March, 1984 LIDS-P-1368

The Specification and Verification of Systolic Wave Algorithms*

C. J. Kuo†

Bernard C. Levy†

Bruce R. Musicus††

Abstract

This paper proposes a rigorous and systematic approach to analyze the systolic wave algorithms described by data flows and local interactions in a regular multiprocessor array. We formulate basic equations called space-time-data (STD) equations to describe the motion of a single data element in particular, and of a whole wave in general. Using this approach, we prove the correctness of the matrix multiplication algorithm on a hexagonal array. Then, the general specification and verification procedure is presented. Finally, the computational wavefront method is explained from this new point of view, and its limitations are discussed.

---

* This work was supported in part by the Army Research Office under Grant No. DAAG29-84-K-0005 and in part by the Advanced Research Projects Agency monitored by ONR under contract N00014-81-K-0742.

† Laboratory for Information and Decision Systems and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139

†† Research Laboratory of Electronics and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139

1. Introduction

Systolic arrays were first proposed by Kung and Leiserson in 1978 [1]. In their paper, they used a graphical approach to show how their ideas work. The examples they considered included matrix-vector multiplication, matrix multiplication on a hexagonal array, LU decomposition of a matrix, convolution, etc. Although these algorithms are correct, no formal mathematical proof was given. It was only in 1982 that a proof of the correctness of the matrix multiplication algorithm was given by Chen and Mead [2]. They expressed the algorithm as a recursive space-time program, and then applied inductive techniques to verify this program. This approach seems tedious and complicated. In addition, it is difficult to understand its relation with the graphical approach. It is surprising that intuitively obvious results such as systolic algorithms cannot be proved in a straightforward way. One important reason is that we lack suitable tools to describe data flow phenomena in multiprocessor arrays. Several researchers have made efforts in this direction. One popular approach is known as the computational wavefront method [3] [4]. However, to our knowledge, this approach does not provide a verification method either.

One important class of systolic algorithms can be characterized by regular data flow through a multiprocessor array of regular geometry. We call this type of algorithm a *systolic wave algorithm*. In this paper, we propose a simple approach to analyze the matrix multiplication algorithm in particular, and other systolic wave algorithms in general. Specification and verification procedures can be stated in a systematic way. In addition, we formulate the concept of "wavefront" in a mathematically rigorous way. In Section 2, we consider the algorithm of matrix multiplication on a hexagonal array as an illustrative example; then the general specification and verification approach is described in Section 3.

Finally, we use our model to interpret the wavefront concept in Section 4.

2. Matrix multiplication on a hexagonal array - an example

Kung and Leiserson's algorithm is described in Fig 1. The first observation is that a hexagonal array can be described as the superposition of two rectangular grids with the same spacing (h_1, h_2), and with origins at $(0,0)$ and at $(\frac{1}{2}h_1, \frac{1}{2}h_2)$. Therefore, the coordinates of the processors are either $(n_1 h_1, n_2 h_2)$ or $(n_1 h_1 + \frac{1}{2}h_1, n_2 h_2 + \frac{1}{2}h_2)$, where n_1 and n_2 are integers. In fact, h_1 and h_2 are related to the physical space only and are not essential in our discussion, so that we denote the coordinates of the processors as the ordered pair (n_1, n_2). We may redraw the picture in the (n_1, n_2) coordinates as shown in Fig 2.

For convenience, we use $a(i,j)$, $b(i,j)$, and $c(i,j)$ to denote the entries of matrices A, B, and C, and the systolic algorithm shown in Figs. 1 and 2 is used to perform the matrix multiplication A B = C. The origin of the coordinates (n_1, n_2) is chosen to be the point at which $a(1,1)$, $b(1,1)$, and $c(1,1)$ coincide with each other. We use k to represent the global clock, which takes nonnegative integer numbers. Initially, for $k = 0$, we assume that $a(1,1)$, $b(1,1)$, and $c(1,1)$ are located at $(-\frac{1}{2}d, \frac{1}{2}d)$, $(\frac{1}{2}d, \frac{1}{2}d)$, and $(-d, 0)$ respectively; where d is an arbitrary positive integer number. At each time step, all $a(i,j)$'s move to the right one half unit and move down one half unit, so that we may describe this motion with a velocity vector $(\frac{1}{2}, -\frac{1}{2})$. Correspondingly, the velocity vectors of $b(i,j)$ and $c(i,j)$ are $(-\frac{1}{2}, -\frac{1}{2})$ and $(0,1)$. The components of the same matrix retain their relative position when they are shifted because they have the

same velocity. We may call the motion of the entire group a *wave*. For the matrix multiplication case, there are three waves, called wave A, wave B, and wave C. The *position function* of some component, say $a(i,j)$ at time k, expressed as $P_a^k(i,j)$, is defined to be the coordinates of the processor where $a(i,j)$ arrives at time k. Therefore, $P_a^0(i,j)$, $P_b^0(i,j)$, and $P_c^0(i,j)$ represent the initial configurations of waves A, B, and C respectively. In order to specify the motion of waves A, B, and C, we must specify $P_a^k(i,j)$, $P_b^k(i,j)$, and $P_c^k(i,j)$. These can be expressed as follows:

$$P_a^k(i,j) = P_a^0(i,j) + (\frac{1}{2}, -\frac{1}{2})k \quad , \tag{2.1a}$$

$$P_b^k(i,j) = P_b^0(i,j) + (-\frac{1}{2}, -\frac{1}{2})k \quad , \tag{2.1b}$$

$$P_c^k(i,j) = P_c^0(i,j) + (0,1)k \quad . \tag{2.1c}$$

Let n_1, n_2 represent the processor coordinates of data $a(i,j)$ at time k. Usually, the processor array has only finite size. However, for convenience, we may assume it extends in a regular manner to infinity in all directions.

