CONF-8705251--2

UCRL--96613

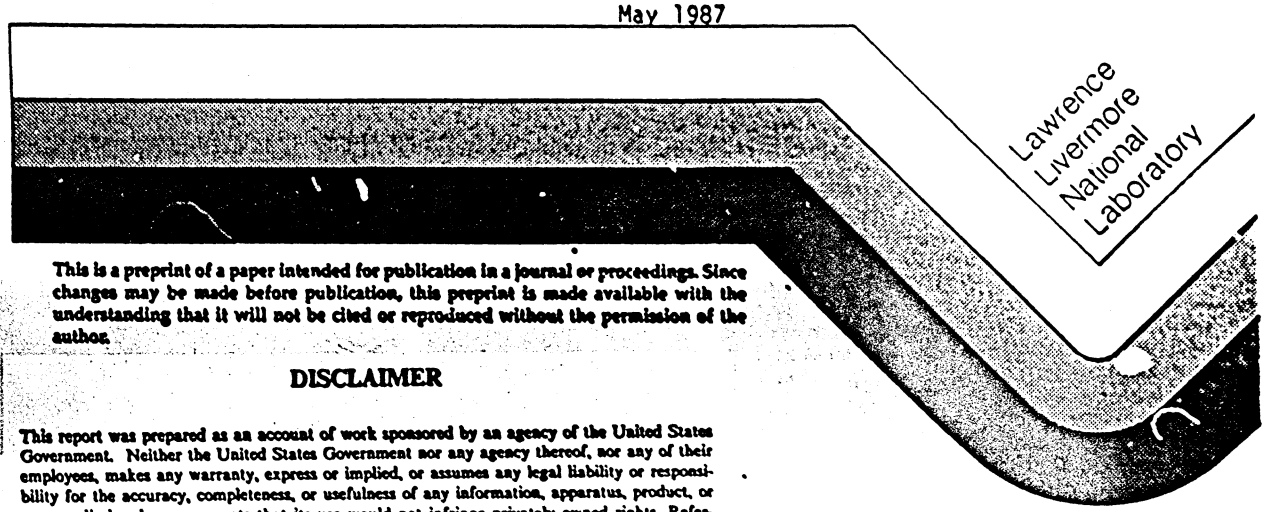
DE87 009563

SPRINT - THE SYSTOLIC PROCESSOR WITH A RECONFIGURABLE INTERCONNECTION NETWORK OF TRANSPUTERS

A. J. De Groot
E. M. Johansson
J. P. Fitch
C. W. Grant
S. R. Parker

This paper was prepared for submittal to IEEE
Fifth Conference on Real-Time Computer
Applications in Nuclear, Particle and Plasma
Physics
San Francisco, CA
5/12-14/87

May 1987



SPRINT - THE SYSTOLIC PROCESSOR WITH A RECONFIGURABLE INTERCONNECTION NETWORK OF TRANSPUTERS†

A. J. De Groot, E. M. Johansson, J. P. Fitch, C. W. Grant, S. R. Parker
Engineering Research Division
Lawrence Livermore National Laboratory
PO Box 808 L-156
Livermore, Calif. 94550

Abstract

The Systolic Processor with a Reconfigurable Interconnection Network of Transputers (SPRINT) is a sixty-four-processor multiprocessor developed at Lawrence Livermore National Laboratory for experimentally evaluating systolic algorithms and architectures. This paper describes the architecture of the SPRINT and several algorithms which have been executed on it.

Introduction

Computationally intensive problems drive research into parallel computing. Work on systolic array architectures [4], pioneered by H. T. Kung, indicates that for an important class of algorithms, computation speed can be linearly increased by the number of processors. For example, for matrix-matrix multiplication, a systolic array of $N \times N$ cells can be constructed to multiply two $N \times N$ matrices in $4N$ time steps. Although each cell is busy only one quarter of the time, the proportion is independent of N, the size of the array.

Researchers have designed systolic architectures for many computationally intensive tasks. Lawrence Livermore National Laboratory has developed the Systolic Processor with a Reconfigurable Interconnection Network of Transputers (SPRINT) to evaluate experimentally systolic algorithms and architectures.

