An Annual Report

NUMERICAL ALGORITHMS FOR FINITE ELEMENT COMPUTATIONS ON ARRAYS OF MICROPROCESSORS

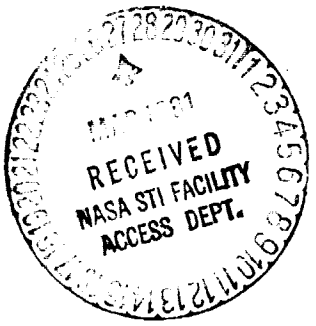
Submitted to:

Division of Structures and Dynamics
NASA/Langley Research Center
Hampton, VA 23665

Attention: Mr. David Loendorf

Submitted by:

J. M. Ortega
Professor and Chairman



Report No. UVA/528190/AMCS81/101

March 1981

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTER SCIENCE

UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA 22901

An Annual Report

NUMERICAL ALGORITHMS FOR FINITE ELEMENT COMPUTATIONS ON ARRAYS OF MICROPROCESSORS

Submitted to:

Division of Structures and Dynamics
NASA/Langley Research Center
Hampton, VA 23665

Attention: Mr. David Loendorf

Submitted by:

J. M. Ortega
Professor and Chairman

Department of Applied Mathematics and Computer Science
RESEARCH LABORATORIES FOR THE ENGINEERING SCIENCES
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA

Report No. UVA/528190/AMCS81/101 Copy No. 4
March 1981

This report summarizes work performed under NASA Grant NAG1-46 during the period September 1, 1980 to February 28, 1981 and constitutes the second semi-annual report. This work has been done under the technical monitorship of David Loendorf of Langley Research Center.

The primary accomplishment during this period has been the development of a multi-colored SOR program for the Finite Element Machine. The multi-colored SOR method, described in detail in the last semi-annual report, uses a generalization of the classical Red/Black grid point ordering for the SOR method. These "multi-colored" orderings have the advantage of allowing the SOR method to be implemented as a Jacobi method, which is ideal for arrays of processors, but still enjoy the greater rate of convergence of the SOR method.

The multi-colored SOR program was written in PASCAL and has run successfully on the current four processor version of the Finite Element Machine. The program has been written for a general nxn array of processors, however, and should be easily convertible to the 36 processor array when that is operational.

The current program was written to solve a general second order self-adjoint elliptic problem on a square region with Dirichlet boundary conditions, discretized by quadratic elements on triangular regions. This problem is described in more detail in the Appendix to this report. We note that for this general problem and discretization, six colors are necessary (and sufficient) for the multi-colored method to operate efficiently. The specific problem that was solved using the six-color program was Poisson's equation; for Poisson's equation, three colors are necessary and sufficient but six may be used.

In general, the number of colors needed will be a function of the differential equation, the region and boundary conditions, and the particular finite elements used for the discretization. One of the problems currently being investigated is how to determine in a systematic way the proper number of colors and the grouping of the unknowns into

the processors. Proper processor assignment is, of course, crucial for the success of the method. In the Appendix a rough analysis is given of some of the questions that arise in processor assignment. The six and three color orderings may be found in Figures 1 and 2 respectively.

The program has thus far been run on two test problems:

Problem 1: $U_{xx} + U_{yy} = 4$ in the unit square

$U = x^2 + y^2$ on the boundary

Problem 2: $U_{xx} + U_{yy} = 1$ in the unit square

$U = 0$ on the boundary

Problem 1 is a standard mathematical test problem. Problem 2 is a model for a fixed boundary membrane and was suggested by the project monitor, David Loendorf. The results on both problems have been very encouraging.

In addition to the program development, a number of questions relating to the efficiency of the multi-color method have been investigated.

1. Will the iterates produced by the multi-color SOR method converge?

2. What is the rate of convergence of the method, especially as a function of the overrelaxation factor?

3. How does one choose an optimum (or good) overrelaxation factor?

The answer to the first question is quite easy if the coefficient matrix of the system is positive definite. In this case, the multi-color ordering is just a permutation transformation of this matrix, and the permuted matrix is also positive definite and SOR will converge.

