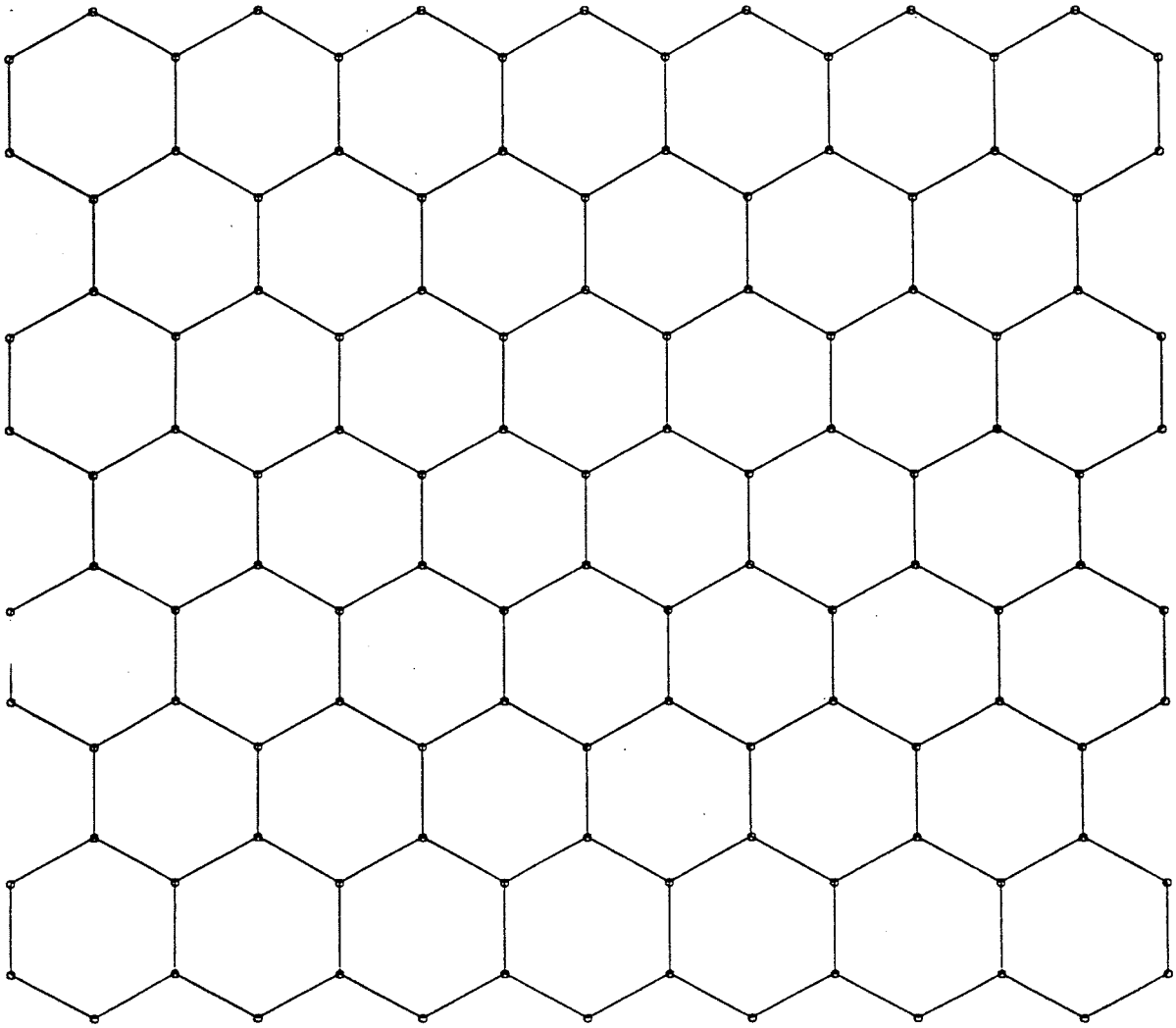


Figure 3.11 Triangular Space Emulation of 6^3

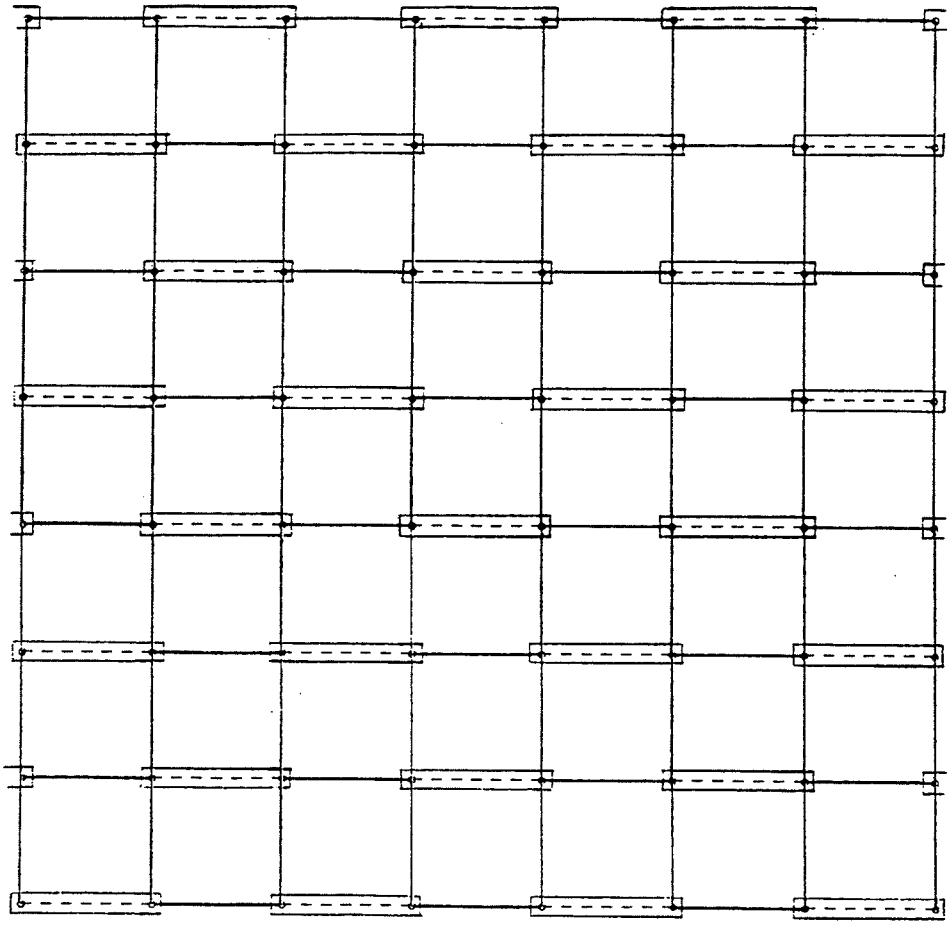


Figure 4.1 Orthogonal Space Emulation of 3^6

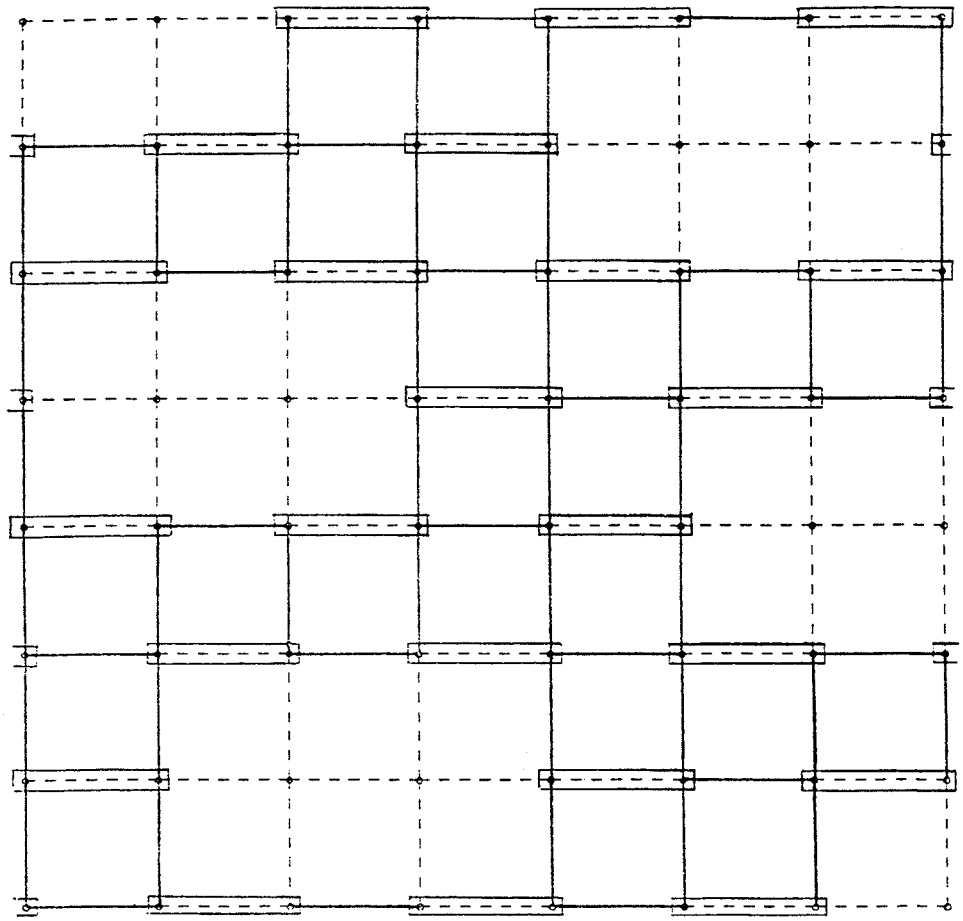


Figure 4.2 Orthogonal Space Emulation of $3^4.6$

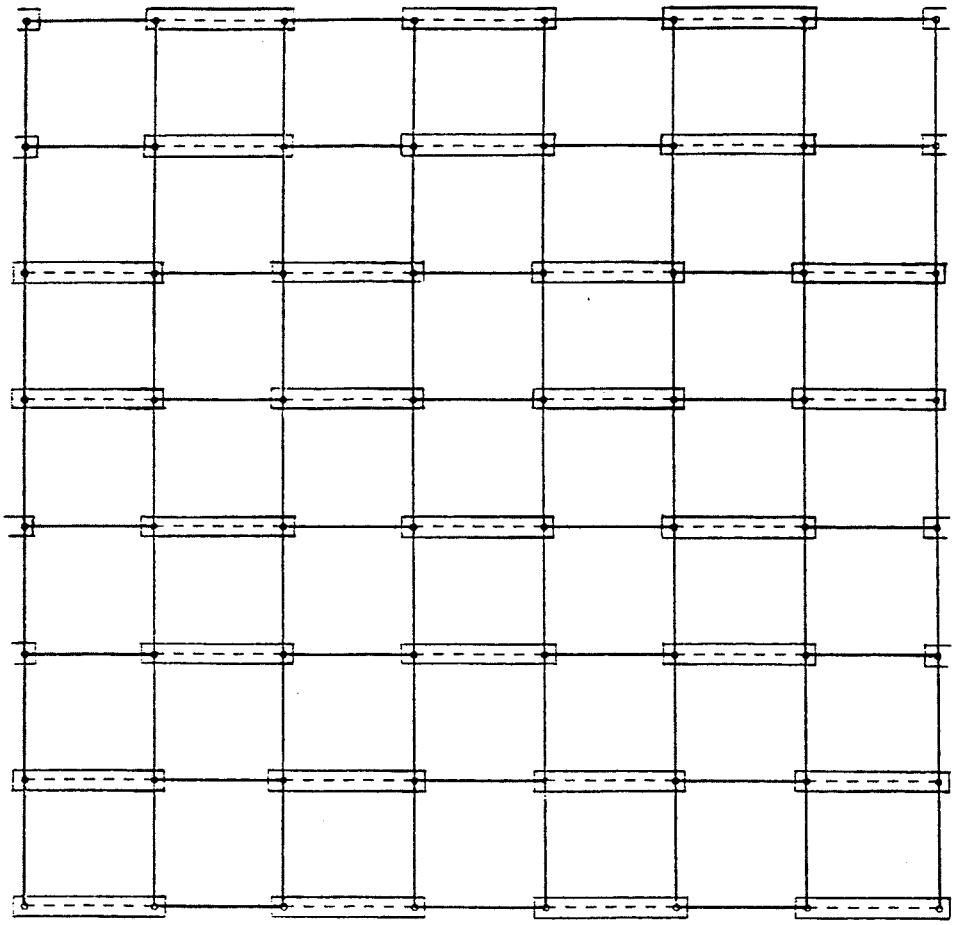


Figure 4.3 Orthogonal Space Emulation of $3^3.4^2$

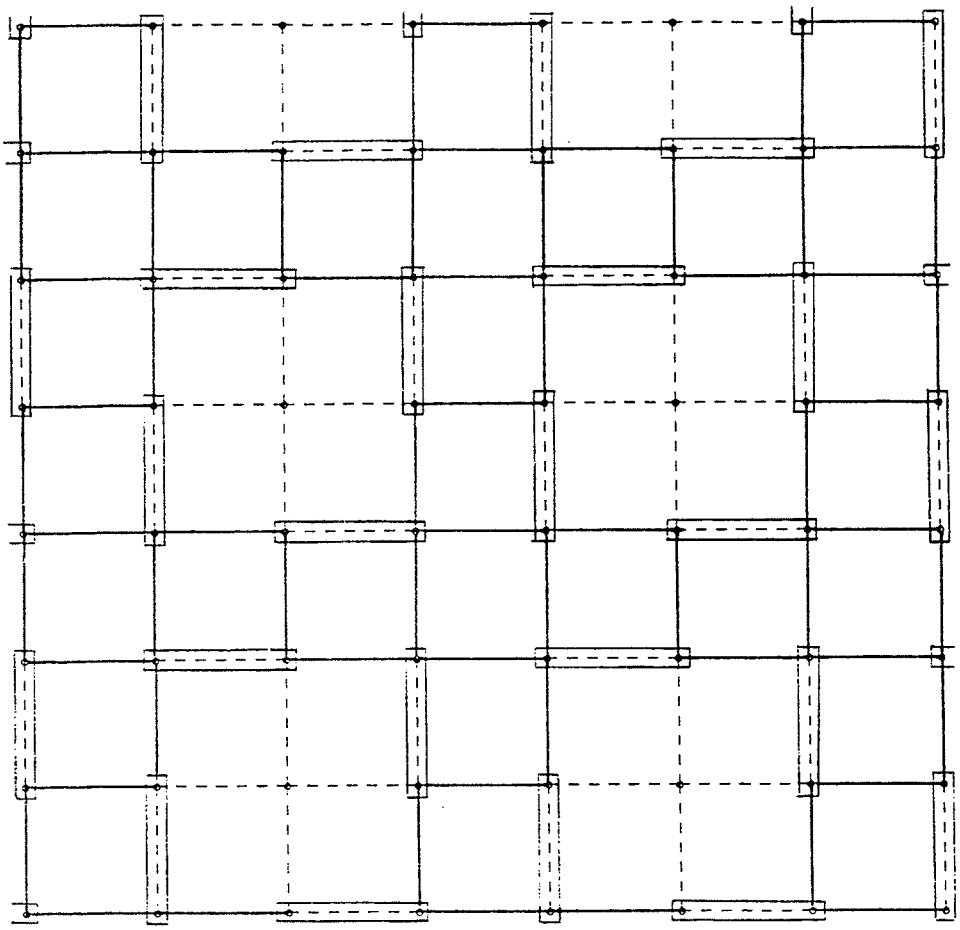


Figure 4.4 Orthogonal Space Emulation of $3^2.4.3.4$

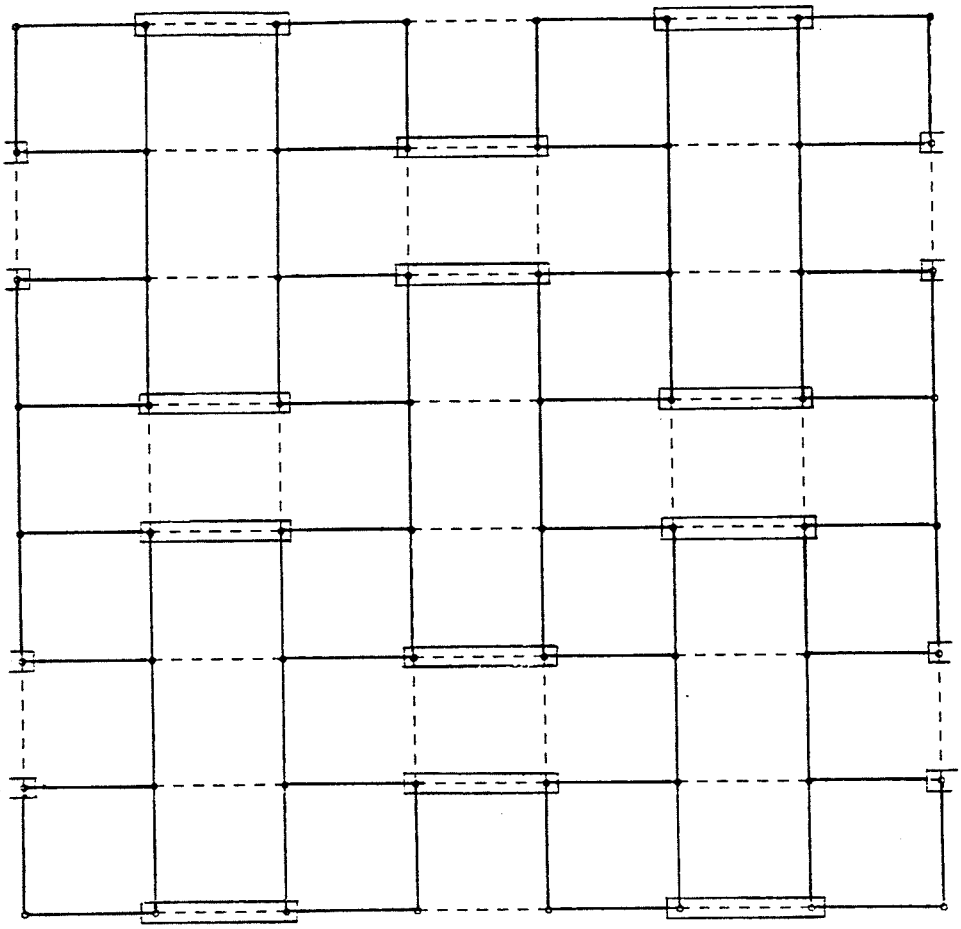