Architecture of the SPRINT

The SPRINT, shown in figure 1, is a sixty-four-processor multiprocessor developed to evaluate experimentally and to demonstrate implementations of systolic arrays. The processors can operate in Single Instruction Multiple Data (SIMD) or Multiple Instruction Multiple Data (MIMD) mode. Processors operating in SIMD mode execute the same instruction simultaneously but operate on different data. Processors operating in MIMD mode can execute different instructions on different data. Each of the sixty-four processors is an INMOS Transputer, a 32-bit single-chip microprocessor. Timing tests show that the T414-15 Transputer performs approximately twice as fast as a Vax 11/780 for integer operations. The processors are interconnected in a reconfigurable network which can emulate networks such as the two-dimensional mesh, the triangular mesh, the binary tree, and the shuffle-exchange network. The two-dimensional mesh and the triangular mesh are used for many systolic algorithms. The binary tree is used in algorithms such as parallel search algorithms. The shuffle exchange network can be used for recursive doubling algorithms such as the Fast Fourier Transform (FFT).

The SPRINT is implemented as a set of eurocards interfaced to a MicroVax II. There are sixteen processor boards, one I/O board, and one network control board. A processor board contains four processors. Each processor has 128kbytes of memory and four asynchronous, full-duplex, 10 Mbit/sec links connected to a 4×8 crossbar switch. Each crossbar switch is memory mapped to its associated processor. The outputs of the sixty-four crossbar switches are interconnected in a six-dimensional hypercube. The I/O board interfaces the processors to the host machine, a MicroVax II. The network control board configures the network into the desired topology by controlling the 4×8 crossbar switches. The MicroVax II contains the programs and data for the SPRINT.

The SPRINT's hypercubic network is defined as follows: Number the vertices of an n-dimensional hypercube from 0 to 2^{n-1} in binary. There is a connection between each pair of numbered vertices whose bit patterns differ by exactly one bit. Let $x=[x_{n-1},\cdots,x_k,\cdots,x_0]$ be the binary label of a node. The neighbors of x are $[x_{n-1},\cdots,\overline{x_k},\cdots,x_0]$ for $0 \le k \le n-1$. Therefore, each node is connected to n other nodes. For example, in a 6-dimensional hypercubic network, node 100110 is connected to node 100111 (bit 0 different), node 100100 (bit 1 different), node 100010 (bit 2 different), node 101110 (bit 3 different), node 110110 (bit 4 different), and node 000110 (bit 5 different).

Although neither an n-level binary tree nor a 2^n-node shuffle-exchange network can be imbedded in an n-dimensional hypercubic network, the SPRINT network can emulate these networks by connecting appropriate nodes through multiple crossbar switches. This connection adds only a minimal delay to messages on the links and does not involve the processors. Figure 2 shows a 4-level tree imbedded in a 16-node version of a SPRINT network. The node numbers represent the node numbers in the underlying 4-dimensional hypercubic network. There cannot be a direct connection from processor 5 (101 in binary) to processor 3 (011 in binary) because the labels differ by two bits, bits 1 and 2. The connection from processor 5 to processor 3 passes through crossbar 7 (111 in binary), but the connection does not involve processor 7.

Theorem 1: No n-level binary tree, $n > 2$, can be imbedded into an n-dimensional hypercube so that neighbors in the tree are mapped to neighbors in the hypercube.

Proof: Label the nodes in the n-dimensional hypercube in binary, so that there is a connection between every pair of nodes whose label differs by exactly one bit. In an n-dimensional hypercube, half the nodes have an even number of ones in their labels. Call these nodes *even parity nodes*. The other half of the nodes have an odd number of ones in their labels. Call these nodes *odd parity nodes*. In a hypercube, the neighbors of a node have opposite parity of the node itself, be-

† This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under Contract W-7405-Eng-48.