The other two questions are more difficult. The coefficient matrix does not have Property A, the classical condition of Young which allows a corre-

spondence between the eigenvalues of the SOR iteration matrix, and the Jacobi iteration matrix. Nevertheless, the computer results so far indicate that the rate of convergence of the multi-color SOR method applied to either Problem 1 or 2 above behaves, as a function of the relaxation factor, very much like the SOR method applied to the classical 5-point finite difference discretization of the Laplace's equation. These experimental results are very encouraging, since SOR on the 5-point discretization is essentially a best case situation. Work is continuing on attempting to understand why the experimental results are true.

It is anticipated that in the near future, additional more demanding proble will be attempted on the Finite Element Machine; in particular, Ms. Adams is planning to spend most of the summer of 1981 at Langley Research Center. We also plan to continue our investigation of certain variants of the conjugate gradient method, as an alternative to the multi-colored SOR method.

APPENDIX

The general problem considered is the second order self-adjoint elliptic problem

$$Lu(x,y) = -f(x,y) \qquad \text{in } \Omega \text{, the unit square}$$
$$u(x,y) = g(x,y) \qquad \text{on } \partial\Omega$$

where

$$Lu = au - (bu_x)_x - (cu_y)_y$$

where a,b,c are functions of x and y.

MATHEMATICAL SOLUTION (Quadratic Elements on Triangular Regions)

$$\int_{\Omega} (au - (bu_x)_x - (cu_y)_y)\, v = \int_{\Omega} -fv$$

Using the divergence theorem, the left side becomes:

$$\int_{\Omega} (auv + bu_x v_x + cu_y v_y) - \int_{\partial\Omega} (bu_x n_1 + cu_y n_2)\, v$$

where $\underline{n} = n_1 e_1 + n_2 e_2$ is the normal to $\partial\Omega$.

Note that for Dirichlet boundary conditions, the natural boundary condition $bu_x n_1 + cu_y n_2$ will always be zero on $\partial\Omega$. Therefore, the equation to be solved is:

$$\int_{\Omega} (auv + bu_x v_x + cu_y v_y) = \int_{\Omega} -fv$$

Now, to solve Poisson's Equation, $U_{xx} + U_{yy} = f$, set a=0, b=c=1 to get,

$$\int_{\Omega} (u_x v_x + u_y v_y) = \int_{\Omega} -fv \tag{1}$$

Discretize the unit square with M points in each row and column including the boundary points. Let $u = \sum_{i=1}^{M-1} \sum_{j=1}^{M-1} u_{ij} \phi_{ij} + \sum_{bp} u_{bp} \phi_{bp}$ and $v = \phi_{kj}$

where the ϕ's are the basis functions for Galerkin's Finite Element Method and the ϕ_{bp}'s represent those basis functions associated with boundary points.

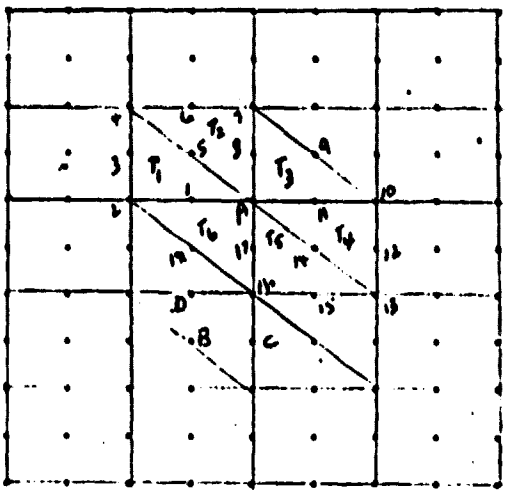
With these substitutions and definitions, (1) now becomes the following (2n-1)x(2n-1) system of linear equations for the solution of the interior points (values of u at discrete points H/2 units apart in both the x and y directions).