From Fig 3, it is not hard to see that the relation between the data indices (i,j) and the initial processor coordinates $P_a^0(i,j) = (n_1, n_2)$ for wave A is :

$$n_1 + n_2 = -i + j \qquad k = 0 \quad , \tag{2.2a}$$

$$3n_1 - n_2 = -2d - 3(i + j - 2) \quad k = 0 \quad . \tag{2.2b}$$

Solving equation (2.2), we get

$$n_1 = -\frac{(2i + j + d - 3)}{2} \qquad k = 0 \quad , \tag{2.3a}$$

$$n_2 = \frac{(3j + d - 3)}{2} \qquad k = 0 \quad . \tag{2.3b}$$

Therefore, we have

$$P_a^0(i,j) = (-\frac{2i+j+d-3}{2}, \frac{3j+d-3}{2}) \quad , \tag{2.4}$$

and from (2.1a) and (2.4), we get

$$P_a^k(i,j) = (n_1, n_2) = (-\frac{2i+j+d-3-k}{2}, \frac{3j+d-3-k}{2}) \quad . \tag{2.5}$$

From above, we can derive two basic equations relating the processor coordinates n_1, n_2, the time index k, and the data indices i, j of matrix A, which is called, therefore, the *Space-Time-Data* (STD) equations of wave A. They are

$$2n_1 + 2i + j + d - 3 - k = 0 \quad , \tag{2.6a}$$

$$2n_2 - 3j - d + 3 + k = 0 \quad . \tag{2.6b}$$

The motion of wave A is totally specified by its STD equations. The variable d is determined by the initial relative positions of these three waves, so we may regard it as a constant in the following discussion.

There are two equations and five variables to describe the motion of wave A. If i, j are fixed, then n_1 and n_2 can be represented as functions of k, which represent the locus of the motion of the component $a(i,j)$ in a two dimensional plane. Furthermore, if k is chosen to be some specific value, the position of $a(i,j)$ is uniquely determined. This is defined as the position function $P_a^k(i,j)$ before. On the other hand, we may fix n_1 and n_2 and view i, j as functions of k. We define the wave A component of the *memory function* of the processor (n_1, n_2) at time k as

$$M_a^k(n_1, n_2) = (i, j) \quad , \tag{2.7}$$

i.e., (i, j) is the index of the element of wave A contained in processor (n_1, n_2) at time k. The memory function tells us what component is stored within a specific processor at a specified time. Rearranging equation (2.6), we obtain

$$M_a^k(n_1,n_2)=(-\frac{3n_1+n_2+d-3-k}{3},\frac{2n_2-d+3+k}{3}) \quad . \tag{2.8}$$

For fixed n_1, n_2 and k, the computed data indices i and j of wave A are not necessarily integers. If they are not integers, where $a(i,j)$ cannot be defined, we may assign $M_a^k(n_1,n_2)$ the value "nil". In fact, it is not hard to see that each processor contains "nil" values for two succesive time steps in k, then gets a new value of $a(i,j)$ on the third time step.

Wave B and C can be treated similarly. Looking at Fig 4 and using the same approach, we get the STD equations for wave B

$$2n_1-i-2j-d+3+k=0 \quad , \tag{2.9a}$$

$$2n_2-3i-d+3+k=0 \quad . \tag{2.9b}$$

The corresponding position function and memory function are

$$P_b^k(i,j)=(\frac{i+2j+d-3-k}{2},\frac{3i+d-3-k}{2}) \quad . \tag{2.10}$$

and

$$M_b^k(n_1,n_2)=(\frac{2n_2-d+3+k}{3},\frac{3n_1-n_2-d+3+k}{3}) \quad . \tag{2.11}$$

Similarly, from Fig 5 and using the previous procedure, we find the STD equations of wave C

$$2n_1+i-j=0 \quad , \tag{2.12a}$$

$$2n_2+3i+3j-6+2d-2k=0 \quad . \tag{2.12b}$$

The position function and memory function are

$$P_c^k(i,j)=(\frac{-i+j}{2},-\frac{3i+3j-6+2d-2k}{2}) \tag{2.13}$$

and

$$M_c^k(n_1,n_2)=(-\frac{3n_1+n_2-k+d-3}{3},\frac{3n_1-n_2+k-d+3}{3}) \quad . \tag{2.14}$$

respectively.