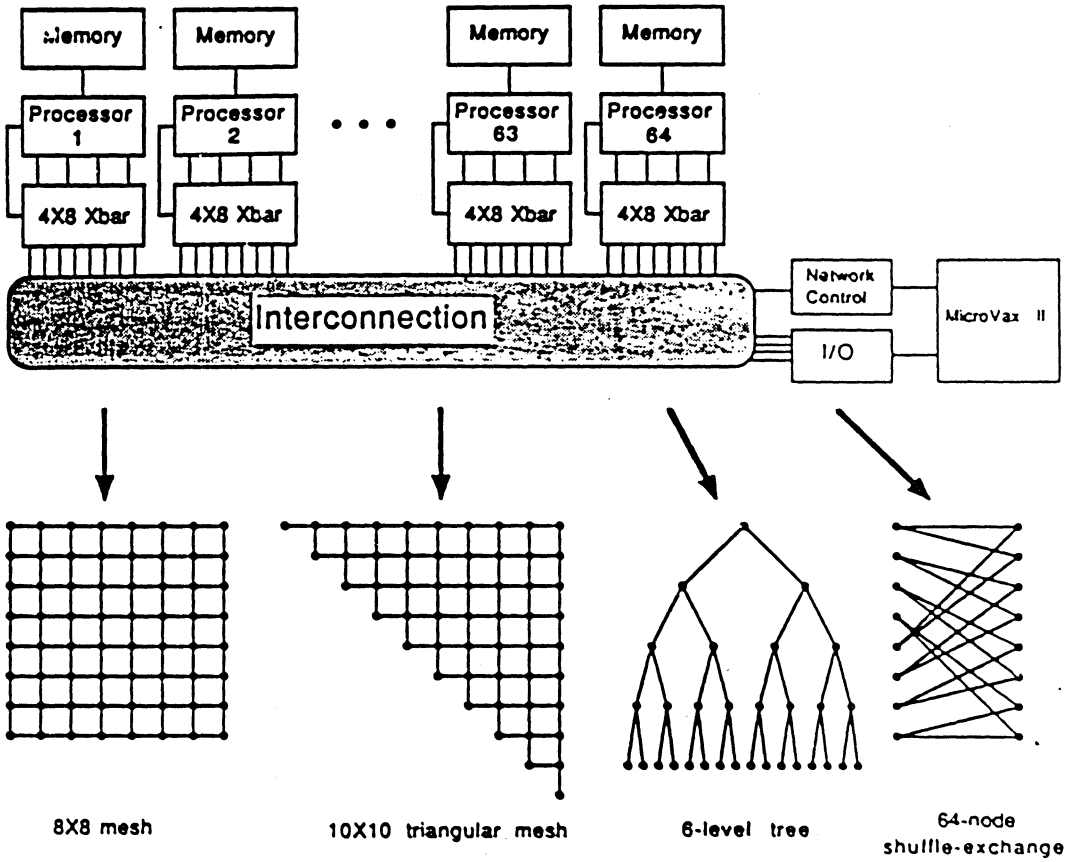


Figure 1. Architecture of the SPRINT

cause the label of each neighbor must differ by exactly one bit. This property must be maintained by any mapping of the tree onto the nodes of the hypercube. The parity of the root of the tree (level 0) differs from the parity of the two nodes at level 1, and the parity of both nodes must be equal. By induction it can be seen that all nodes at a given level must be equal in parity, and that successive levels in the tree must alternate in parity. Half the nodes are leaf nodes (level n-1), which are equal in parity, say parity p. Labelling these nodes with the same parity leaves the rest of the hypercube nodes (all of parity $\bar{p}$) for the rest of the tree. The rest of the tree nodes requiring parity p cannot be assigned, because the nodes of parity p are exhausted. For example, all nodes at level n-3 require nodes of parity p.

Theorem 2: No shuffle-exchange network with 2^n nodes, $n > 1$, can be imbedded into an n-dimensional hypercube.

Proof: Number the nodes in a shuffle-exchange network from 0 to 2^n-1. The shuffle-exchange network with 2^n nodes, $n > 1$, contains cycles of both even and odd lengths. There will be a three-cycle from node 0 to node 1 to node 2^{n-1} to node 0. The hypercube contains only even-length cycles. Therefore no network which contains odd-length cycles can be imbedded in a hypercube.