$$\begin{bmatrix} \int_\Omega \nabla\phi_{11}\cdot\nabla\phi_{11} & \cdots & \int_\Omega \nabla\phi_{km}\cdot\nabla\phi_{11} & \cdots & \int_\Omega \nabla\phi_{2n-1,2n-1}\cdot\nabla\phi_{11} \\ \vdots & & \vdots & & \vdots \\ \int_\Omega \nabla\phi_{11}\cdot\nabla\phi_{km} & \cdots & \int_\Omega \nabla\phi_{km}\cdot\nabla\phi_{km} & \cdots & \int_\Omega \nabla\phi_{2n-1,2n-1}\cdot\nabla\phi_{km} \\ \vdots & & \vdots & & \vdots \\ \int_\Omega \nabla\phi_{11}\cdot\nabla\phi_{2n-1,2n-1} & \cdots & \int_\Omega \nabla\phi_{km}\cdot\nabla\phi_{2n-1,2n-1} & \cdots & \int_\Omega \nabla\phi_{2n-1,2n-1}\cdot\nabla\phi_{2n-1,2n-1} \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{km} \\ \vdots \\ u_{2n-1,2n-1} \end{bmatrix} = \begin{bmatrix} -\int_\Omega f\phi_{11} - \sum_{bp} \nabla\phi_{11}\cdot\nabla\phi_{bp} \\ \vdots \\ -\int_\Omega f\phi_{km} - \sum_{bp} \nabla\phi_{km}\cdot\nabla\phi_{bp} \\ \vdots \\ -\int_\Omega f\phi_{2n-1,2n-1} - \sum_{bp} \nabla\phi_{2n-1,2n-1}\cdot\nabla\phi_{bp} \end{bmatrix}$$

By using the six basis functions for quadratic elements and performing the integration over a standard triangle, the integrals of the coefficient matrix can be evaluated. Likewise, by using a numerical technique, the integrals on the right side can be evaluated.

CONNECTIVITY

The nature of the basis functions (ϕ's) will cause the coefficient matrix to be sparse. In particular, ϕ_{ij} can be nonzero only on triangles the point ij is on. This implies that ϕ_{ij} is nonzero on either 6 or 2 triangles. This connectivity is illustrated for ϕ_A, ϕ_B, ϕ_C, and ϕ_D below:



ϕ_A: ϕ_B: ϕ_C:

ϕ_D:

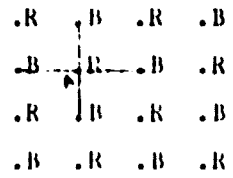
Note that ϕ_A is nonzero on triangles T_1, T_2, T_3, T_4, T_5, and T_6. Hence, only $\int_{\Omega} \nabla\phi_A \cdot \nabla\phi_1$, $\int_{\Omega} \nabla\phi_A \cdot \nabla\phi_2$, . . . , and $\int_{\Omega} \nabla\phi_A \cdot \nabla\phi_{18}$ need to be evaluated. Thus the iterative equation for the next value of U_A will only contain U_1, U_2, ... , and U_{18}. From the connectivity of ϕ_B, ϕ_C, and ϕ_D a similar analysis can be made for the solutions of U_B, U_C, AND U_D.

COLORING SCHEME

In order to solve for any of the points, say U_A for example, the values of all the points that point A is connected to must be known. These points must either have been calculated on a previous iteration or must have been calculated already on the current iteration.

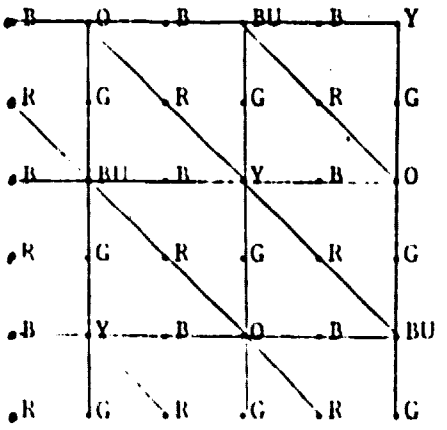
If processors are to be working in parallel, with each processor calculating values of a given number of these points, it is desired that the calculation be ordered so that each processor will not have to wait for values from other processors before computation can begin (no starvation in a sense). This can be accomplished by assigning to each point a color in such a way that the point will be a different color from all other points it is connected to. To illustrate this consider the simple connectivity of the 5-star:

Clearly two colors, red and black, will produce the desired result. This is illustrated below:



Note that each R point only needs values from B points, and each B point only needs values from each R point. Hence, all R points are calculated first, using B values that were available from the last iteration, and then all B points are calculated by using the R points that were calculated this iteration. The effect is that two Jacobi sweeps constitute one Gauss-Siedel sweep.