In order to prove the components relationship for the matrix multiplication, i.e.

$$c(i,j)=\sum_{p=1}^{N} a(i,p)\,b(p,j) \quad , \tag{2.15}$$

All we have to do here is to guarantee that the suitable data will come to the same processor at the right time. So let us keep track of the locus of motion of some arbitrary component of the matrix C, say $c(i,j)$, and see what components, which belong to other waves, are within the same processor at the same time. This statement can be described precisely by two compound functions: $M_a^k(P_c^k(i,j))$ and $M_b^k(P_c^k(i,j))$. Using (2.8), (2.11), and (2.13), we can evaluate the values of these two compound functions and get

$$M_a^k(P_c^k(i,j))=(i,p(k)) \quad , \tag{2.16a}$$

$$M_b^k(P_c^k(i,j))=(p(k),j) \quad . \tag{2.16b}$$

where

$$p(k)=-i-j+3+k-d \quad . \tag{2.17}$$

If $k \le i+j+d-3$, then $p(k) \le 0$. However, $a(i,j)$ and $b(i,j)$ are defined only when i and j are positive integers. That means the component $c(i,j)$ has not yet encountered the other two waves. If $k \ge i+j+d-2$, then $p(k)$ is monotonically increasing from 1 with the time clock until the upper bound, which is determined by the size of the input matrix A (or B), say N here. Let us take a simple case, say, $i=1$ and $j=1$. When $k=d$, these three components, i.e., $a(1,1)$, $b(1,1)$, and $c(1,1)$ meet one another at the processor $(0,0)$, which equals $P_c^d(1,1)$. This satisfies our original assumption. When $k=d+1$, $a(1,2)$, $b(2,1)$ and $c(1,1)$ coincide at the processor $(0,1)$, which is $P_c^{d+1}(1,1)$, and so on. Therefore, we prove the correctness of the systolic matrix

multiplication algorithm.

3. The Specification and Verification of General Systolic Wave Algorithms

In general, to specify a systolic wave algorithm precisely requires knowledge of processor coordinates, the action of each processor, the data wave values and their physical locations as a function of time. First, assume that there is a fixed network of processors at "coordinates" $\underline{n}$. These coordinates may be the physical location of each processor in space, or they may just be a convenient index. Let us also assume that the data flowing through the network may be partitioned into N "waves" of data $a_1(\underline{i}), \ldots, a_N(\underline{i})$, where each $a_j(\underline{i})$ is the j^{th} set of data indexed by coordinate $\underline{i}$. As the computation proceeds, the value of each element in the wave will change, so let us call $a_j^k(\underline{i})$ the value of the $\underline{i}^{th}$ element of wave j at time k.

As the data flows through the network of processors, differing elements of each wave will be located in different processors at different times. We assume that this data flow is independent of the values calculated by the processors. Let $\underline{i} = M_j^k(\underline{n})$ be the element of the j^{th} wave which is located in processor $\underline{n}$ at time k. We assume that at most one element from each wave may be located in a single processor at any one time. If no element of the j^{th} wave is at $\underline{n}$ at time k, we will say that the value of the "memory function" $M_j^k(\underline{n})$ is *nil*, and the values $a_j(nil)$ and $M_j^k(nil)$ we define to be *nil*. It is convenient to define the "position function" $P_j^k(\underline{i})$ as the inverse of $M_j^k(\underline{n})$; i.e. $\underline{n} = P_j^k(\underline{i})$ is the processor containing element $\underline{i}$ of the j^{th} wave at time k. This function will have value *nil* if the element $\underline{i}$ is not assigned to any processor at time k. (The equations $\underline{n} = P_j^k(\underline{i})$ form the STD equations of section 2).

Finally, suppose that each processor at time k takes the N values $a_1, \ldots, a_N$ located in that processor from each data wave, and computes new values for the corresponding elements of each data wave. Let $a_j^{k+1}(\underline{i}) = G_{j\,\underline{n}}^{k}(a_1, \ldots, a_N)$ be the new value of the $\underline{i}^{th}$ element of the j^{th} wave computed in processor $\underline{n}$ at time k given data values $a_1, \ldots, a_N$. If no element of wave j is located at processor $\underline{n}$ at time k, i.e. if $M_j^k(\underline{n}) = nil$, then we assume that $G_{j\,\underline{n}}^{k}(a_1, \ldots, a_N)$ is also nil. If at time k an element of a wave $a_j^k(\underline{i})$ is not located in any processor, $P_j^k(\underline{i}) = nil$, then we treat the element as if it were in a temporary storage cell during this clock period, and set $a_j^{k+1}(\underline{i}) = a_j^k(\underline{i})$.

Now to verify a systolic algorithm, we start with some assumed initial values for all the waves of data, $a_1^0(\underline{i}), \ldots, a_N^0(\underline{i})$, compute the values for all data waves at all times k as the data flows through the processors, and verify that the final values $a_1^{\infty}(\underline{i}), \ldots, a_N^{\infty}(\underline{i})$ are the desired results. This verification needs to iterate over all processor nodes $\underline{n}$, all waves j, all wave elements $\underline{i}$, and all time k, and can be organized in a variety of ways. The simplest approach is to track the activity of every processor as a function of time:

For all time $k = 0, 1, 2, \ldots$

For all processor nodes $\underline{n}$:

$$\underline{i}_j = M_j^k(\underline{n}) \qquad \text{for } j = 1, \ldots, N$$

$$a_j^{k+1}(\underline{i}_j) = G_{j\,\underline{n}}^{k}\left(a_1^k(\underline{i}_1), \ldots, a_N^k(\underline{i}_N)\right) \qquad \text{for } j = 1, \ldots, N$$

This procedure simply duplicates the calculation of every processor node over time, and thus precisely simulates the network. An alternative verification method which is more similar to that used in the matrix multiplication example above, is to track each wave of data through the array:

For all time $k = 0,1,2,\ldots$

For each wave $j = 1, \ldots, N$

For all elements $\underline{i}$:

$$\underline{n} = P_j^k(\underline{i})$$

$$\underline{i}_m = M_m^k(\underline{n}) \qquad \text{for } m = 1, \ldots, N$$

$$a_j^{k+1}(\underline{i}_j) = G_{j\,\underline{n}}^k\left(a_1^k(\underline{i}_1), \ldots, a_N^k(\underline{i}_N)\right)$$

This approach may be more efficient in cases such as matrix multiplication where elements are only sparsely distributed through the processor array, so that $M_j^k(\underline{n})$ has many *nil* values.