Figure 4.5 Orthogonal Space Emulation of 3.4.6.4

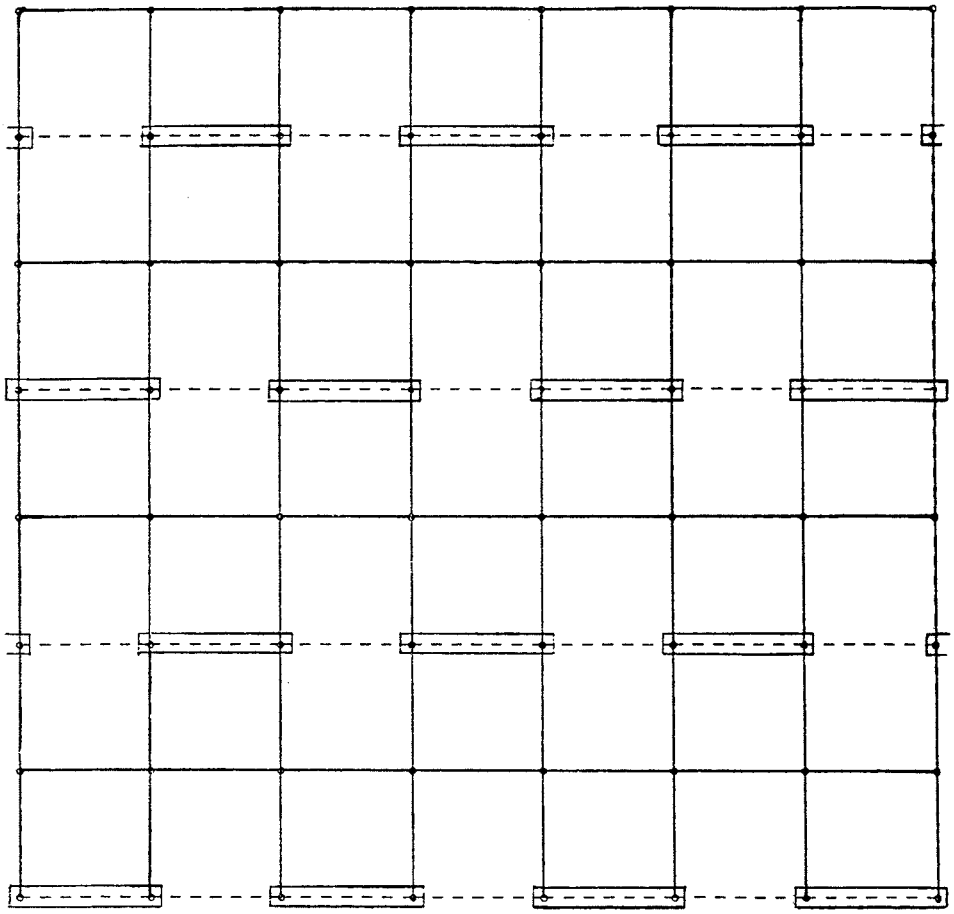


Figure 4.6 Orthogonal Space Emulation of 3.6.3.6

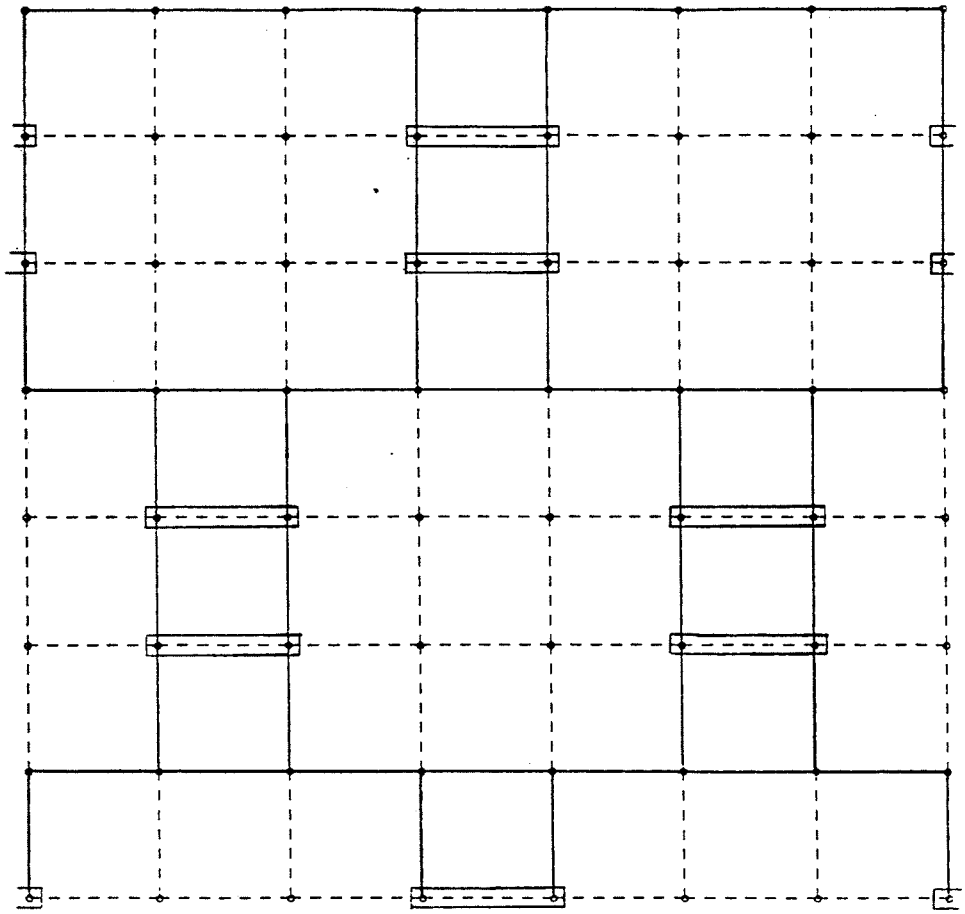


Figure 4.7 Orthogonal Space Emulation of 3.12^2

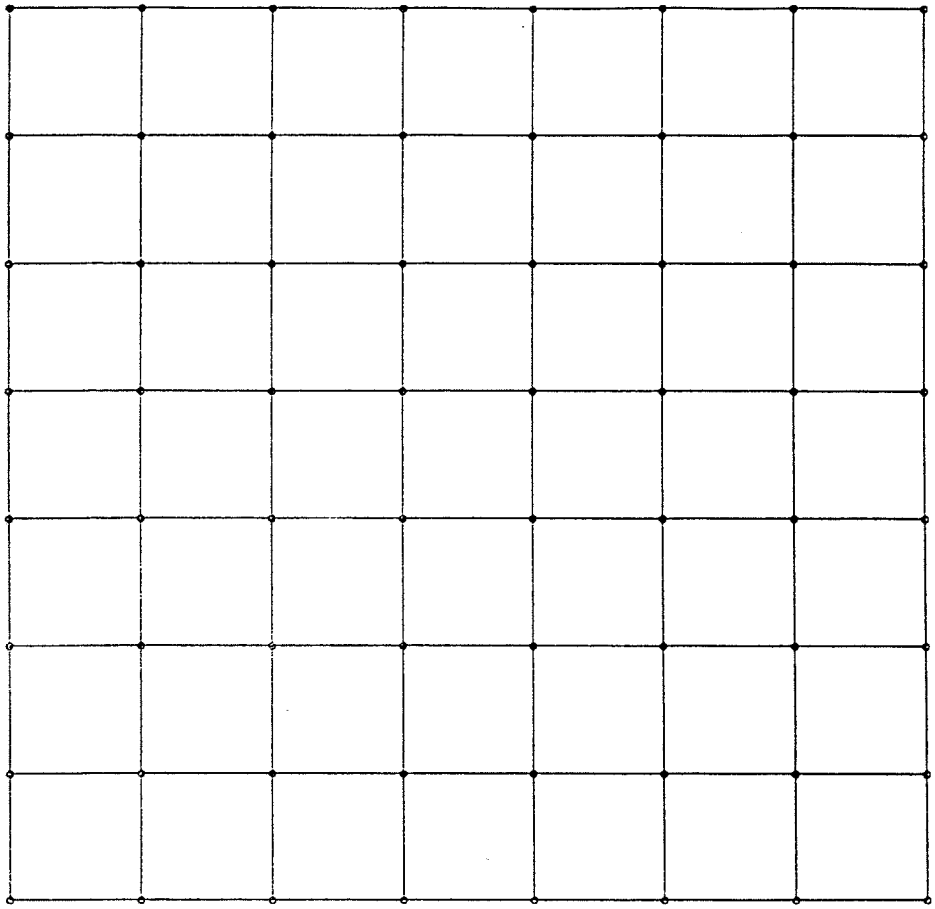


Figure 4.8 Orthogonal Space Emulation of 4^4

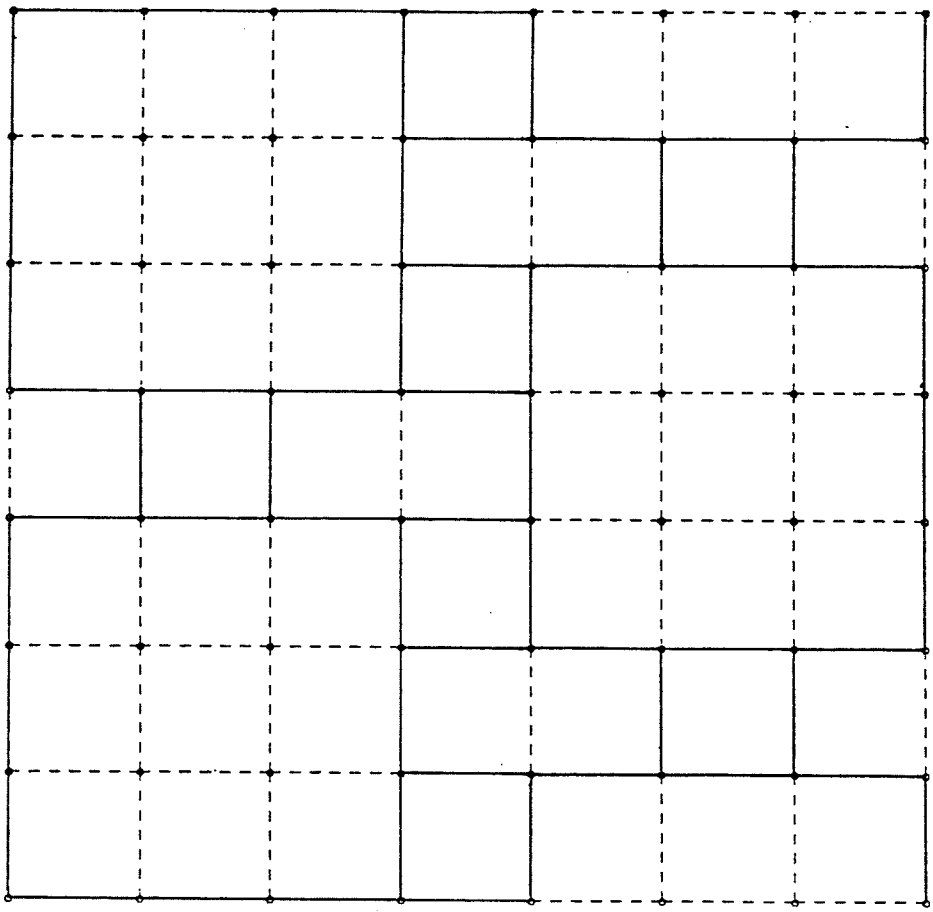


Figure 4.9 Orthogonal Space Emulation of 4.6.12

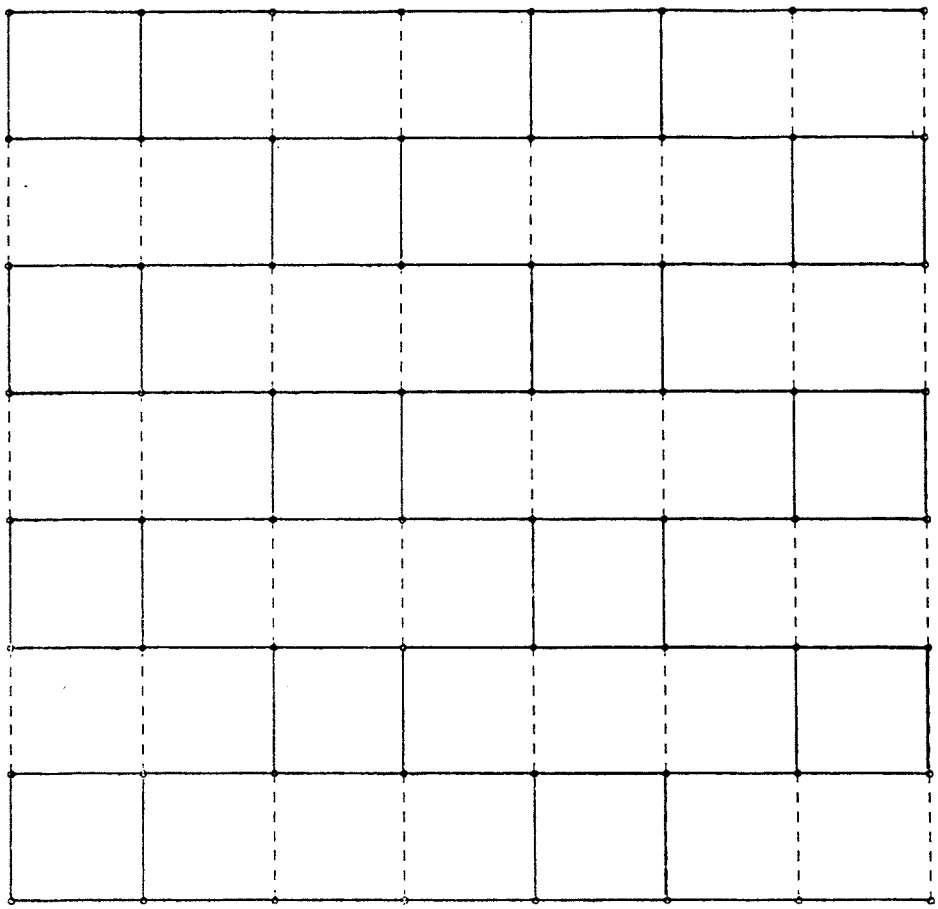


Figure 4.10 Orthogonal Space Emulation of 4.8^2