Algorithms Executed on the SPRINT

Several algorithms have been executed on the SPRINT [1,2], including matrix-matrix multiplication, the Fadeev algorithm, several image processing algorithms, and a finite difference time domain electromagnetic simulation algorithm. Define efficiency, η, to be

$$\eta = \frac{T_1}{n T_n}, \tag{1}$$

where T_1 is the time required to complete the algorithm with one processor, and T_n is the time required to complete the algorithm with n processors. Define speedup S_n to be

$$S_n = \frac{T_1}{T_n}. \tag{2}$$

The goal of parallel computation is to approach linear speedup, i.e., when $S_n = kn$, where k is independent of n. Linear speedup occurs when efficiency is independent of the number of processors. Then

$$S_n = \frac{T_1}{T_n} = \eta n.$$

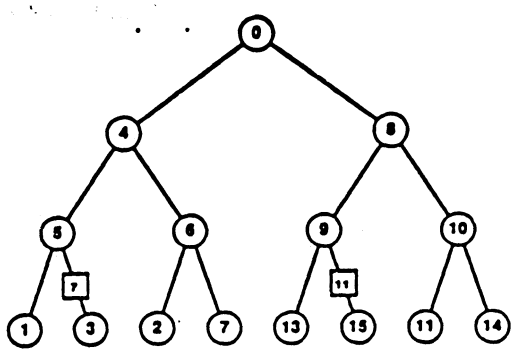


Figure 2. A 4 level binary tree mapped into the SPRINT network.

For each of these algorithms, the SPRINT exhibited linear speedup with respect to the number of processors and executed the algorithms more than ninety times faster than a Vax-11/780.

Matrix-Matrix Multiplication

Each element c_{ij} of the matrix product $C = AB$ can be expressed as the inner product of the ith row of A with the jth column of B, so that

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}. \tag{4}$$

This can be expressed as a recurrence in which the kth term of c_{ij} can be computed as

$$\begin{aligned} c_{ij}^{(0)} &= 0 \qquad 1 \le i, j \le n \\ c_{ij}^{(k)} &= c_{ij}^{(k-1)} + a_{ik} b_{kj} \quad 1 \le i, j, k \le n \end{aligned} \tag{5}$$

where $c_{ij}^{(n)}$ is the final result. This set of recurrences is implemented by various systolic architectures, including the engagement processor [6] and the Wavefront Array Processor (WAP) [5].

The engagement processor and the WAP are mesh-connected multiprocessors. Row i of A is supplied to the west connection of processor $(i,1)$, and column j of B is supplied to the north connection of processor $(1,j)$ for all $1 \le i,j \le n$. The product, matrix C, is contained in the processors. At time $t+1$, processor (i,j) computes $c_{ij}^{(t+1)}$, which is given by

$$c_{ij}^{(t+1)} = c_{ij}^{(t)} + a_{i,t+(i-1)+(j-1)} \cdot b_{t+(i-1)+(j-1),j}. \tag{6}$$

At time $t = (i-1)+(j-1)+n$, processor (i,j) completes computation of c_{ij}. The matrix product completes when the processor (n,n) completes its computation at time $t = (n-1)+(n-1)+n = 3n-2$. Efficiency, η_{n^2} is given by

$$\eta_{n^2} = \frac{n^3}{(3n-2)n^2} = \frac{n}{(3n-2)}, \tag{7}$$

which is slightly over $\frac{1}{3}$.

The SPRINT multiplied two $n \times n$ block matrices where each block is an $\frac{n}{q} \times \frac{n}{q}$ submatrix. The recurrences described above hold for the product of two block matrices, so the above analysis holds for the product of block matrices. The algorithm was modified to take advantage of the fact that the Transputer can compute and communicate simultaneously, so that while a processor is performing the matrix-matrix multiply-accumulate operation, it is sending its operands to its neighbors. This step reduces the total completion time to be

$$T_{n^2} = nT_{comp} + (2n-2)T_{io} = n(\frac{m}{n})^3 T_{mac} + (2n-2)(\frac{m}{n})^2 T_{vac}, \tag{8}$$

where T_{comp} is the computation time for an $\frac{n}{q} \times \frac{n}{q}$ submatrix multiply, T_{io} is the time required to communicate an $\frac{n}{q} \times \frac{n}{q}$ submatrix, T_{mac} is the time required to perform one scalar multiply-accumulate function, and T_{vac} is the time required to communicate one scalar to a neighboring processor. Efficiency, η_{n^2}, is given by

$$\eta_{n^2} = \frac{m^3 T_{mac}}{n^2(n(\frac{m}{n})^3 T_{mac} + (2n-2)(\frac{m}{n})^2 T_{vac})}. \tag{9}$$