Note that this approach also covers networks in which each processor is a finite state machine with state $\underline{x}^k(\underline{n})$ at time k. Simply add a new "data wave" $a_{N+1}^k(\underline{i})$ whose value is the state and whose position does not change with time.

4. Computational Wavefronts

In special cases, the waves of data flowing through the processor array can be treated as if they were waves propagating through a homogeneous medium. This will be the case if each wave $a_j^k(\underline{i})$ retains its "geometric shape" as it flows through the network. More rigorously, we will say that the processor array has "wavefront" behavior if the position and memory functions $P_j^k(\underline{i})$ and $M_j^k(\underline{n})$ are linear functions of the time and wave indices:

$$\begin{aligned} P_j^k(\underline{i}) &= \Phi_j\,\underline{i} + k\,\underline{\gamma}_j \qquad \textit{whenever} \quad P_j^k(\underline{i}) \neq \textit{nil} \\ M_j^k(\underline{n}) &= \Psi_j\,\underline{n} + k\,\underline{\xi}_j \qquad \textit{whenever} \quad M_j^k(\underline{n}) \neq \textit{nil} \end{aligned} \tag{4.1}$$

where Φ_j and Ψ_j are matrices, and where $\underline{\gamma}_j$ can be interpreted as the "physical wavefront velocity vector", and $\underline{\xi}_j$ can be interpreted as the "data wavefront velocity vector". In the

matrix multiplication case, for example, the physical velocity of wave A was $\underline{\gamma}_a = (\frac{1}{2}, -\frac{1}{2})$, and the data velocity vector of wave A was $\underline{\xi}_a = (\frac{1}{3}, \frac{1}{3})$.

Note that wave-like behavior can only occur on a processor network with the appropriate "shift-invariant" geometric regularity. Namely, there must be processors $\underline{n} = \Phi_j \underline{i} + k \underline{\gamma}_j$ for every $\underline{i}, j, k$ for which $P_j^k(\underline{i}) \neq nil$.

Although this geometric regularity does not necessarily imply that all processors compute the same function $G_{j\underline{n}}^k(a_1, \ldots, a_N)$, the wavefront behavior is particularly useful if the processors do all compute the same function at each time k, so that $G_{j\underline{n}}^k$ is not a function of $\underline{n}$. Assume that $P_j^k(\underline{i})$ has no *nil* values, and is linear in k and i. In this case, it is easy to show that the mapping from the waves $a_j^k(\underline{i})$ to $a_j^{k+1}(\underline{i})$ is spatially invariant, and thus that the final data wave values $a_j^\infty(\underline{i})$ are a spatially invariant mapping from the initial values $a_j^0(\underline{i})$. This is the case, for example, in the matrix multiplication example of section 2. For such systems, our second verification procedure is particularly easy since we need only track a single element $\underline{i}$ of each wave through the processor network as time evolves, and verify that $a_j^\infty(\underline{i})$ obeys the correct mapping from the initial data wave values. Spatial invariance then guarantees that all data values will be correct.

By analogy with wave propagation through a medium, we might define a "computational wavefront" for each wave of data as the set of processors $\underline{n}$ located in a hyperplane "orthogonal" to the velocity vector, $\{\underline{n} \mid \underline{\gamma}_j^T \underline{n} = constant\}$. This implies a corresponding wavefront of data elements $\underline{i}$ which flow through these processors at time k, whose coordinates can be derived from the position function:

$$constant = \underline{\gamma}_m^T \underline{n} = \left(\underline{\gamma}_j^T \Phi_j\right) \underline{i} + k \left\| \underline{\gamma}_j \right\|^2 \qquad (4.2)$$

Since the wave moves as a unit through the processor array, these wavefronts can never cross.

While this "wavefront" idea is rather elegant, unfortunately, it has some technical problems. Most importantly, if data elements are kept in temporary storage and are not used on every cycle, then the position and memory functions $P_j^k(\underline{n})$ and $M_a^k(\underline{i})$ have numerous *nil* values. As a result, the data flow cannot be spatially invariant, and a simple "velocity" vector will not properly describe the behavior of this data flow. Another problem is that defining the "angle" between processors or data paths is a rather arbitrary concept. Thus the choice of which processors belong on a "wavefront" is rather arbitrary. In short, for systolic networks which are not spatially invariant, the existence of computational wavefronts may be intuitively pleasing and may simplify the synthesis of systolic algorithms, but it does not significantly simplify our procedure for proving correctness of a systolic algorithm.

5. Conclusion

In this paper we have used a new approach to describe data flow phenomena in systolic arrays. It is easy, rigorous and systematic. We have also clarified the concept of a "computational wavefront". Kung mentioned two open problems in [5]: one is the specification and verification of systolic algorithms, and the other is automatic algorithm design. We have solved the first problem for an important class of systolic algorithms. However, if the geometry is not very regular or the flow pattern is extremely complicated, it will not be easy to analyze the systolic algorithm. In this case, it may be preferable to

start with a simple implementation of the desired algorithm which can be easily proved correct, and then systematically transform it into a parallel and pipelined form. Thus, we come to the second problem mentioned in [5] - automatic systolic algorithm design. This is still an open area of research.