The same basic idea of coloring the points can be applied to the connectivity described earlier by using six colors. This is illustrated below and in Figure 1. Note from the figure that all the hypotenuse points can be the same color, R, because they are not connected. Likewise, all the vertical sides can be the same color, G, and all the horizontal size midponts can be the same color, B. Three extra colors are needed to color the vertices.

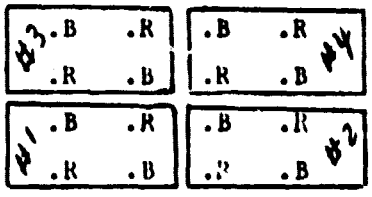


PROCESSOR ASSIGNMENT

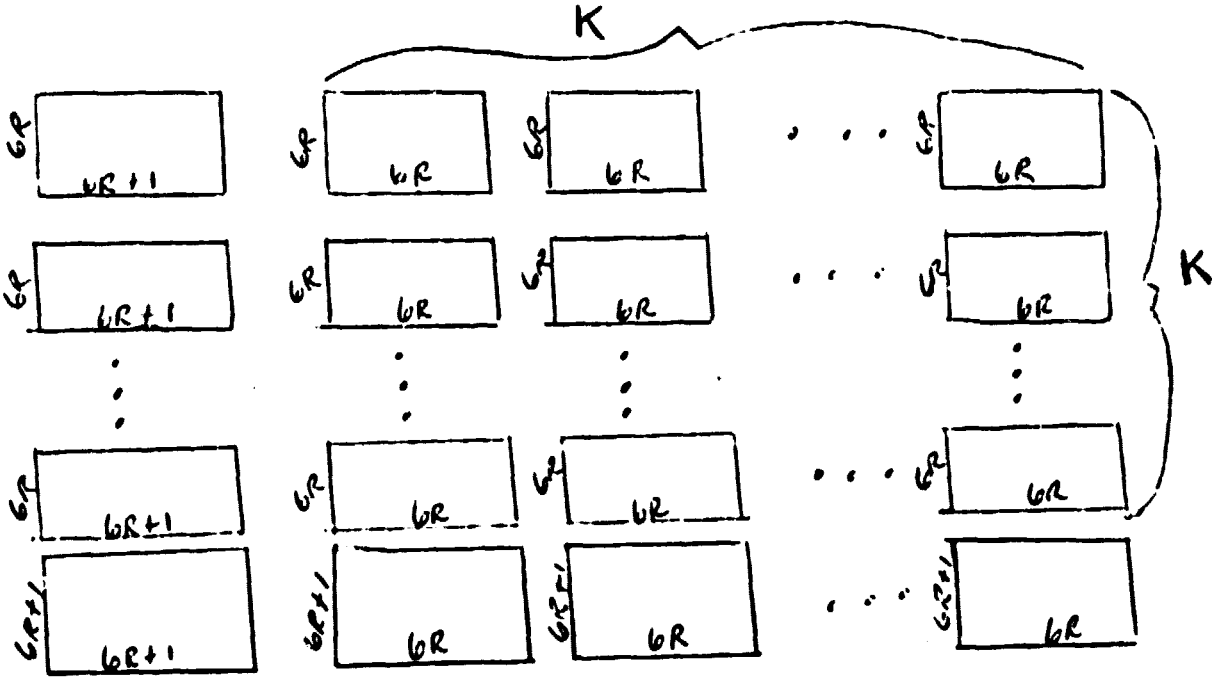
One consideration in assigning points to processors is the amount of work that will be required for each processor. Ideally, each processor should be kept busy at all times; that is, each processor should do the same fraction of the work. In this case, the above statements say that ideally each processor should have the same number of points to calculate.

Another consideration is the algorithm that will be used by each processor. If the points can be assigned in such a way that the algorithm in each processor will be the same (SIMD), the task of programming in parallel is greatly reduced, as well as the guarantee that each processor will be doing its fraction of the work.

These considerations give motivation for assigning to each processor the same color structure. For the simple connectivity of the 5-star and two colors and four processors, the assignment could be as follows:



For the problem under consideration, please note from Figure 1 that the color structure repeats every 6R by 6R square of points. Therefore, if processors are assigned to every 6R by 6R square, each processor will have the identical structure and an identical algorithm. This assignment is illustrated below. Note that because of the triangularization of the square region, an odd number of points will be in each row and column.