If m is sufficiently large so that $\frac{m}{n} T_{mac} >> 2T_{vac}$, then

$$T_{n^2} \approx n(\frac{m}{n})^3 T_{mac} \tag{10}$$

and

$$\eta_{n^2} \approx \frac{m^3 T_{mac}}{n^3(\frac{m}{n})^3 T_{mac}} = 1. \tag{11}$$

Therefore, for sufficiently large m, efficiency is approximately 1, three times the efficiency of the engagement processor and the WAP. Linear speedup is predicted from these results.

The SPRINT exhibited linear speedup when multiplying two matrices. Figure 3 shows the execution of $n \times n$ block matrices on a $n \times n$ mesh of processors, where $1 \le n \le 8$. The block size, $\frac{m}{n}$ is 64. The figure shows that the SPRINT multiplied two 512 x 512 matrices of 32-bit integers in 12.7 seconds. The Vax-11/780 executing an optimized Fortran algorithm multiplied the matrices in 1434 seconds, which is 110 times longer than the SPRINT. The Cray-X/MP executing a vectorized Fortran algorithm multiplied the matrices in 18 seconds, which is 1.4 times longer than the SPRINT.

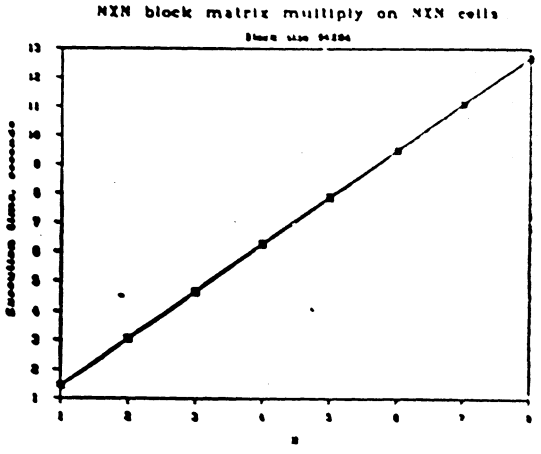


Figure 3. Execution of $n \times n$ block matrices on a $n \times n$ mesh of processors.

Fadeev Algorithm

The Fadeev algorithm [3] is a good candidate algorithm for systolic array implementation because of the several important linear algebra operations it can perform. The Fadeev algorithm calculates $CX + D$, where C and D are given, and X is the solution of $AX = B$, where A and B are given. All quantities are $N \times N$ matrices. The quantity calculated is $CA^{-1}B + D$. Other linear algebra operations are obtained by setting combinations of A, B, C and D to constants. For example, by setting $B = C = I$ and $D = 0$, the result is A^{-1}, the inverse of a matrix. By setting $A = I$, $D = 0$, the result is CB, the product of two matrices.

The SPRINT network topology used for the Fadeev algorithm is an $n \times n$ triangular mesh cascaded with an $n \times n$ mesh. This topology requires $\frac{3}{2}n^2 + \frac{1}{2}n$ processors, where $1 \le n \le 6$. The Fadeev algorithm requires $O(n^3)$ operations to complete. The SPRINT executed the Fadeev algorithm's $O(n^3)$ operations in $O(n)$ time with $O(n^2)$ processors. Efficiency is therefore a constant, implying that the SPRINT exhibited linear speedup with respect to the number of processors when executing the Fadeev algorithm.

Image Processing Algorithms

SPRINT executed several image processing algorithms including dilation, erosion, bilinear expansion, autocorrelation, and the median, low pass and high pass filters. For these algorithms, the network is configured to an 8×8 mesh. For an $m \times m$ image contained in an array of $n \times n$ processors, each processor contains an $\frac{m}{n} \times \frac{m}{n}$ subimage. Processor (i,j) contains the pixels $P[k,l]$, where

$$\frac{m}{n}(i-1)+1 \le k \le \frac{m}{n}i, \text{ and } \frac{m}{n}(j-1)+1 \le l \le \frac{m}{n}j. \quad (12)$$

Figure 4 shows a 4×4 array of processors containing a 12×12 image. The 4×4 dotted boxes represent the processors. Each processor contains a 3×3 subimage, represented in the figure by solid boxes. For example, processor (2,3) contains pixels $P[k,l]$, where $4 \le k \le 6$, and $7 \le l \le 10$.