6. Acknowledgement

The first author wishes to thank Wei K. Tsai for helpful discussions.

References

[1] H. T. Kung and C. E. Leiserson, "Systolic Array (for VLSI)," in *Sparse Matrix Proc. 1978*, SIAM, 1979, pp. 256-282.

[2] M. C. Chen and C. A. Mead, "Concurrent Algorithms as Space-Time Recursion Equations," in *USC Workshop on VLSI and Modern Signal Processing*, 1982.

[3] S. Y. Kung, K. S. Arun, R. J. Gal-ezer, and D. V. Bhaskar Rao, "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computer*, vol. 31, no. 11, pp. 1054-1066, Nov. 1982.

[4] U. Weiser and A. Davis, "A Wavefront Notational Tool for VLSI Array Design," in *VLSI Systems and Computations*, Rockville, MD: Computer Science Press, 1981, pp. 226-234.

[5] H. T. Kung, "Why Systolic Architectures?," *Computer*, vol. 15, no. 1, pp. 37-46, Jan. 1982.

Figure Captions

Fig 1. Systolic matrix multiplication on a hexagonal array

Fig 2. Systolic matrix multiplication rearranged in (n_1, n_2) coodinates

Fig 3. Initial configuration of wave A

Fig 4. Initial configuration of wave B

Fig 5. Initial configuration of wave C

$$\begin{bmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & & \ddots & \\ 0 & & & & \end{bmatrix} \quad \begin{bmatrix} b_{11} & b_{12} & b_{13} & & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & \\ & b_{32} & b_{33} & b_{34} & b_{35} \\ & & b_{43} & \ddots & \\ 0 & & & & \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & & \ddots & \\ 0 & & & & \end{bmatrix}$$