A few words should be said about the boundary processors. All boundary processors have values that do not have to be calculated, values that need not be sent to other processors, and values that need not be received from other processors. In this sense, the boundary processors could have a different algorithm than the other processors. However, from a programming standpoint, it is easier to test to see if a processor is on the boundary and take appropriate action than to write a separate algorithm for the boundary processor. The algorithm uses a PASCAL CASE statement to determine the type of processor. As written, the algorithm is SIMD, but the MIMD algorithm is easily extracted from the CASE statements.

ANALYSIS OF PROCESSOR ASSIGNMENT

First of all, since each processor is doing its fraction of the work, the best speedup over a uniprocessor one can hope to obtain is O(NP), where NP = $(K+1)^2$, the number of processors. In practice, however, this will not be realized since speedup will be slowed down by the processor inner communications (that is, data transmissions). An analytical expression for T(K), the number of local data transmissions needed when a K by K grid of 6*R by 6*R processors is used, is derived below but

$\hat{K}$ -- # of rectangles

H -- width of each rectangle

M -- # of points in each row M = 2n + 1 = interior plus boundary

R -- determines the 6R by 6R topology

B-A -- width of original square (1 for this example)

K -- # of 6R by 6R processors in a row (see picture)

$$M/6R = K + (6R+1)/6R \quad \text{or} \quad M = 6RK + 6R + 1 \qquad (1)$$

$$\text{But} \quad M = 2(B-A)/H + 1 \qquad (2)$$

Combine (1) and (2) to get

$$R = (B-A)/H(3K + 3) = \hat{R}/(3K + 3)$$

Since K is odd, 3K + 3 is even. This implies that $\hat{R}$ must be chosen to be even and divisible by 3K + 3. At first this appears to be a restriction, but it is not really since, if a specified $\hat{R}$ is not divisible by 3K + 3, it may be increased until it is, thereby improving the accuracy of the solution (Note that $\hat{R}$ increasing means H is decreasing).

It is an easy task to add up the transmissions that occur between processors (two-way) to get for the K by K topology the following:

$$T(K) = 6K^2 - 6K + 12KR \qquad (3)$$

Also, it is easy to get an expression for the number of transmission links (lines) needed:

$$L(K) = 4(2K^2 + K)$$

Of these links, almost ½, namely $4K^2$, will have either 1 or 2 transmissions only.

It is interesting to note that the number of transmissions for the 6R by 6R topology is bounded above and below by 3 and 2 times the number of transmissions for a 12R by 12R topology respectively:

$$2T(K) < T(2K+1) < 3T(K)$$

or (5)

$$2T_{12R \times 12R} < T_{6R \times 6R} < 3T_{12R \times 12R}$$

Also,

$$4L(K) < L(2K+1) \leq 5L(K)$$

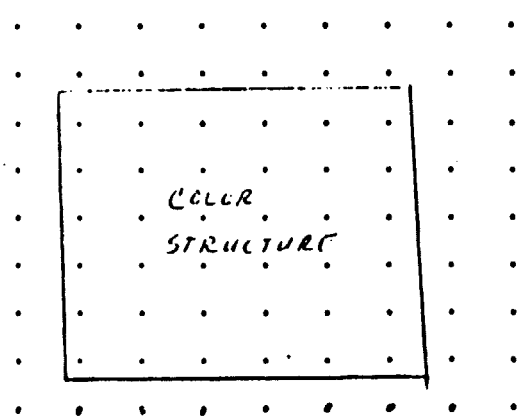
or (6)

$$4L_{12R \times 12R} < L_{6R \times 6R} \leq 5L_{12R \times 12R}$$

Equation (5) is encouraging, since reducing the blocks from 12R by 12R to 6R by 6R gives a potential speedup of 4 in computation time but only requires between 2 and 3 times as many data transmissions. If the architecture of the parallel computer is such that data transmission is rapid, a potential savings should occur. Also, if the architecture permits transmissions (sends and receives) to occur simultaneously with computation, the coloring of the points suggests that many of these data transmissions can occur while computation is being carried out. In other words, the amount of slowdown due to the transmissions is very architecture dependent and should be determined experimentally as well.

