SYSTOLIC COMMUNICATION

H. T. KUNG
Department of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Systolic communication is a general architectural technique for supporting efficient execution of systolic algorithms on a processor array. Under the systolic communication model, a cell in the array can directly access data on the cell's I/O queues, thus reducing the need of accessing the cell's local memory. By avoiding the unnecessary access to cells' local memories, the systolic communication can be much more efficient than the usual memory-to-memory communication when implementing systolic algorithms.

1. INTRODUCTION

The principle of systolic array architectures is that by replacing a single processing element (PE) with a regular array of processing elements, called *cells*, a higher computation throughput can be achieved without increasing the I/O bandwidth with the outside world [7]. Figure 1 depicts this principle when the processor array interfaces with an external memory.

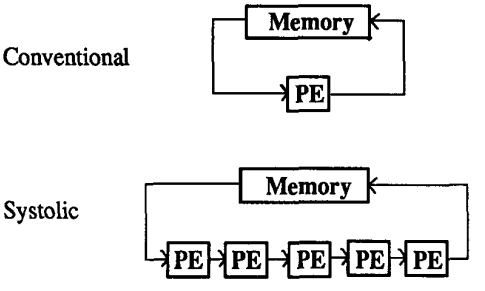

Figure 1. Conventional processor with one processing element (PE), and systolic array with an array of PEs or cells

The key of this approach is in the design of systolic algorithms. These algorithms ensure

---

The research was supported in part by Defense Advanced Research Projects Agency (DOD) monitored by the Space and Naval Warfare Systems Command under Contract N00039-87-C-0251, and in part by the Office of Naval Research under Contracts N00014-87-K-0385 and N00014-87-K-0533.

that once a data item is brought to the processor array from the outside it can be used effectively at each cell it passes while being "pumped" from cell to cell along the array. To support this style of computation, each cell needs to have efficient means of picking up data from the data streams flowing on the array and also inserting data into the data streams. One of the most challenging tasks in designing a systolic array is to streamline this interface between cell operations and flows of data on the array.

In this paper we introduce the notion of *systolic communication*, a general architectural technique for supporting efficient implementation of systolic algorithms. This notion is described and compared to the usual memory-to-memory communication in Section 2. In Section 3 the Warp systolic array machine is shown to support the systolic communication. A processor array such as Warp that supports the systolic communication will be most effective in executing systolic algorithms, for which data move during computation. This is illustrated by a simple matrix multiplication example in Section 4. Advantages of the systolic communication are summarized in Section 5. The last section contains some concluding remarks.

For presentation simplicity, all the discussions and illustrations in this paper are centered around one-dimensional processor arrays. It should be straightforward to see how the systolic communication concept generalizes to 2-dimensional or higher-dimensional arrays.

2. SYSTOLIC COMMUNICATION VS. MEMORY-TO-MEMORY COMMUNICATION

We assume that a cell in a processor array has its own *local memory*, and *input* and *output queues* through which the cell receives and sends data from and to other cells, respectively. There are two models under which a cell can operate on data sent to it by other cells. The first model is the *systolic model of communication*. Under this model the computation engine of each cell, called its CPU, can operate directly on data residing at the front of any of the cell's input queues and move computed results directly to any of the cell's output queues. Incoming and outgoing data need not be stored in the cell's local memory, if not required by the computation. By avoiding these local memory accesses, the machine can achieve high efficiency when executing many systolic algorithms.

The second model is the conventional *memory-to-memory model of communication*, often used in message-passing, distributed systems [6, 14, 16]. Under the memory-to-memory model of communication, the CPU of a cell does not directly read from or write to the cell's I/O queues. Data residing in an input queue must first be brought in the cell's local memory (usually by the operating system), before they are accessible to the cell's CPU. Similarly, computed results must first be stored in the cell's local memory, before they can be shipped out from the cell via output queues. Therefore a total of at least *four* local memory accesses are needed for a cell to update a data item flowing through the array. In contrast, with the systolic model of communication, there could be no local memory access at all; many systolic algorithms such as convolution need only accesses to the I/O queues [8]. Since memory access is typically a bottleneck in the cell's performance, the

systolic model of communication can be much more efficient than the memory-to-memory model of communication when implementing systolic algorithms. Figure 2 depicts these two models of communication for a 3-cell linear array. Note that each queue may be both an input queue of a cell and an output queue of a neighboring cell.