A B C

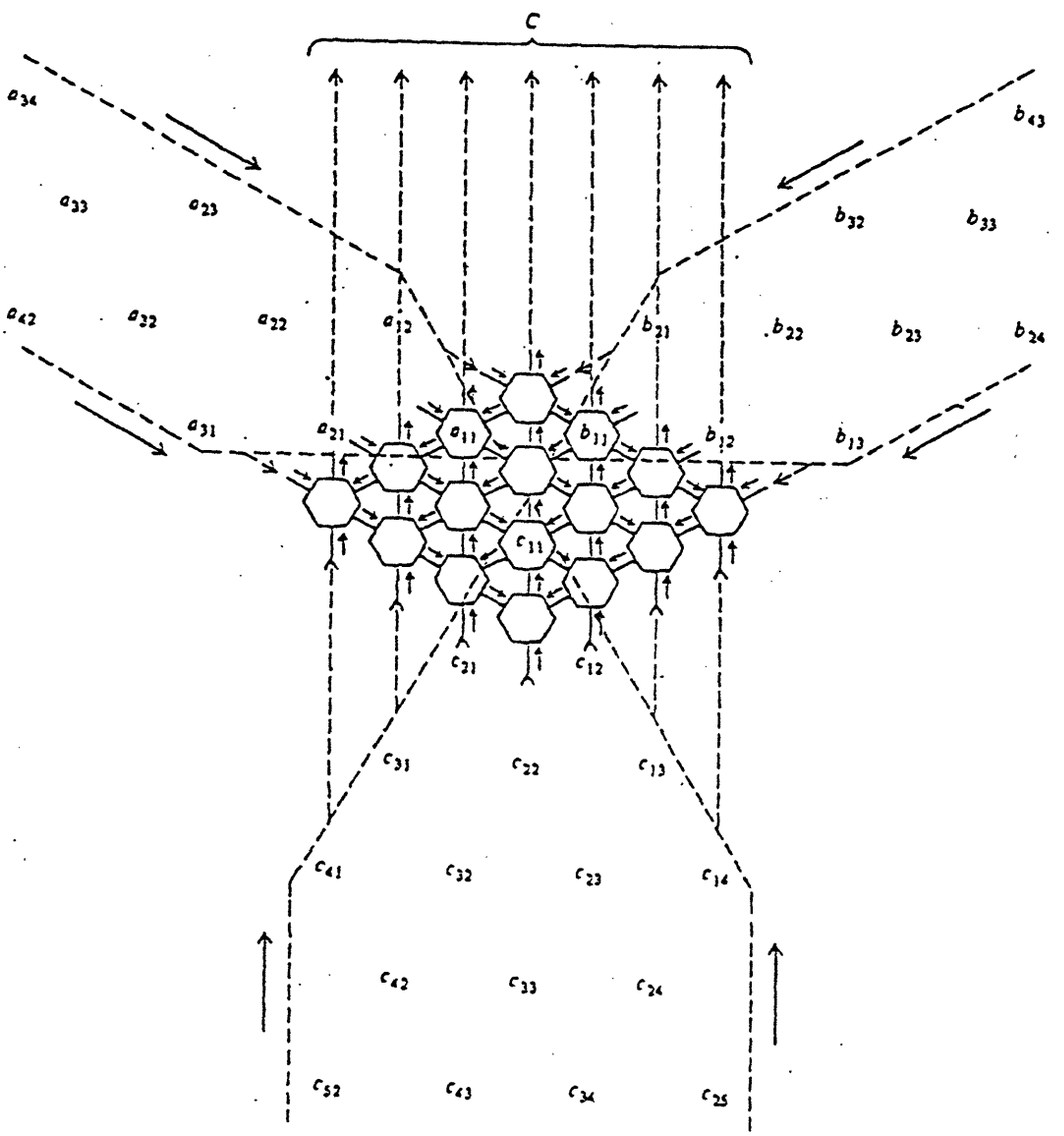

Fig. 1

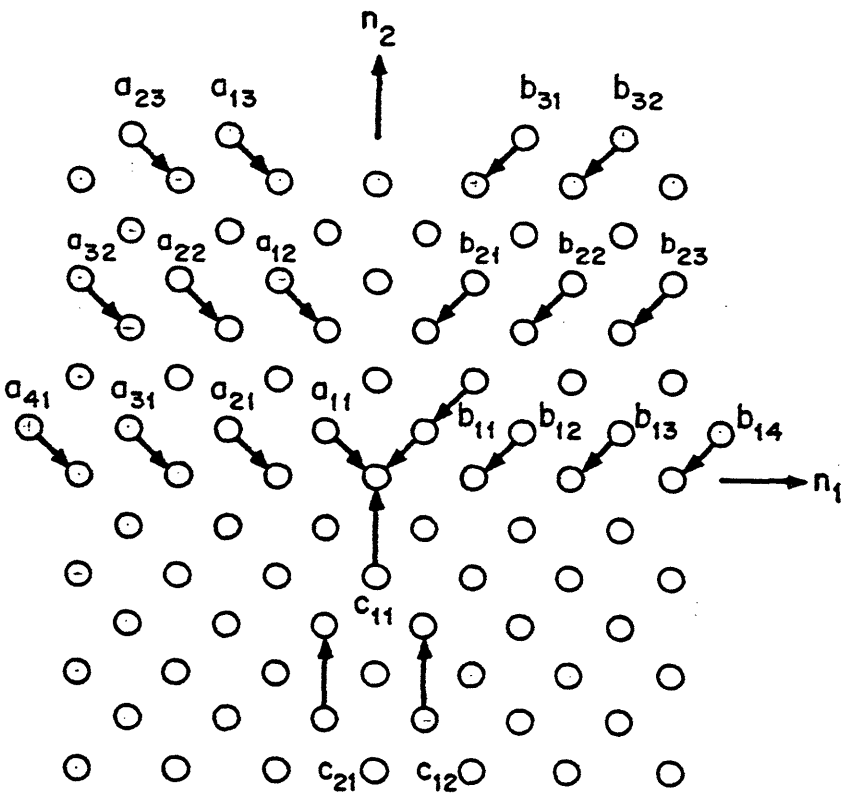
n2
a23
a13
b31
b32
a32
a22
a12
b21
b22
b23
a41
a31
a21
a11
b11
b12
b13
b14
n1
c11
c21
c12

Fig. 2

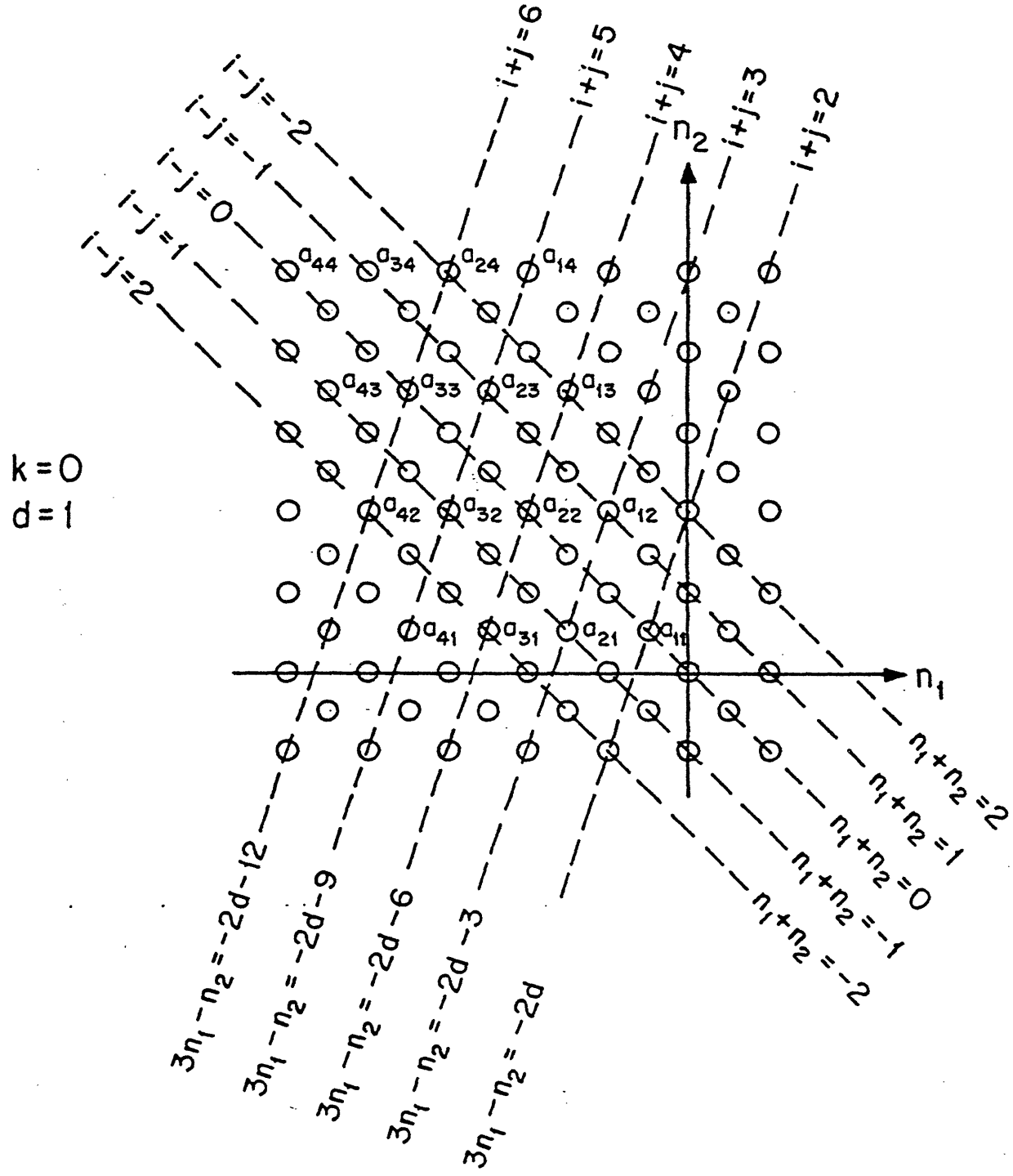
k = 0
d = 1
i-j=-2
i-j=-1
i-j=0
i-j=1
i-j=2
i+j=6
i+j=5
i+j=4
i+j=3
i+j=2
n2
n1
a44 a34 a24 a14
a43 a33 a23 a13
a42 a32 a22 a12
a41 a31 a21 a11
n1+n2=2
n1+n2=1
n1+n2=0
n1+n2=-1
n1+n2=-2
3n1-n2=-2d-12
3n1-n2=-2d-9
3n1-n2=-2d-6
3n1-n2=-2d-3
3n1-n2=-2d