DATA STRUCTURE USED FOR EACH PROCESSOR

It was discovered that writing the algorithm became fairly easy once the "correct" data structure was discovered. At first, it was believed that each processor should only store its color structure. However, this idea was quickly abandoned when it was realized that each processor must receive values from other processors, and these values were used to perhaps calculate more than one point in the color structure. It was decided to use a 6R+3 by 6R+3 array to store the 6R by 6R color structure, one column of receives from the west processor, two columns of receives from the east processor, one row of receives from the south processor, and two rows of receives from the north processor. Below is an illustration of the data structure used.(R=1 in the illustration).



Note that this data structure applies to the boundary processors as well. The boundary values can be thought of as receives that are stored only once. Hence, the same data structure can be used in all the processors.

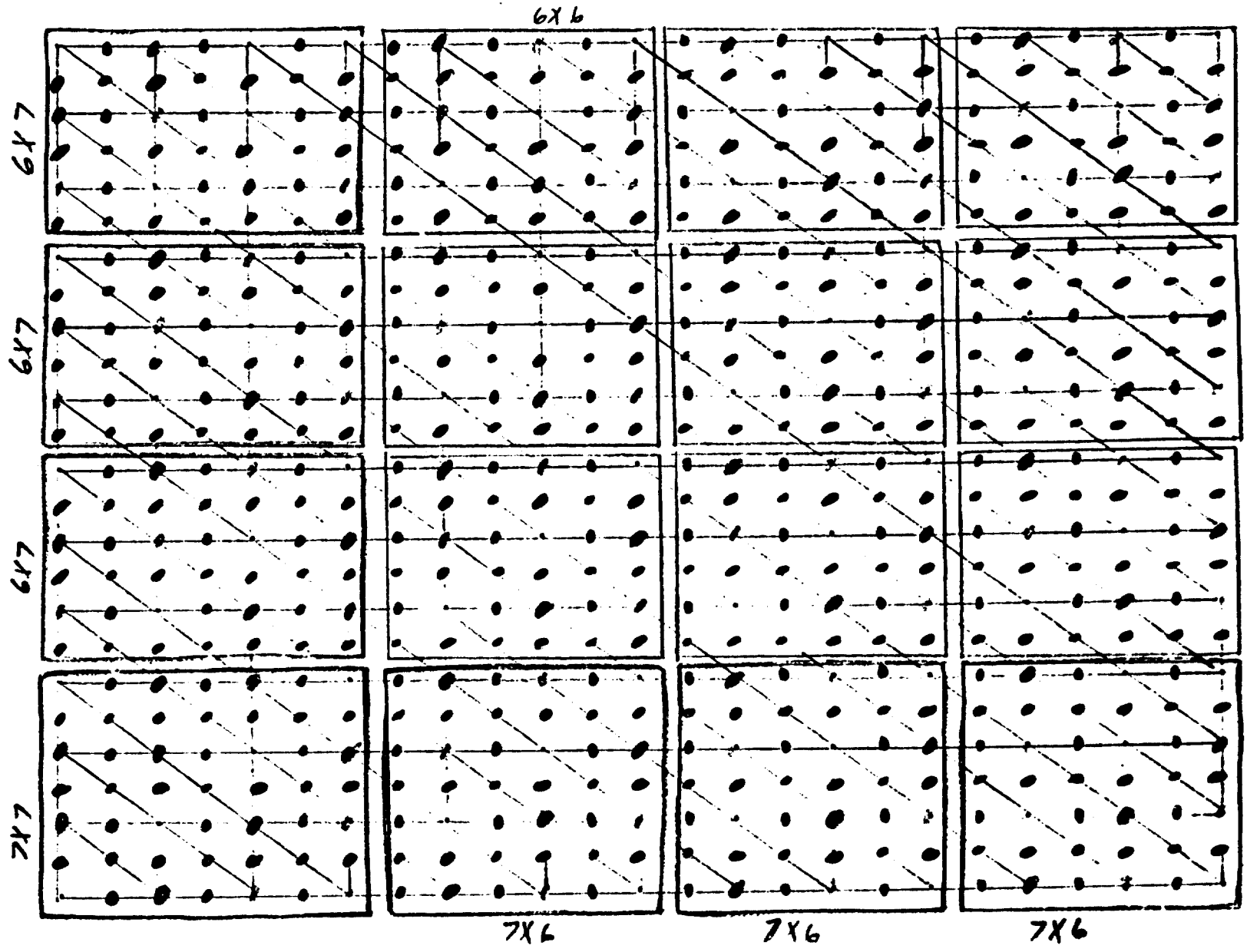


Figure 1
(6X6 Topology)