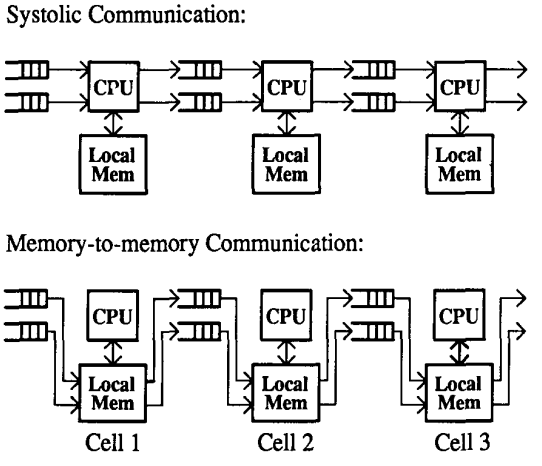

Figure 2. Two models of communication

There is a big difference between these two models with respect to the flexibility of data access by a cell's program. The local memory of a cell can be accessed *randomly*, while the I/O queues of the cell can be accessed only *sequentially*. Therefore, under the systolic communication, one must make sure that whenever the cell's program reads from an input queue, the right data item will appear at the front of the queue. Also, whenever the program writes to an output queue, it must be safe to insert the data item at the end of the queue in the sense that when the data item emerges from the front of the queue some cycles later, some other cell's program will be ready to read it. If the above is not ensured, then deadlocks may occur. Deadlock avoidance schemes have been proposed in a separate paper [9].

3. SYSTOLIC COMMUNICATION ON THE WARP MACHINE

During 1984-87 Carnegie Mellon developed a programmable systolic array called Warp [1]. Currently produced and marketed by GE, the machine has a linear systolic array of 10 or more identical cells. Each cell has a peak computation rate of 10 million floating-point operations per second (10 MFLOPS), giving the current 10-cell machine a peak rate of 100 MFLOPS.

More precisely, each cell is implemented as a programmable horizontal micro-engine, with its own microsequencer and program memory. The cell data path includes a 5 MFLOPS floating-point multiplier (Mpy), a 5 MFLOPS floating-point adder (Add), a local memory, and two data input queues. All these components, operating in a cycle time

of 200 ns, are connected through a crossbar. Via the crossbar the floating-point units can directly access data at the front of any input queue, and insert computed results at the end of any input queue of the next cell. Thus, the systolic communication is supported on Warp. A (much) simplied cell data path is depicted in Figure 3.

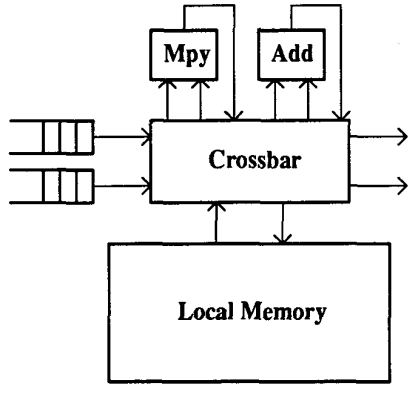

Figure 3. Cell data path (much simplied) for the Warp machine to support systolic communication

Utilizing the systolic communication is essential for the performance of each cell. Although already implemented with high-speed static RAMs, the cell's local memory can only perform one read and one write every 200 ns. Thus for the cell to sustain the peak performance of 10 MFLOPS its two floating-point units (Mpy and Add) together cannot read more than one word from the memory and write more than one word to the memory every 200 ns. This implies that to achieve the peak or near-peak performance some of the operands for the floating-point units must come directly from the input queues, and computed results must be sent directly to the input queues of the next cell.