Fig. 3

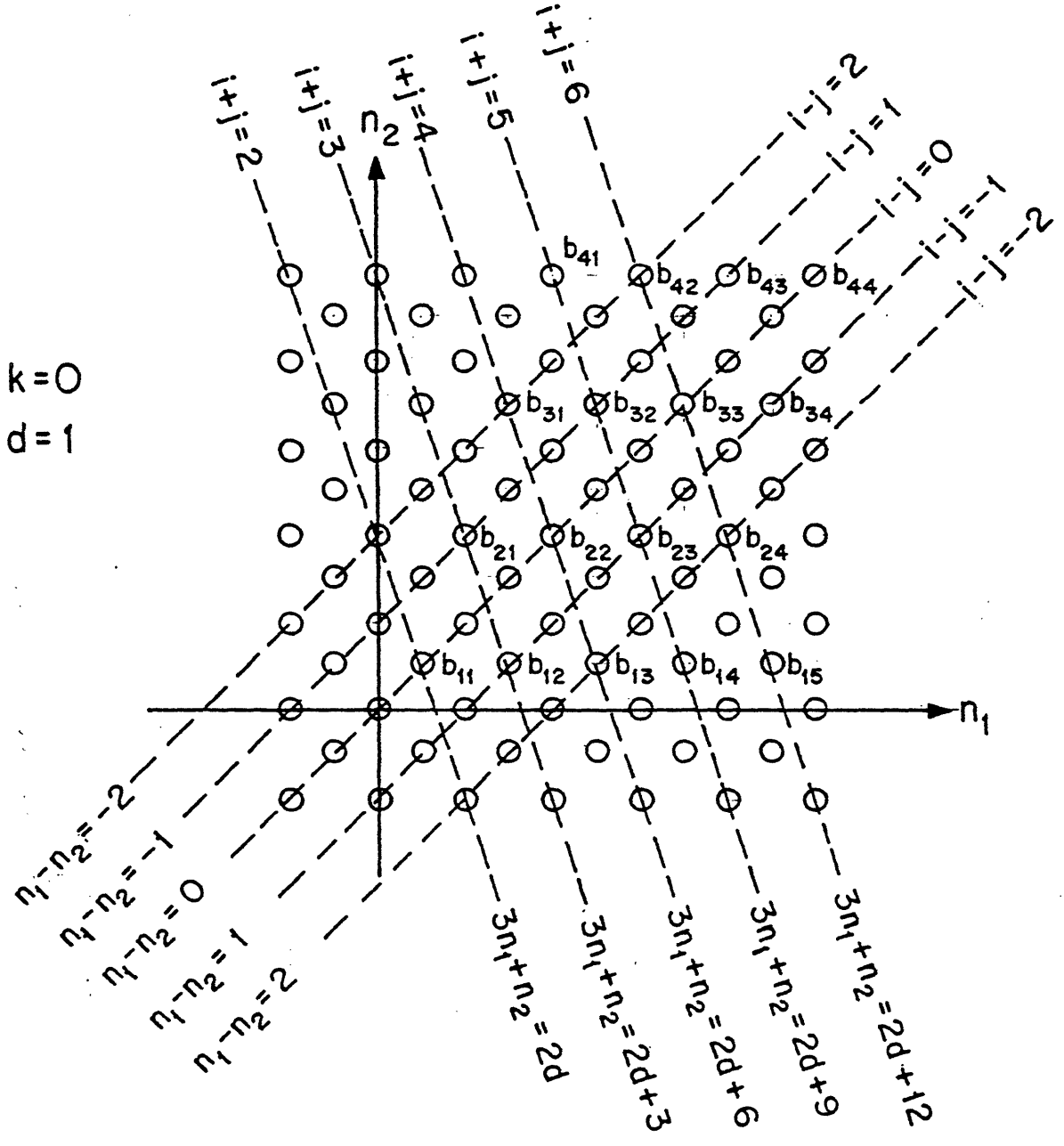
k=0
d=1
i+j=2
i+j=3
i+j=4
i+j=5
i+j=6
i-j=2
i-j=1
i-j=0
i-j=-1
i-j=-2
n2
n1
b41
b42
b43
b44
b31
b32
b33
b34
b21
b22
b23
b24
b11
b12
b13
b14
b15
n1-n2=-2
n1-n2=-1
n1-n2=0
n1-n2=1
n1-n2=2
3n1+n2=2d
3n1+n2=2d+3
3n1+n2=2d+6
3n1+n2=2d+9
3n1+n2=2d+12