In this class of processing algorithms, each resultant pixel $P_{out}[i,j]$ is given by

$$P_{out}[i,j] = f(P_{in}[i+k_1, j+k_2]), \quad -1 \le k_1, k_2 \le 1, \quad (13)$$

which is a function of its nearest neighbors. For example, in the image erosion algorithm, each output pixel $P_{out}[i,j]$ is given by

$$P_{out}[i,j] = \min(P_{in}[i+k_1, j+k_2]), \quad -1 \le k_1, k_2 \le 1 \quad (14)$$

which is the minimum of its nearest neighbors. Many of the pixels can be computed without any interprocessor communication. Each processor needs data from its neighbors only when calculating its boundary pixels. The pixels required by a processor are all pixels adjacent to its boundary pixels. For example, processor (2,3) in the figure can compute pixel $P[5,8]$ without communicating with its neighbors. However, the processor cannot compute pixel $P[6,8]$ without receiving pixels $P[7,7]$, $P[7,8]$, and $P[7,9]$. Processor (2,3) must receive those pixels from processor (3,3). The small dotted boxes adjacent to the solid boxes in the figure represent the storage necessary for the pixels adjacent to the boundary pixels.

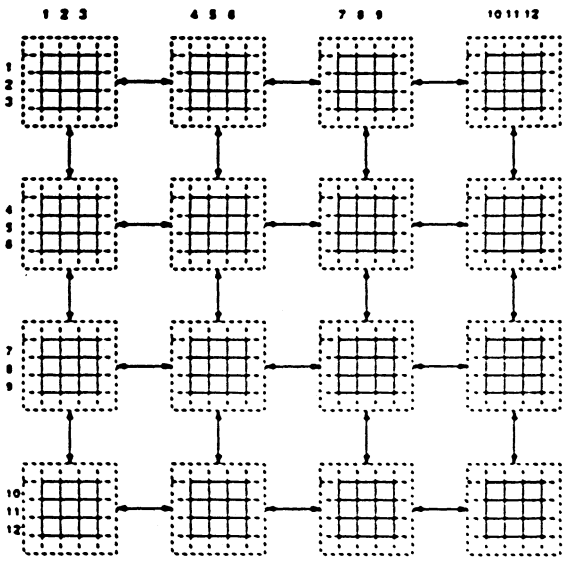


Figure 4. 4x4 mesh of processors containing 12x12 image.

Each processor needs to send and receive $4\frac{m}{n} - 4$ pixels, with the exception of the boundary processors. Define the time to calculate one pixel to be T_{calc}, and the time to send one pixel to be T_{comm}. The total computation for one processor, T_1, is

$$T_1 = m^2 T_{calc} \quad (15).$$

The total computation time required for n^2 processors to complete the algorithm, T_{n^2}, is

$$T_{n^2} = (\frac{m}{n})^2 T_{calc} + 8(\frac{m}{n} + 1)T_{comm}. \quad (16)$$

and efficiency, η_{n^2}, is given by

$$\eta_{n^2} = \frac{m^2 T_{calc}}{n^2(\frac{m}{n})^2 T_{calc} + 8(\frac{m}{n} + 1)T_{comm}}. \quad (17)$$

If $k = \frac{m}{n}$, then

$$\eta_{n^2} = \frac{k^2 T_{calc}}{k^2 T_{calc} + 8(k+1)T_{comm}}, \quad (18)$$

which means that m can be chosen to be sufficiently large so that the efficiency is independent of n. Therefore, linear speedup is predicted for these algorithms. Also,

$$\eta \to 1 \text{ as } k \to \infty. \quad (19)$$

Execution on the SPRINT verifies linear speedup with respect to the number of processors. If t seconds are required for $n \times n$ processors to compute an $m \times m$ image, then t seconds are required for $kn \times kn$ processors to compute a $km \times km$ image.