4. MATRIX MULTIPLICATION EXAMPLE

Given a processor array such as Warp that supports the systolic communication, we need to map computations onto the array carefully to exploit the potential efficiency offered by the systolic communication. We use matrix multiplication to illustrate this point. Given $n \times n$ matrices $A=(a_{ij})$ and $B=(b_{ij})$, we want to compute their product $C=(c_{ij})$ on a linear processor array that supports the systolic communication. We assume that the linear array has k cells, with k much less than n, and in the illustrations below, $k=4$. We show two mapping methods; the first method, called the *data-staying* method, can not take advantage of the systolic communication, whereas the second method, called *data-moving* method, can. As a result, when implementing on Warp, we will see that the second method is much more efficient than the first method.

4.1. Data-staying Method

In the data-staying method, we evenly allocate the columns of matrix A and rows of matrix B to the cells so that cell i has submatrices A_i and submatrix B_i, as depicted in Figure 4. The input data assigned to a cell will stay at the cell during entire computation. The matrix product will be formed in two steps:

Step 1: Compute $C_i=A_i \cdot B_i$ on cell i for all i.

Step 2: Compute $C=C_1+C_2+\cdots+C_k$.

For Step 1 each cell takes $O(n^3/k)$ cycles. To perform Step 2, cell 1 pumps entries of C_1 to cell 2, where they will be added to entries of C_2; cell 2 pumps the resulting values to cell 3 where they will be added to entries of C_3; and so on. Cell i, for $i>1$, can start its computation as soon as it has received the first input from cell $i-1$. Thus the computation times for all the cells are overlapped significantly. It is easy to check that after n^2+k cycles, cell k will have all the entries of C. Since k is much less than n, we see that it is Step 1 that consumes the majority of the total computation time.

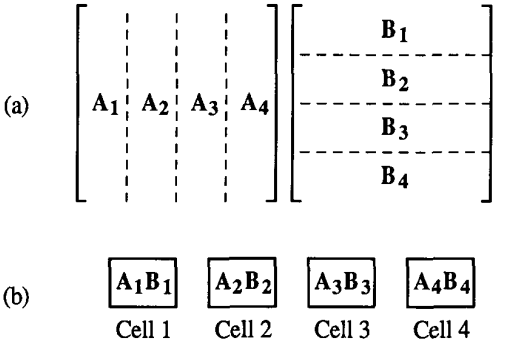

Figure 4. Data-staying method for matrix multiplication: (a) partitioning of the operand matrices, and (b) allocation of the resulting submatrices to the cells

For Step 1 cell i performs the following operations. It first loads entries of A_i and B_i into its local memory. Then it performs inner products for all pairs of row and column of A_i and B_i, respectively. Each inner product involves reading in a row of A_i and a column of B_i from the cell's local memory, and performing a sequence of multiply-accumulate operations. Two memory reads are needed for each multiply-accumulate operation as shown in Figure 5.

Suppose that this scheme is implemented on Warp. Then a multiply-accumulate operation takes 400 ns on a single cell, since this is what two memory reads take. This implies that for this implementation the performance of a 10-cell Warp machine is at most 50 MFLOPS, as compared to the 100 MFLOPS peak performance of the machine.

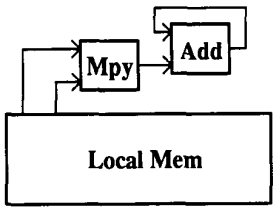

Figure 5. Two memory accesses per multiply-accumulate operation

4.2. Data-moving Method

In the data-moving method, while columns of matrix B are evenly allocated to the cells, entries of matrix A will be input to the left-most cell in the row-major ordering, and shifted to the right from cell to cell during computation. This is depicted in Figure 6.