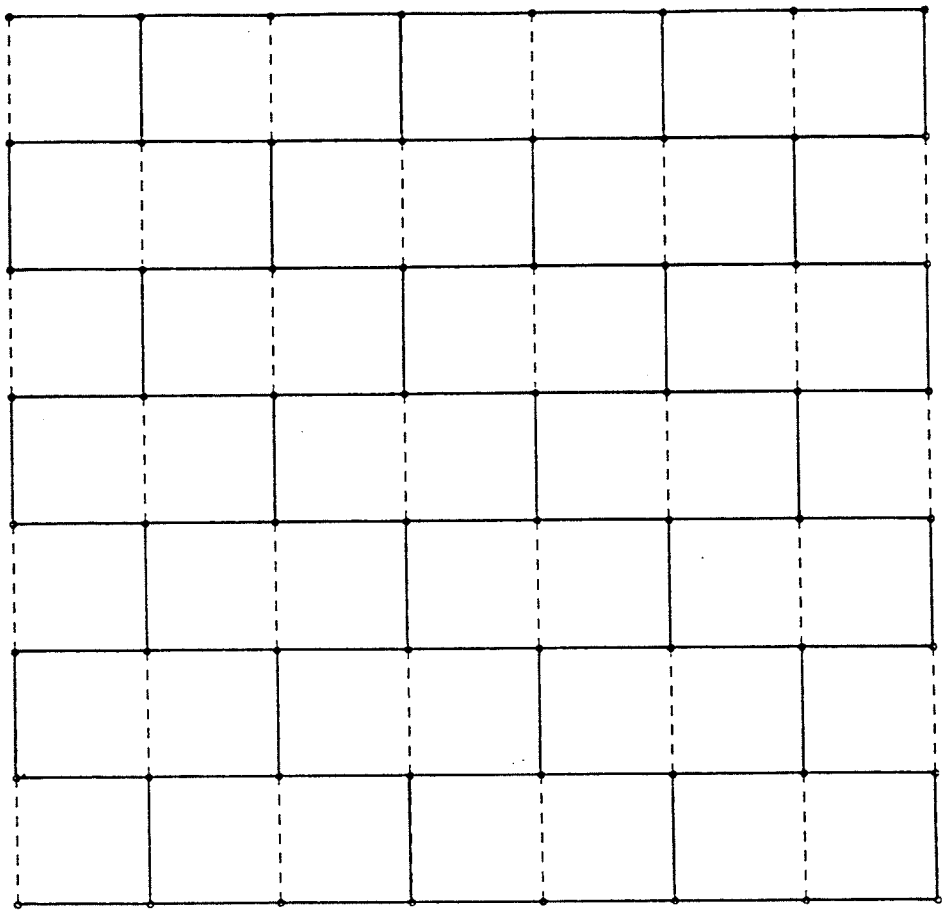


Figure 4.11 Orthogonal Space Emulation of 6^3

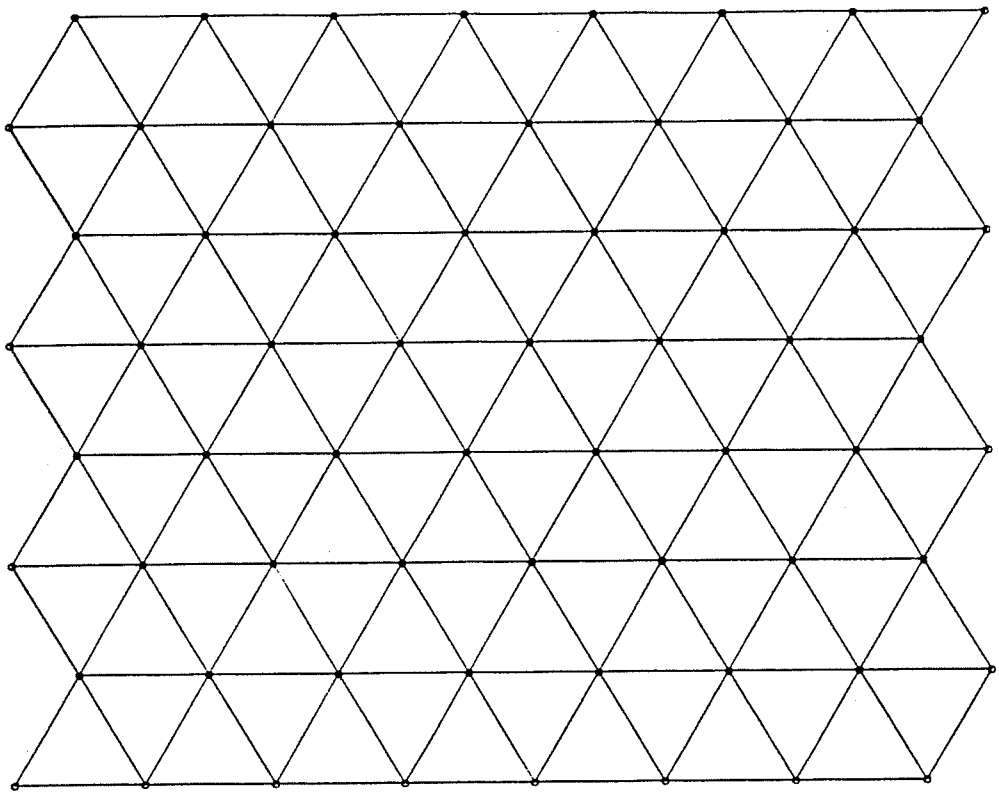


Figure 5.1 Hexagonal Space Emulation of 3^6

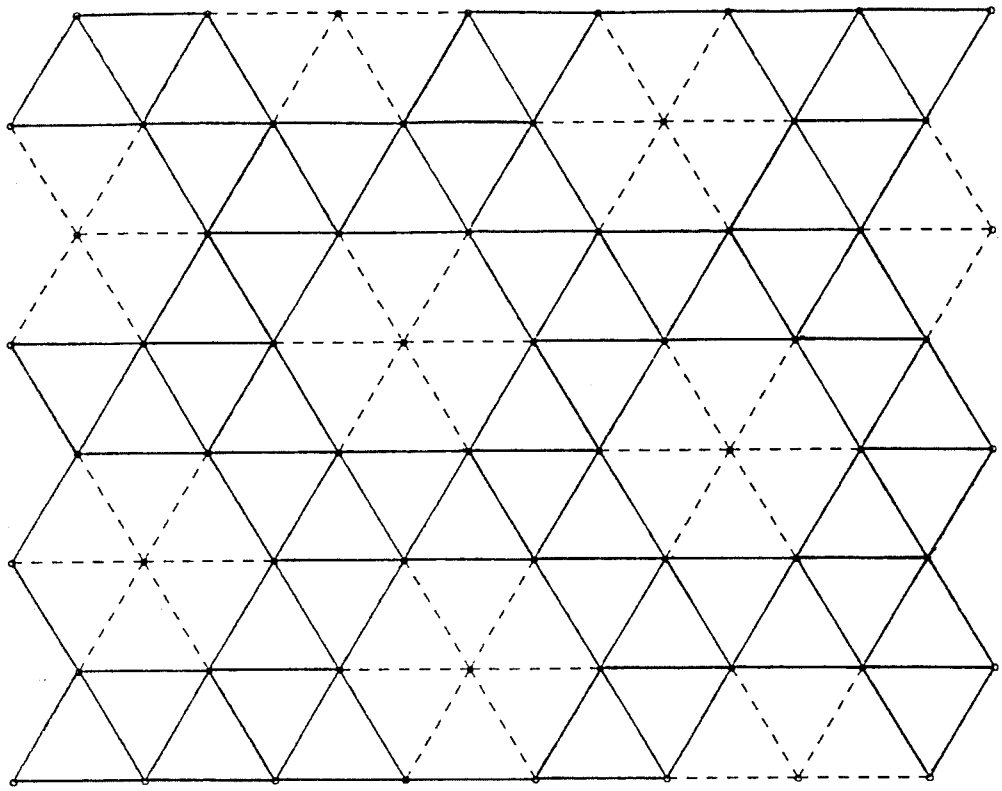


Figure 5.2 Hexagonal Space Emulation of $3^4.6$

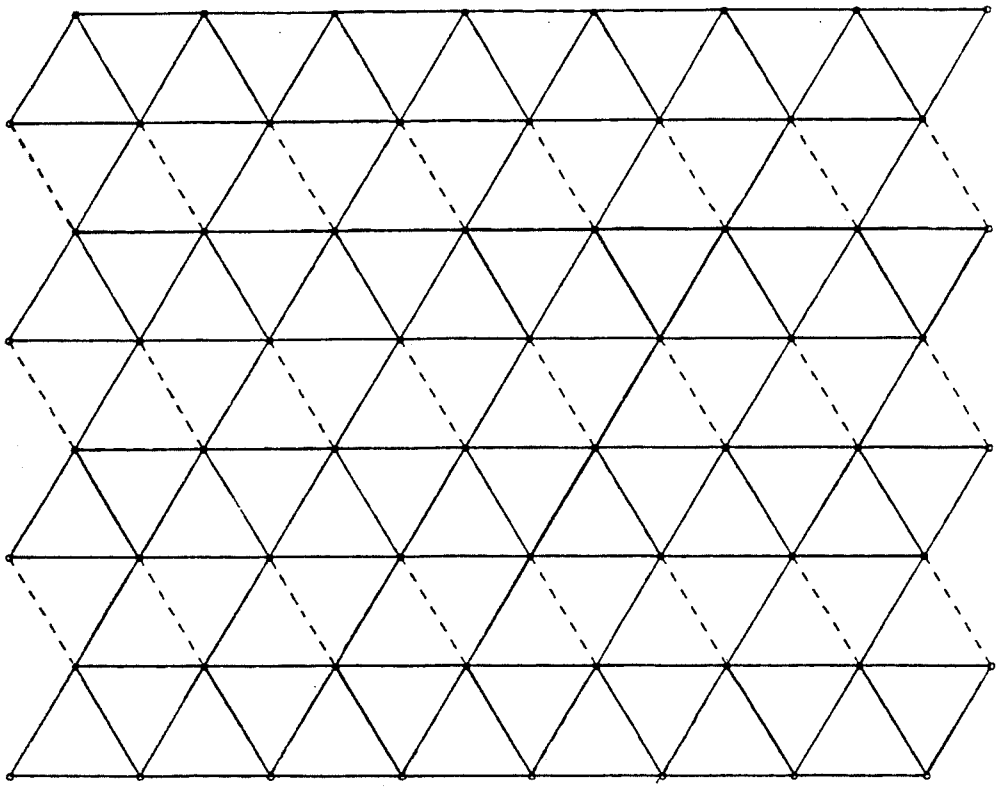


Figure 5.3 Hexagonal Space Emulation of $3^3.4^2$

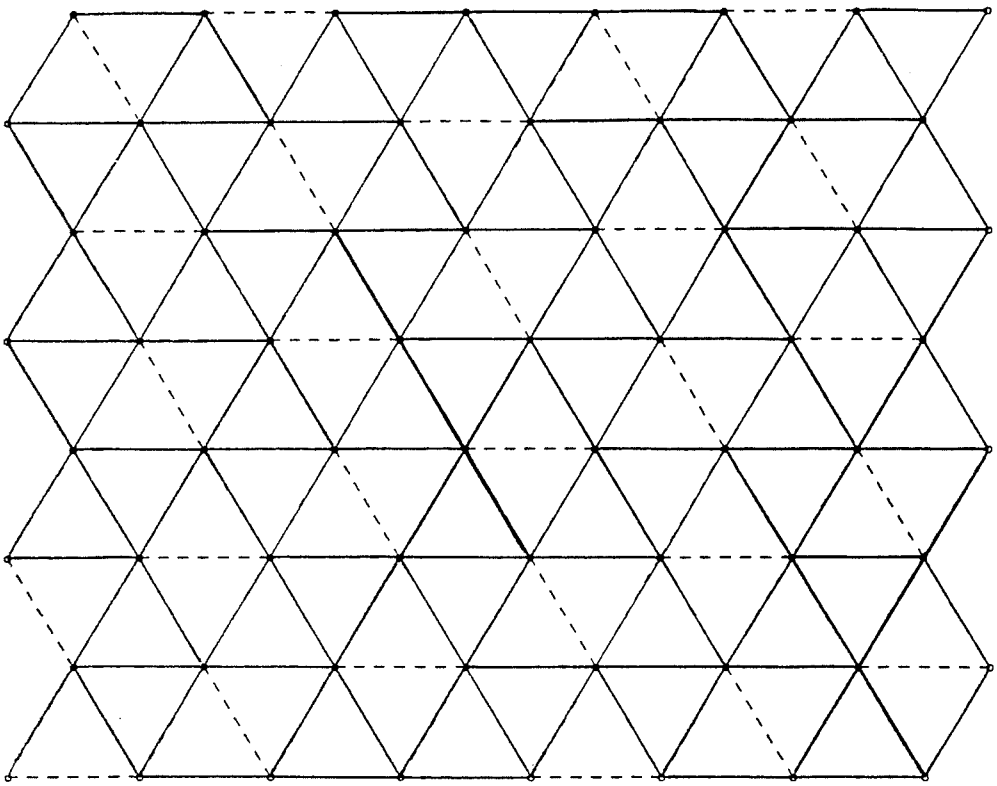


Figure 5.4 Hexagonal Space Emulation of $3^2.4.3.4$

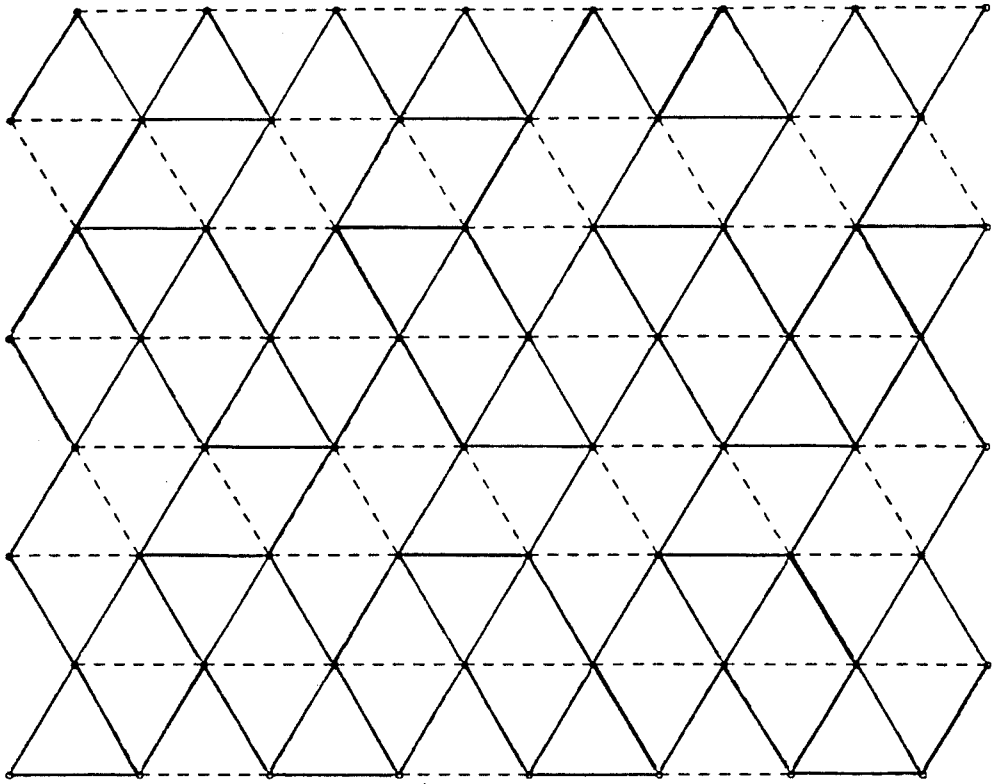