Fig. 4

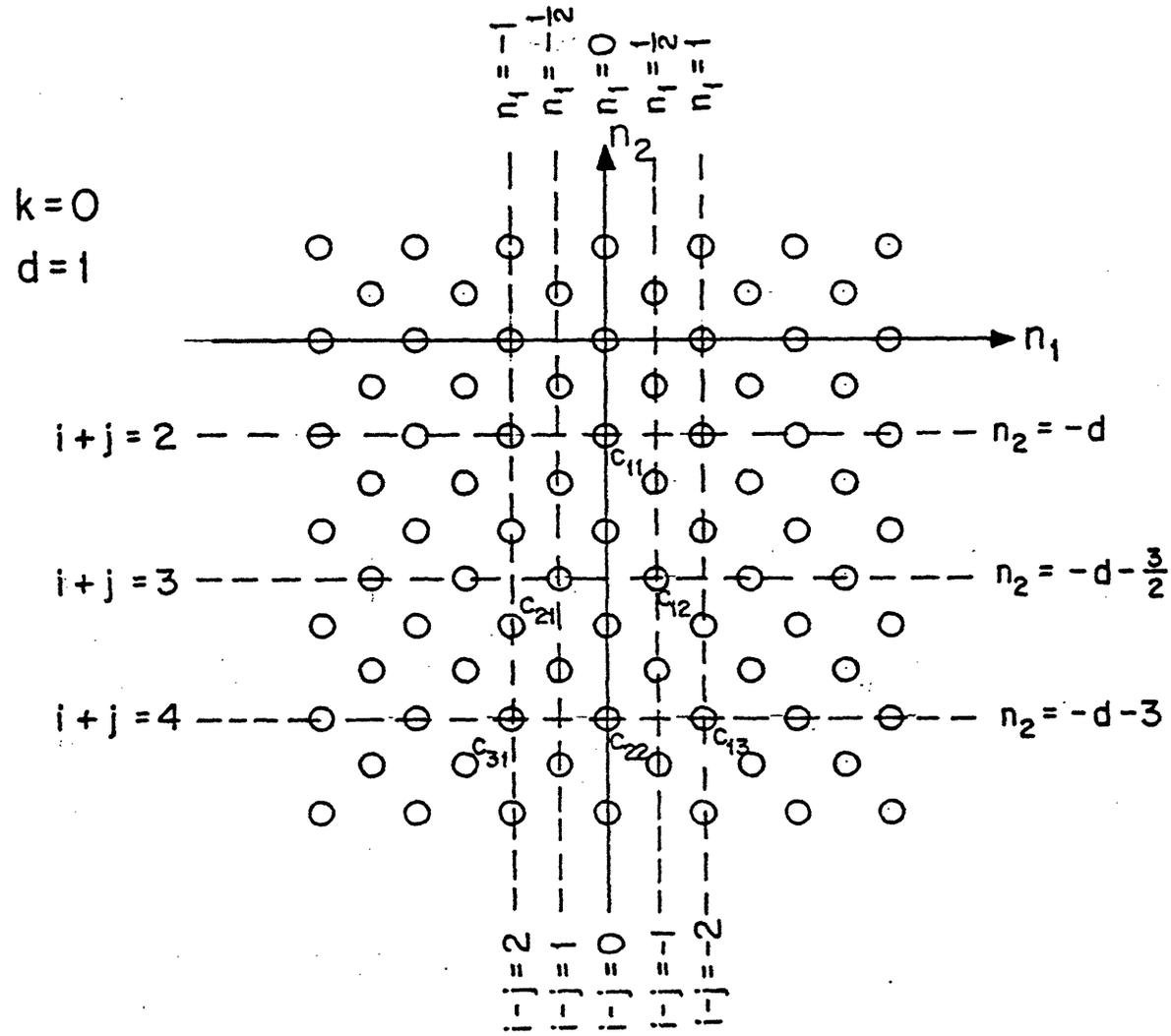
k=0
d=1
n2
n1
i+j=2
i+j=3
i+j=4
n2=-d
n2=-d-3/2
n2=-d-3
c11
c21
c12
c31
c22
c13
n1=1
n1=1/2
n1=0
n1=-1/2
n1=-1
i-j=2
i-j=1
i-j=0
i-j=-1
i-j=-2

Fig. 5