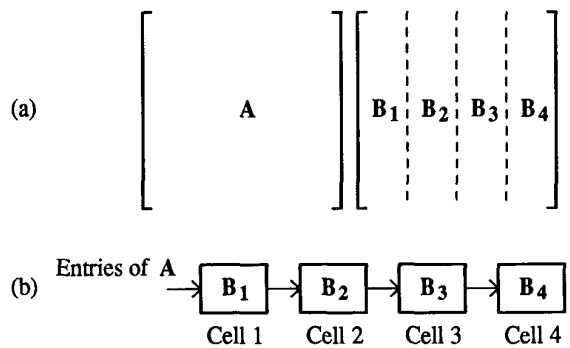

Figure 6. Data-moving method for matrix multiplication: (a) partitioning of matrix B, and (b) allocation of the resulting submatrices of B to the cells; entries of A moving to the right during computation

Cell i first loads entries of B_i into its local memory. Then it performs inner products for all pairs of row and column in A and B_i, respectively. Each inner product involves reading in a row of A from one of its input queues and a column of B_i from the cell's local memory, and performing a sequence of multiply-accumulate operations. Only one memory read is needed for each multiply-accumulate operation, as shown in Figure 7.

Suppose that this scheme is implemented on Warp. Then a multiply-accumulate operation takes 200 ns on a single cell, since this is what a memory read, a multiply, or an addition takes. This implies that for this implementation the performance of a 10-cell Warp machine can be close to the 100 MFLOPS peak.

5. ADVANTAGES OF SYSTOLIC COMMUNICATION

There are several advantages of using the systolic communication, including:

1. *Reduce access to the local memory of a cell*, as discussed above.

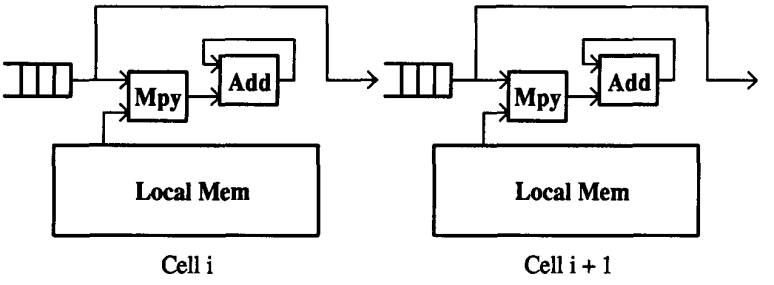

Figure 7. Using the systolic communication, only one memory access per multiply-accumulate operation needed

2. *Reduce address generation*, as there is no need to provide addresses to access I/O queues.

3. *Minimize synchronization overhead.* Synchronization is automatically provided in the sense that a cell stalls when it tries to receive data from an empty queue or send data to a full queue.

We have seen the performance advantage of using the systolic communication with applications on Warp. One example is the back propagation neural network simulation, whose inner-most loop is well-known to be a matrix-vector multiplication. We have two implementations running on Warp based on two partitioning schemes—network partitioning and data partitioning [13]. The latter method exploits the fact that weight-updates are usually small compared to the magnitude of the weight. Thus, we can run several simulations with fixed weights on different training patterns. This implies that the inner-most loop can actually be a matrix-matrix multiplication rather than matrix-vector multiplication. The data-partitioning implementation corresponds to the data-moving method for matrix-matrix multiplication described above. Because of the systolic communication, this method is about twice as fast as the network partitioning method. Our current Warp simulator is about eight times faster at simulating the NETtalk text-to-speech network than the fastest back-propagation simulator previously reported in the literature.

Another example is two Warp implementations of the 3×3 two-dimensional convolution on a 512×512 image. One method is to distribute evenly the image to the 10 cells in the Warp machine and then have each cell perform the convolution for one-tenth of the output image. The other method is based on a systolic algorithm [8], where each output pixel is computed over nine cells with intermediate result flowing from cell to cell. At each cell the systolic implementation requires no access to the cell's local memory at all, and is about twice as fast as the other implementation.

6. CONCLUDING REMARKS

The systolic communication exploits the fact that in systolic algorithms each cell receives its operands from the outside and sends its operands or results to the outside regularly. By accessing these data items directly, the cell can operate at a higher speed than what its local memory can support, as confirmed by applications on Warp. Therefore the systolic

communication is a method of increasing a cell's performance without increasing its local memory's bandwidth. It is instructive to contrast this with the principle of systolic array architectures as depicted in Figure 1, which represents a method of increasing a processor array's performance without increasing its communication bandwidth with the outside world.