Figure 5.5 Hexagonal Space Emulation of 3.4.6.4

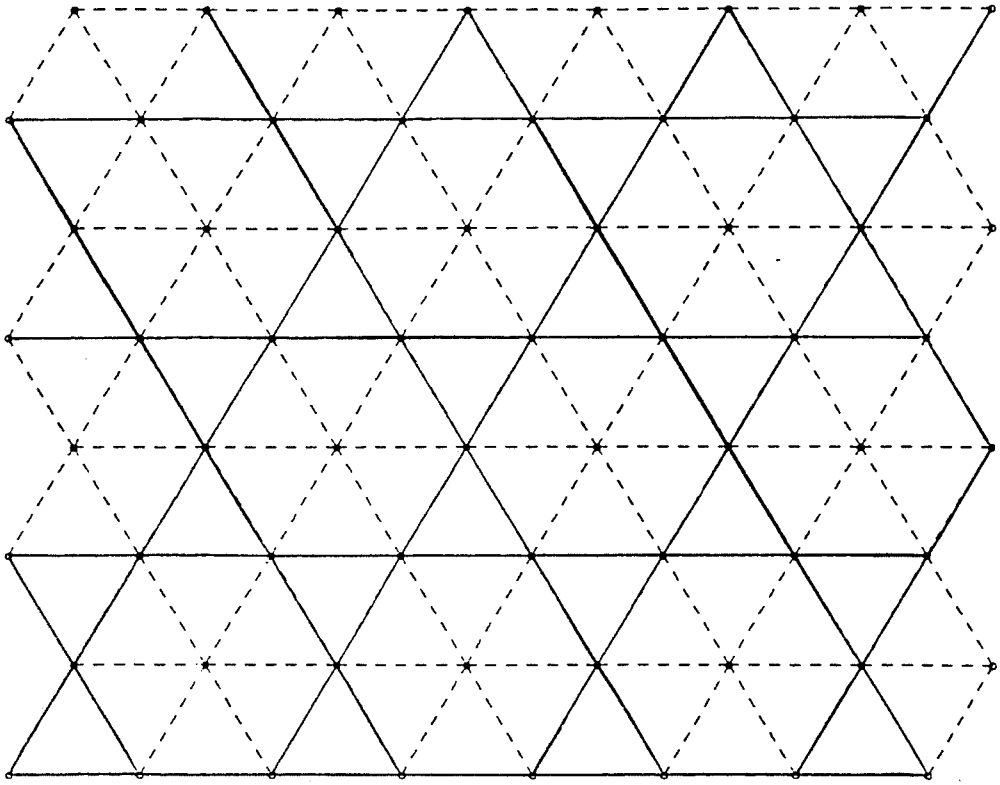


Figure 5.6 Hexagonal Space Emulation of 3.6.3.6

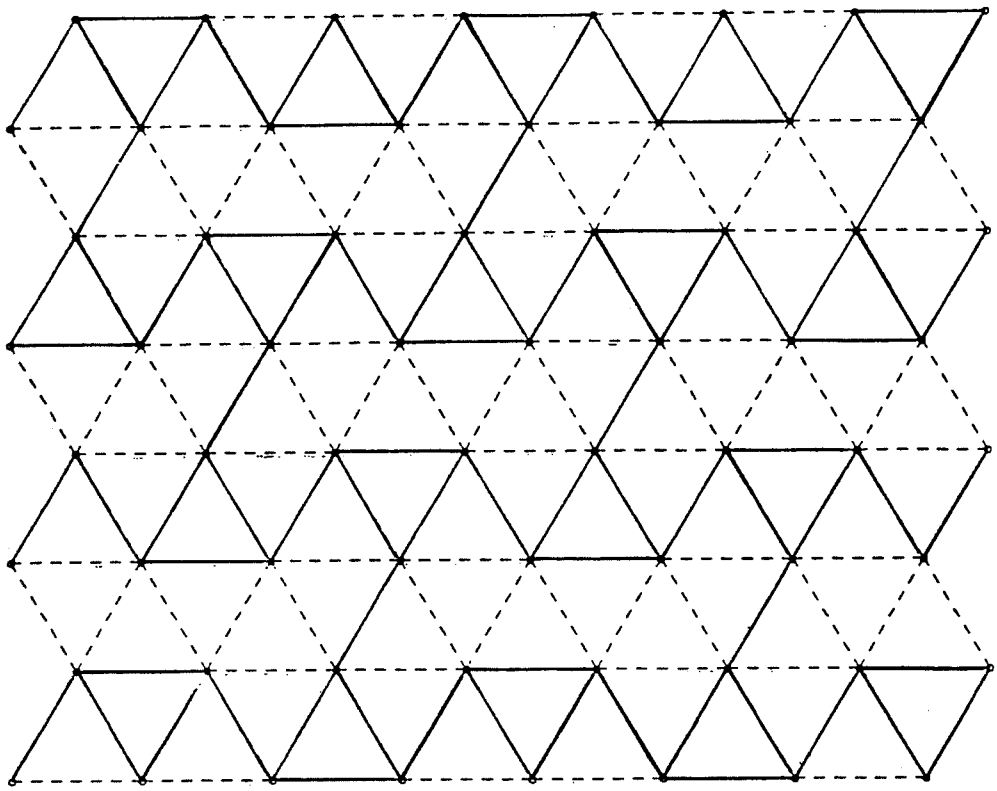


Figure 5.7 Hexagonal Space Emulation of 3.12^2

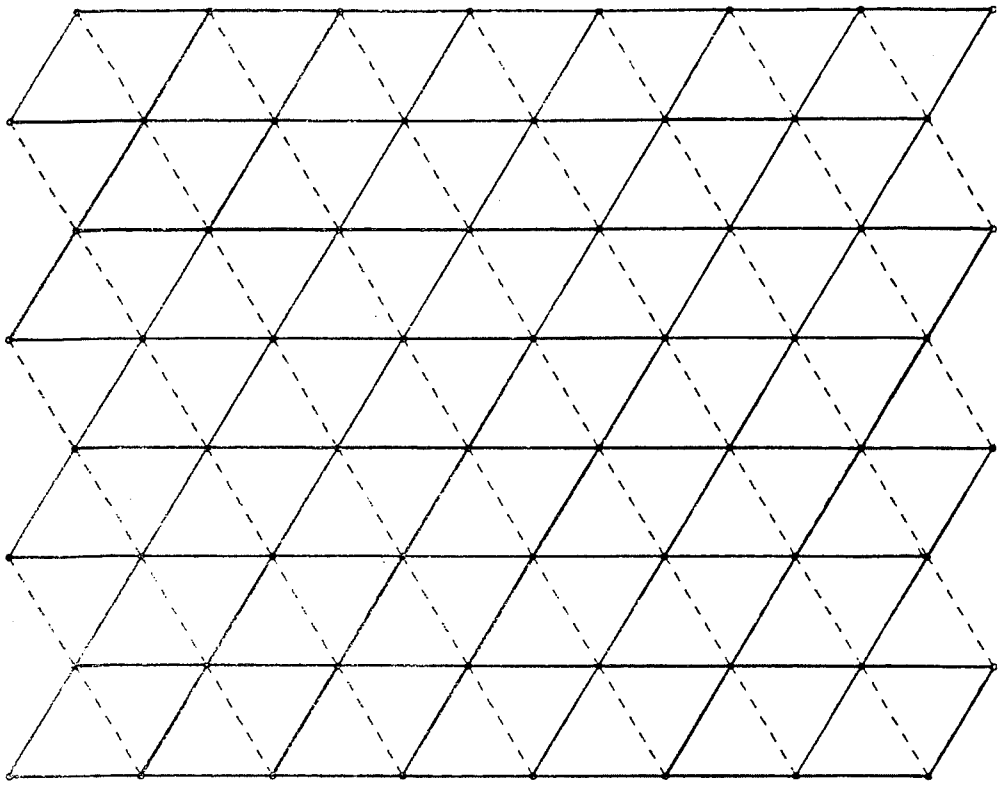


Figure 5.8 Hexagonal Space Emulation of 4^4

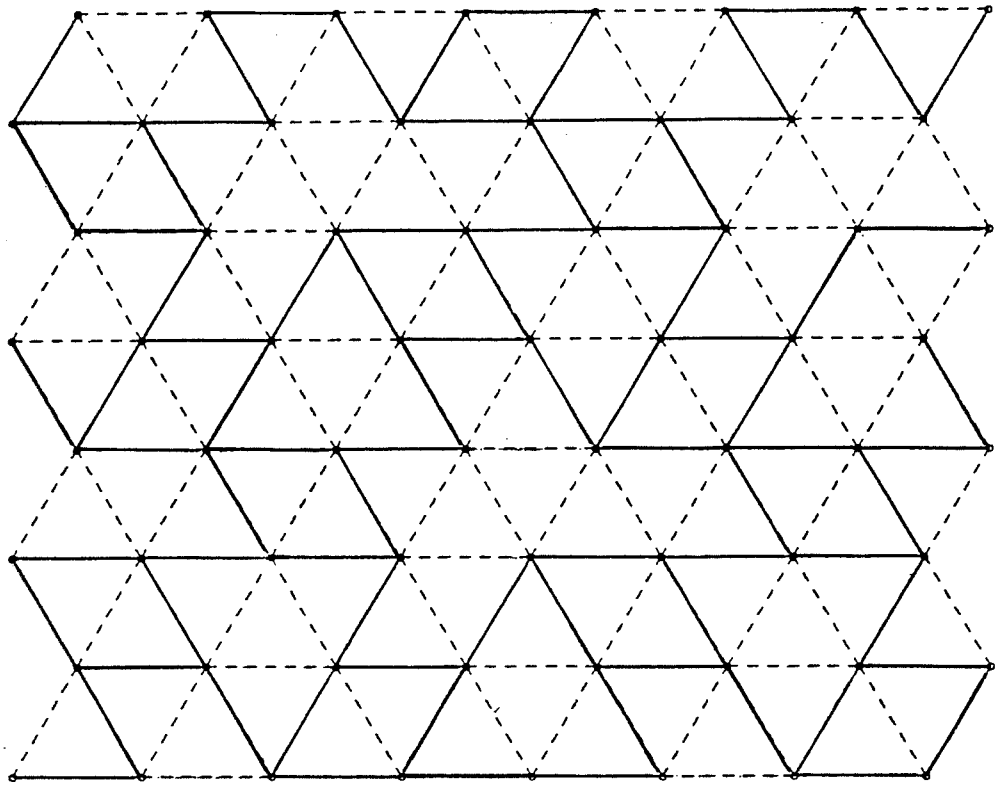


Figure 5.9 Hexagonal Space Emulation of 4.6.12

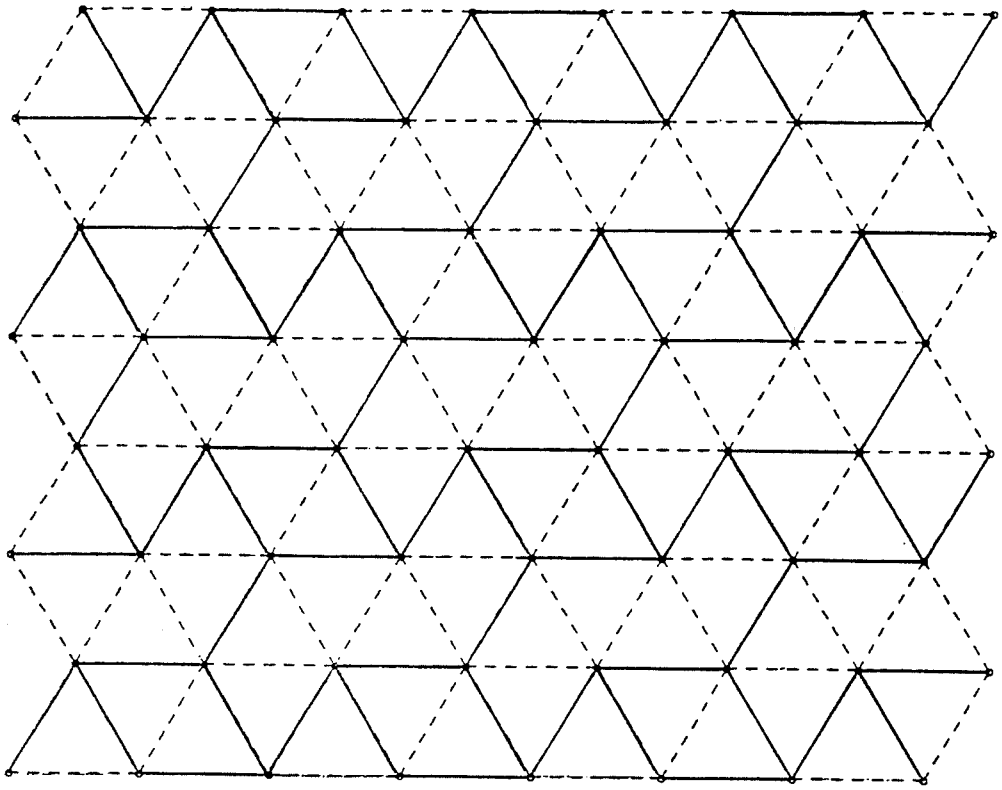


Figure 5.10 Hexagonal Space Emulation of 4.8^2

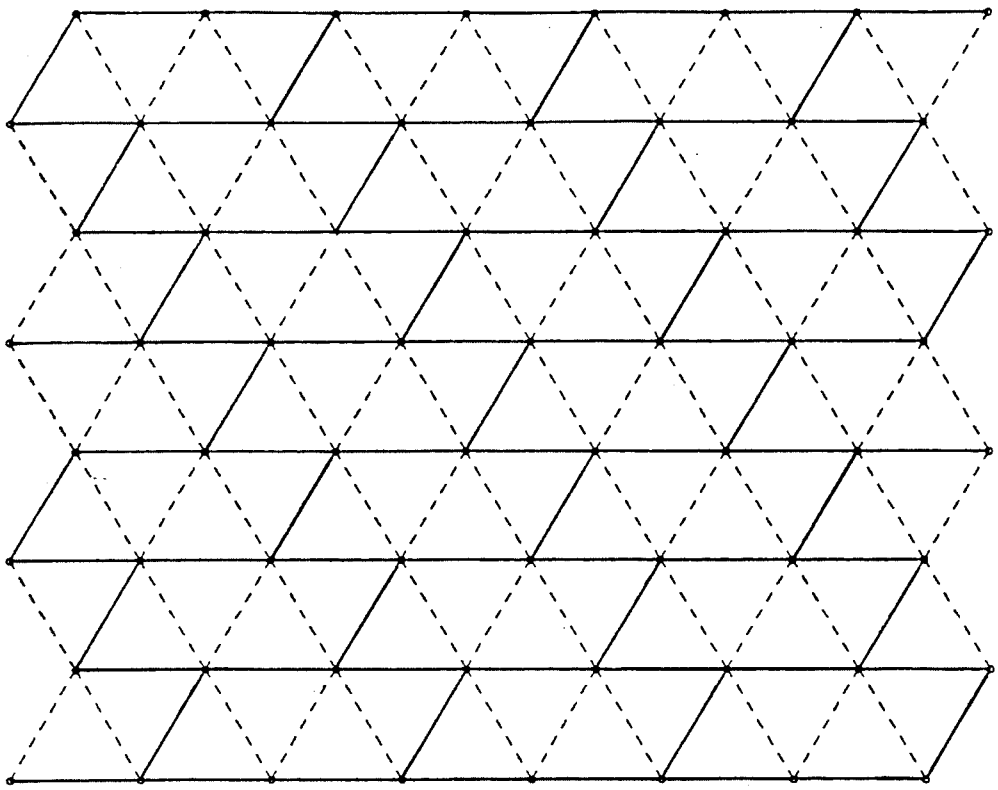