The SPRINT required less than 700 milliseconds to execute any of these algorithms on a 512×512 image using a 3×3 kernel. The Sun III and the Vax-11/780 require more than one minute to execute any of these algorithms.

Finite Difference Algorithm

The SPRINT executed a one-dimensional, finite difference time domain electromagnetic simulation algorithm. The algorithm calculates the electric and magnetic fields for the m points $P[j]$, $1 \le j \le m$, by solving Maxwell's equations. For this algorithm, the SPRINT's 64 processors are configured to be in a linear pipeline, so that any processor i is connected to processor $i-1$ and to processor $i+1$. The one-dimensional space is partitioned into $n = 64$ segments of size $\frac{m}{n}$. Processor i contains points

$$P[j], \quad \text{for } \frac{m}{n}(i-1)+1 \le j \le \frac{m}{n}i. \tag{20}$$

The computation of each point is dependent on only its nearest neighbors. Therefore, each processor communicates with only its nearest neighbors. The processors compute the fields for most of the points without any communication. Each processor needs data from its neighbors only when calculating its two end points. Define the time to calculate one point for one time step to be T_{calc}, the time to communicate one point to be T_{comm}, and the number of time steps to be S. The total time required for n processors to complete the algorithm, T_n, is

$$T_n = S(\frac{m}{n}T_{calc} + 2T_{comm}), \tag{21}$$

so that

$$\eta = \frac{T_1}{nT_n} = \frac{mT_{calc}}{mT_{calc} + 2nT_{comm}}. \tag{22}$$

If $k = \frac{m}{n}$, then

$$\eta = \frac{kT_{calc}}{kT_{calc} + 2T_{comm}}, \tag{23}$$

which means that m can be chosen to be sufficiently large so that the efficiency is independent of n. Therefore, linear speedup is predicted for this algorithm. Also,

$$\eta \to 1 \text{ as } k \to \infty. \tag{24}$$

Execution on the SPRINT verifies linear speedup with respect to the number of processors. If t seconds are required for n processors to complete an m-point problem, then t seconds are required for kn processors to complete a km-point problem. The same result should hold when the algorithm is extended to two- and three-dimensional problems.

Conclusion

The SPRINT and other systolic arrays can execute many computationally intensive tasks with speedup proportional to the number of processors in the array. The SPRINT has shown this experimentally for up to 64 processors, and should exhibit linear speedup with respect to the number of processors when executing other algorithms such as the FFT, beamforming, computer tomography, and other signal and image processing algorithms.

References

[1] A. J. De Groot, E. M. Johansson, and S. R. Parker, "A Systolic Array for Efficient Execution of the Fadeev Algorithm", to be published in SPIE's 31st Annual International Symposium on Optical and Optoelectronic Applied Sciences and Engineering, August 1987.

[2] A. J. De Groot, E. M. Johansson, and S. R. Parker, "A Systolic Array for Efficient Execution of Matrix Multiplication", submitted to the International Conference on Parallel Processing, August 1987.

[3] V. N. Fadeeva, *Computational Methods of Linear Algebra*, translated by Curtis D. Benster, Dover Publications, 1959.

[4] H. T. Kung and C. E. Leiserson, "Systolic Arrays (for VLSI)", *Sparse Matrix Proceedings 1978*, I. S. Duff and G. W. Stewart, ed., Society for Industrial and Applied Mathematics, pp. 256-282, 1979.

[5] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer and D. V. Bhaskar Rao, "Wavefront Array Processor: Language, Architecture, and Applications", *IEEE Trans. Comput.*, Vol. C-31, No. 11, Nov. 1982, pp. 1054-1066. also in *Proc. Conf. Advanced Res. VLSI, M.I.T.*, Jan. 1982, pp. 4-19.

[6] H. J. Whitehouse, J. M. Speiser and K. Bromley, "Signal Processing Applications of Systolic Array Technology", in *VLSI and Modern Signal Processing*, S. Y. Kung, H. J. Whitehouse and T. Kailith, ed., Englewood Cliffs: Prentice Hall, 1985, pp. 25-41.