$\hat{R} = 12$
$K = 3$
$R = 1$

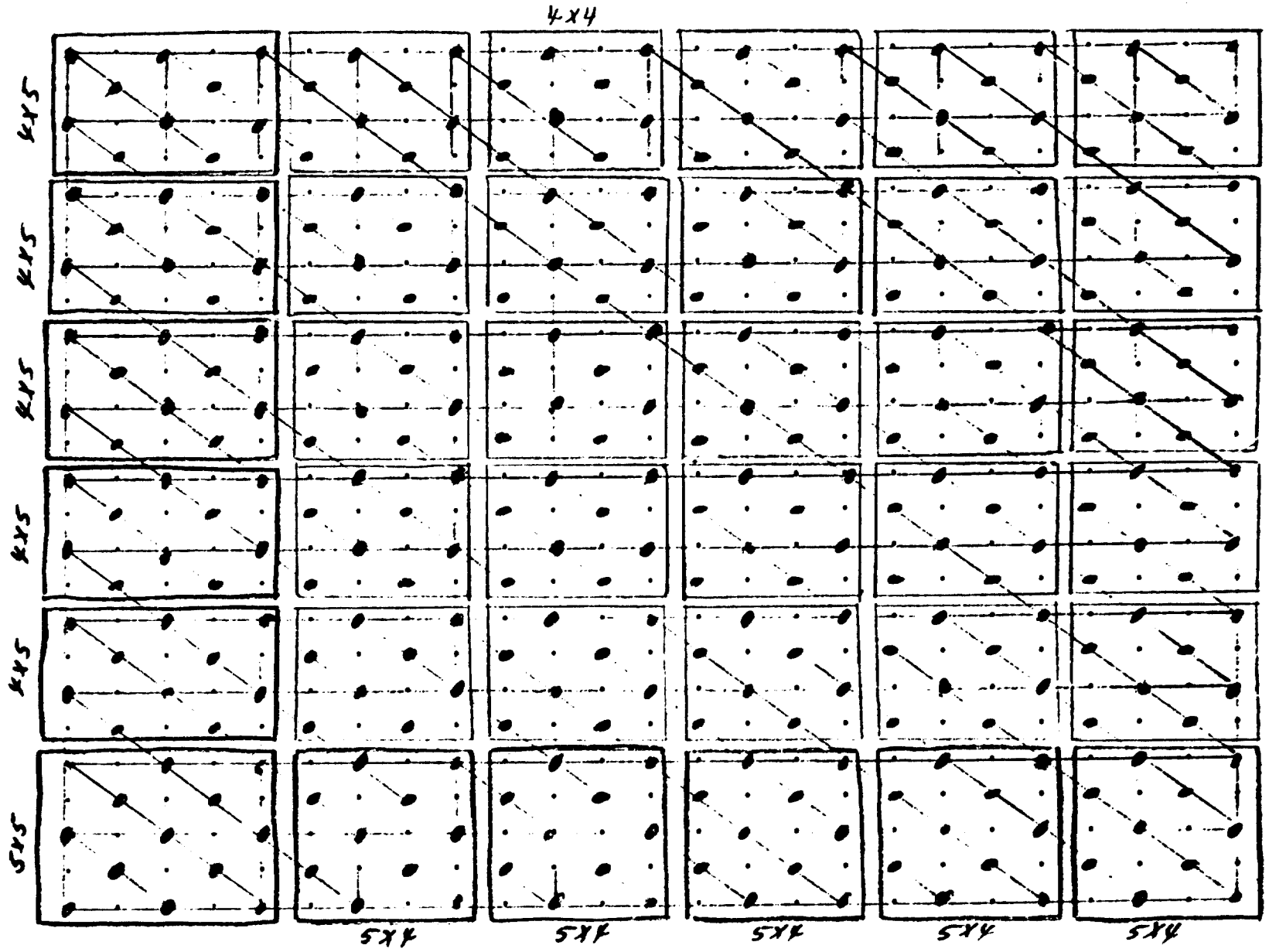


Figure 2