Figure 5.11 Hexagonal Space Emulation of 6^3

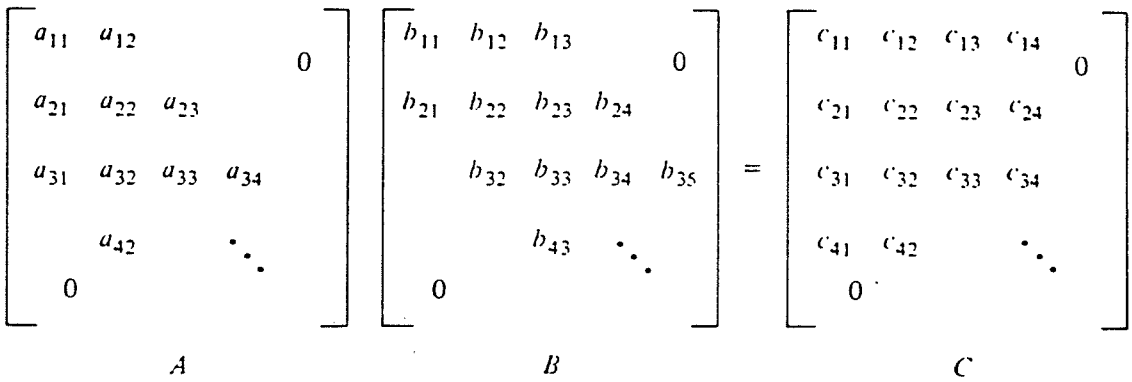


Figure 6. Matrix Multiplication Problem

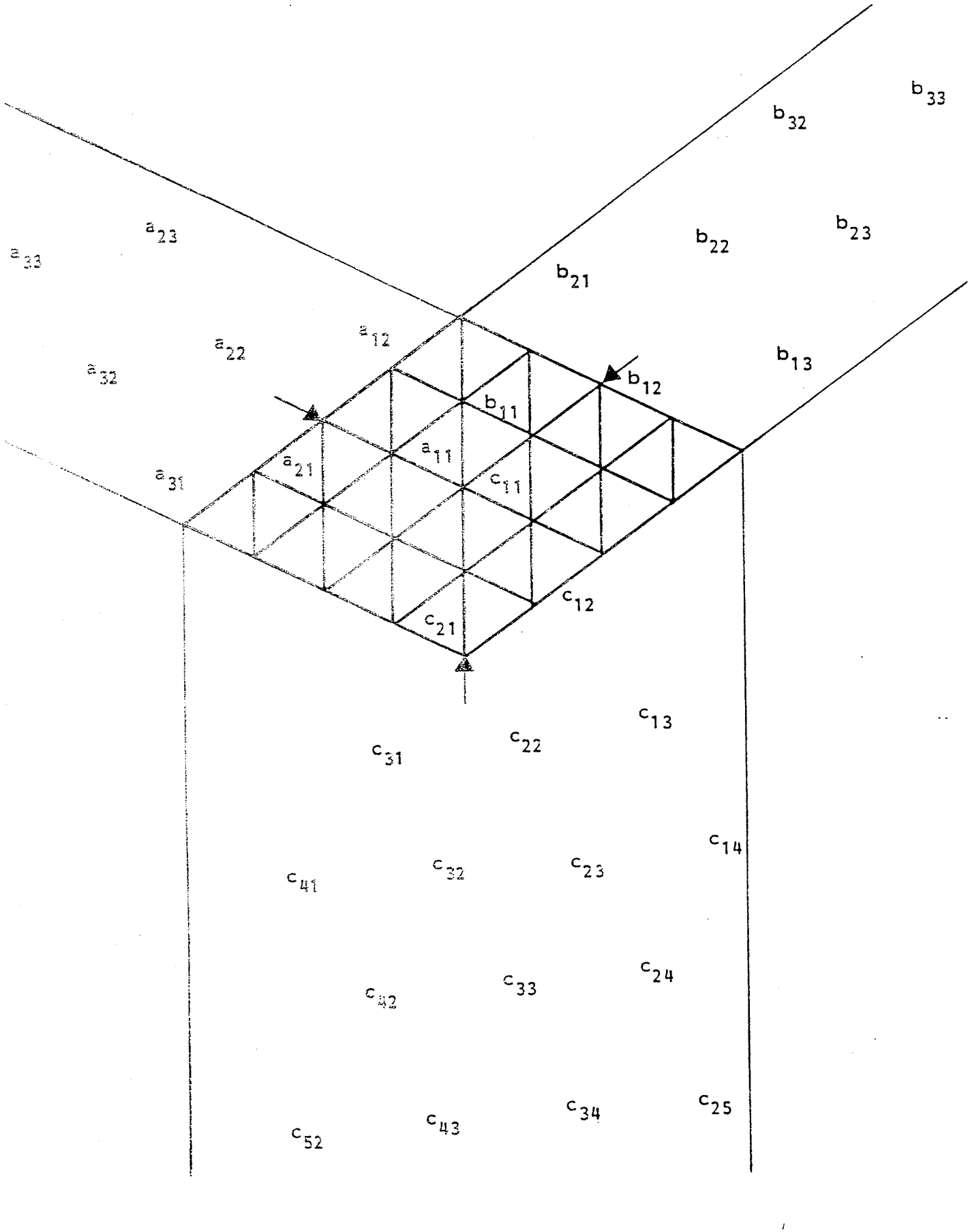


Figure 7. Triangular Processor Array for Matrix Mulitplication

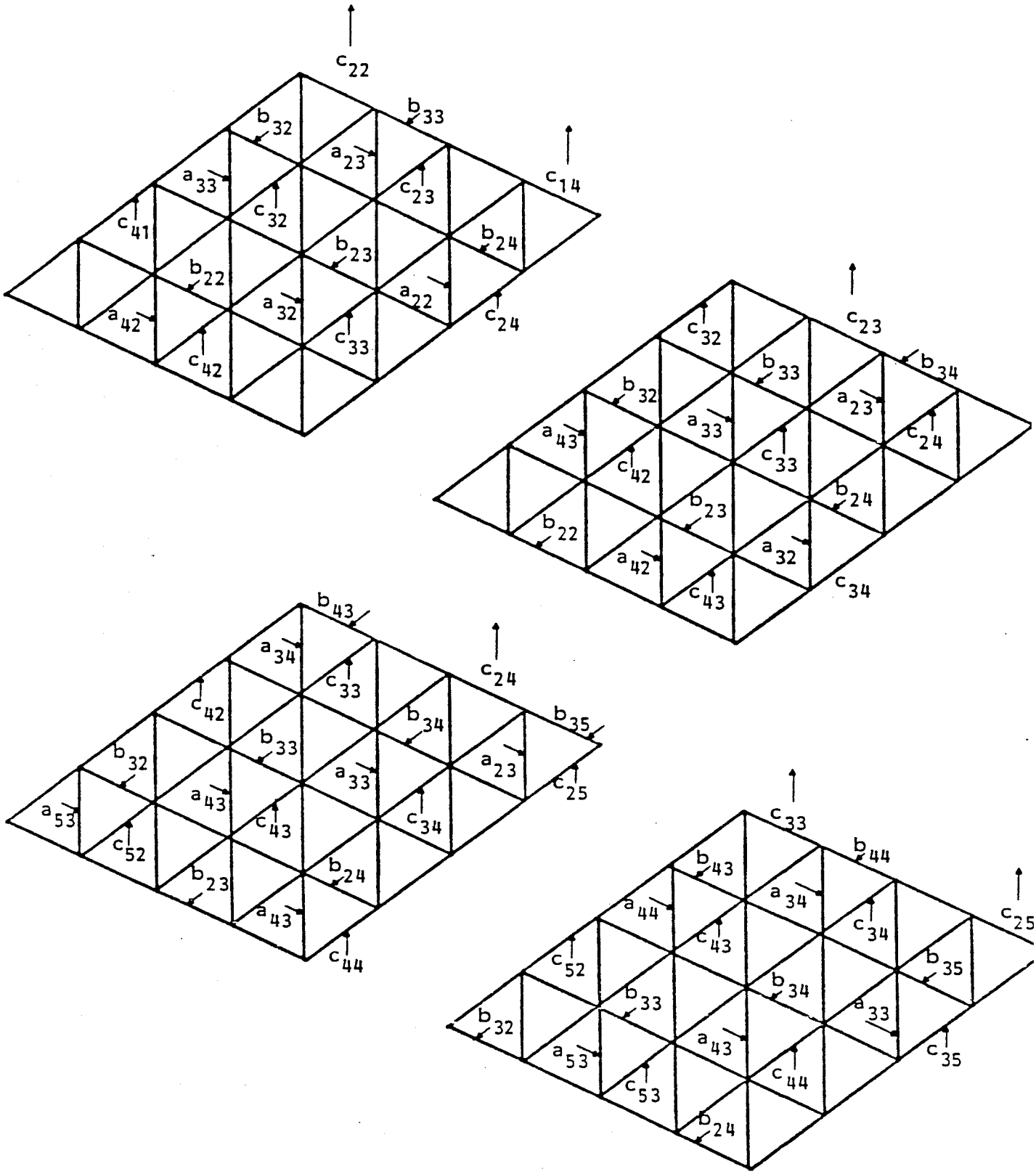


Figure 8. Four Steps in Matrix Multiplication on the Triangular Processor Array

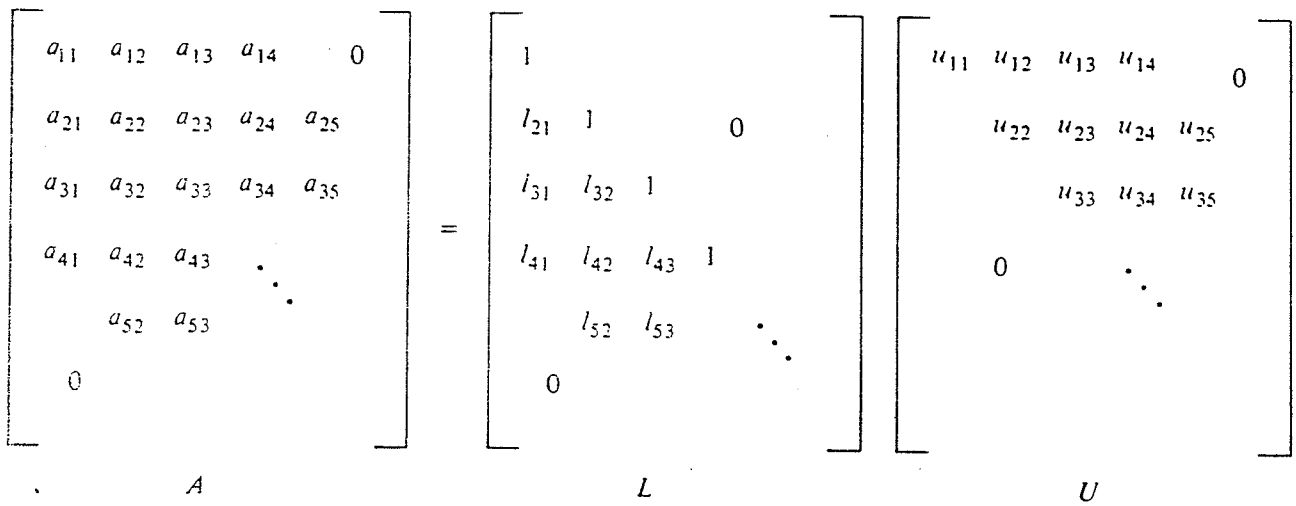


Figure 9. LU Decomposition Problem

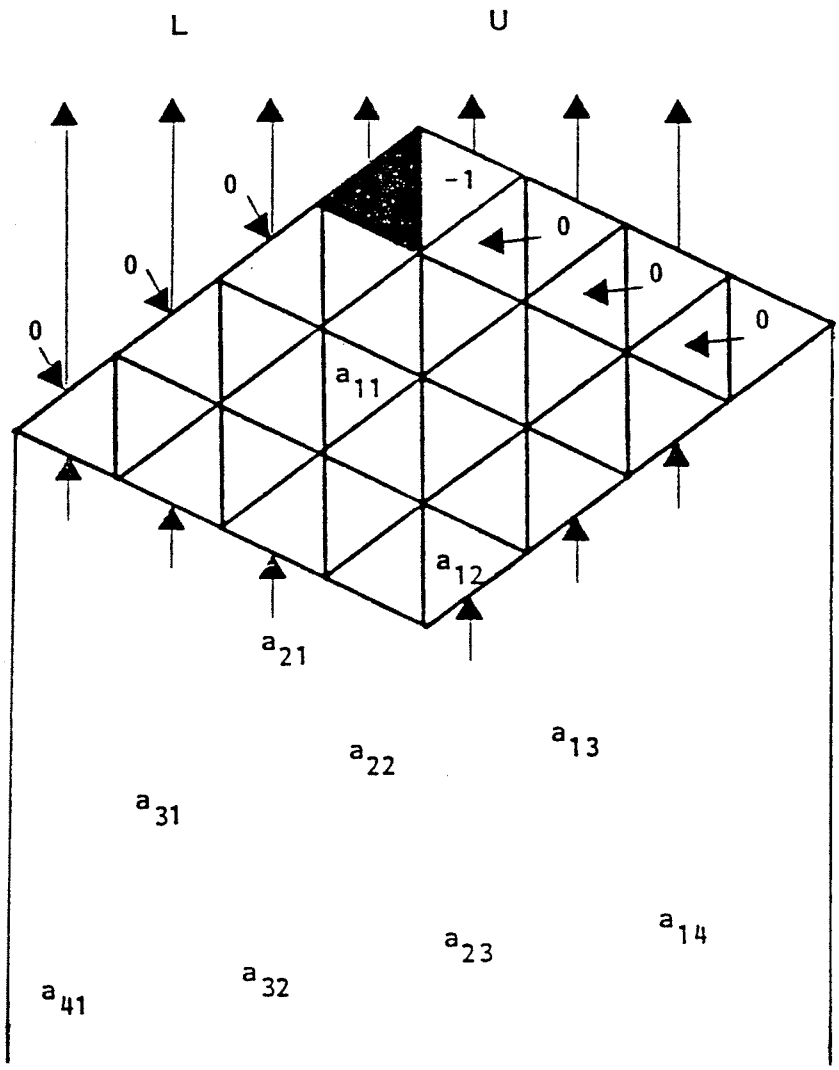


Figure 10. Triangular Processor Array for LU Decomposition

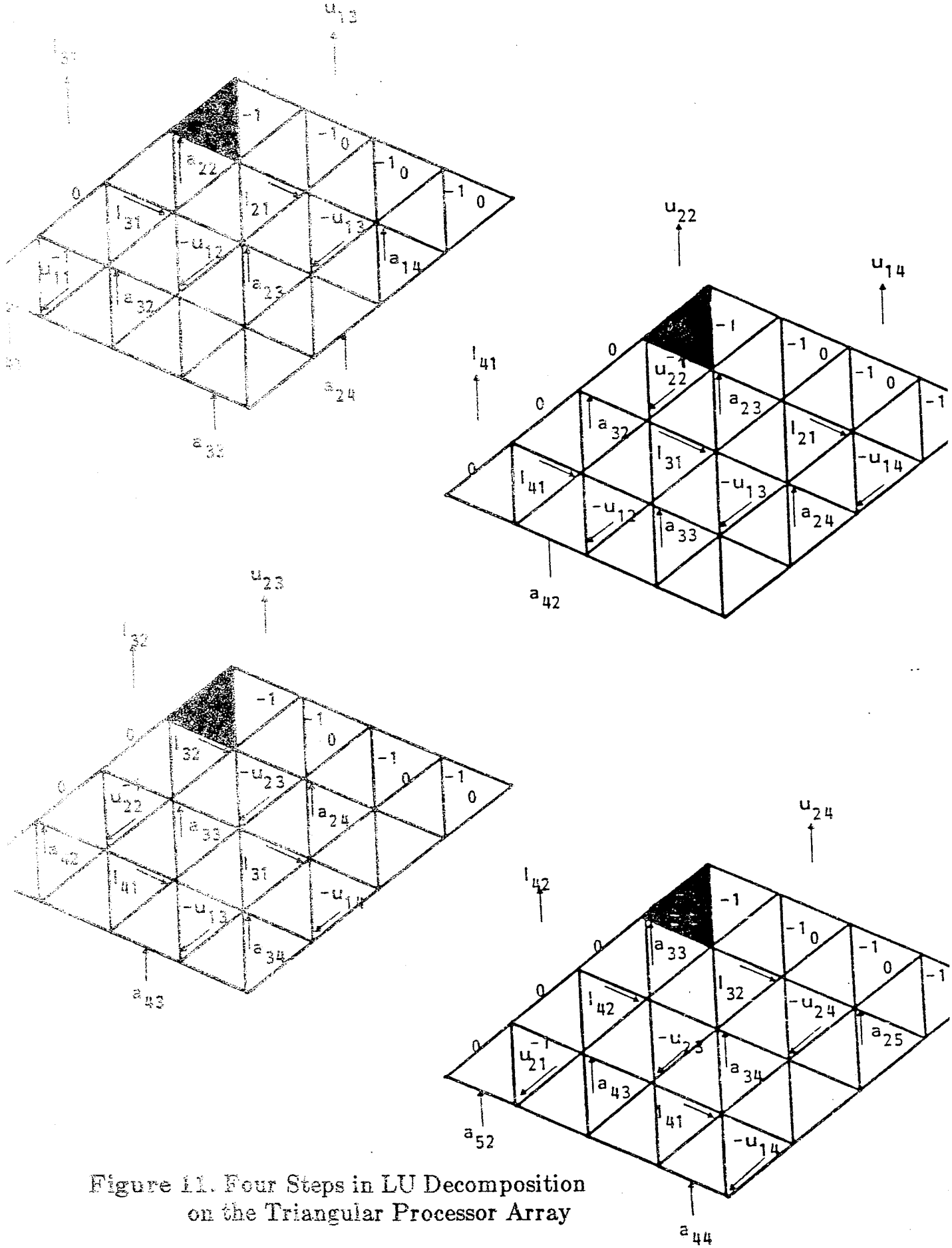


Figure 11. Four Steps in LU Decomposition on the Triangular Processor Array

BIBLIOGRAPHIC DATA SHEET	1. Report No. CSRD-734	2.	3. Recipient's Accession No.
4. Title and Subtitle REGULAR PROCESSOR ARRAYS			5. Report Date January 1988 6.
7. Author(s) Allen D. Malony			8. Performing Organization Rept. No. CSRD-734
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Center for Supercomputing Research and Development Urbana, IL 61801-2932			10. Project/Task/Work Unit No. 11. Contract/Grant No. US NSF MIP-8410110; US DOE-DE-FG02-85ER25001; AFOSR-F49620-86-C-0136; IBM Do1
12. Sponsoring Organization Name and Address National Science Foundation, Washington, DC; U.S. Department of Energy, Washington, DC; U.S. Air Force Office of Scientific Research, Washington, DC; IBM Corporation, Armonk, NY			13. Type of Report & Period Covered Technical Report 14.

15. Supplementary Notes

16. Abstracts

Regular is an often used term to suggest simple and uniform structure of a parallel processor's organization 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. We also show how algorithms developed for one regular processor array might be transferred to another regular array using matrix multiplication and LU decomposition as examples.

17. Key Words and Document Analysis. 17a. Descriptors

architecture
VLSI

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	UNCLASSIFIED	70
	20. Security Class (This Page) UNCLASSIFIED	22. Price