Under the systolic model of communication a cell relies on I/O queues instead of general memory to increase the total bandwidth for accessing operands. Although queues are much easier and cheaper to implement in hardware than memory, they are in general difficult to program since their contents are not randomly accessible. As illustrated by the matrix multiplication example, users may need to do a careful mapping of their computations onto the processor array in order to make proper use of the queues. Once an efficient mapping is obtained, the systolic communication can offer very fast computations using a modest amount of hardware.

The systolic communication concept was present in many of the earlier systolic array machines [2, 3, 4, 5, 10, 11, 12, 15]. Actually for many special-purpose systolic arrays it was so natural to provide the systolic communication that the concept was hardly noticeable. The systolic communication became a significant architectural issue only when we started implementing programmable systolic array machines such as Warp. We realized that this communication model was very different from the usual memory-to-memory communication model. Adopting the systolic communication model has profound implications on how computations are synchronized and compiled for an array of processors; it usually represents a tradeoff between users' programming convenience and performance efficiency of the machine. Perhaps the main contribution of this paper is to make this important notion of systolic communication explicit.

Finally, for those who must decide whether or not a given processor array should be classified as a systolic array machine, the notion of systolic communication can offer a useful criteria for making the decision. A regular processor array which has high intercell communication bandwidth and supports the systolic communication must be efficient in implementing some systolic algorithms. Therefore it should be safe to classify this processor array as a systolic array machine.

REFERENCES

1. Annaratone, M., Arnould, E., Gross, T., Kung, H. T., Lam, M., Menzilcioglu, O. and Webb, J. A. "The Warp Computer: Architecture, Implementation and Performance". *IEEE Transactions on Computers C-36*, 12 (December 1987), 1523-1538.

2. Avila, J. and Kuekes, P. One-Gigaflop VLSI Systolic Processor. Proceedings of SPIE Symposium, Vol. 431, Real-Time Signal Processing VI, Society of Photo-Optical Instrumentation Engineers, August, 1983, pp. 159-165.

3. Blackmer, J., Frank, G. and Kuekes, P. A 200 Million Operations per Second (MOPS) Systolic Processor. Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV, Society of Photo-Optical Instrumentation Engineers, August, 1981, pp. 10-18.

4. Fisher, A.L., Kung, H.T., Monier, L.M. and Dohi, Y. "The Architecture of a Programmable Systolic Chip". *Journal of VLSI and Computer Systems 1*, 2 (1984), 153-169. An earlier version appears in *Conference Proceedings of the 10th Annual Symposium on Computer Architecture*, Stockholm, Sweden, June 1983, pp. 48-53.

5. Foster, M.J. and Kung, H.T. "The Design of Special-Purpose VLSI Chips". *Computer Magazine 13*, 1 (Jan. 1980), 26-40. Reprint of the paper appears in *Digital MOS Integrated Circuits*, edited by Elmasry, M.I., IEEE Press Selected Reprint Series, 1981, pp. 204-217.

6. Hayes, J. P., Mudge, T. N., Stout, Q. F., Colley, S. and Palmer, J. Architecture of a Hypercube Supercomputer. Proceedings of the 1986 International Conference on Parallel Processing, IEEE Computer Society, Aug., 1986, pp. 653-660.

7. Kung, H.T. "Why Systolic Architectures?". *Computer Magazine 15*, 1 (Jan. 1982), 37-46.

8. Kung, H.T. Systolic Algorithms for the CMU Warp Processor. Proceedings of the Seventh International Conference on Pattern Recognition, International Association for Pattern Recognition, 1984, pp. 570-577. A revised revion appears as Chapter 3 in *Systolic Signal Processing Systems*, edited by E. E. Swartzlander, Jr., pp. 73-95, New York, Marcel Dekker, 1987.

9. Kung, H. T. Deadlock Avoidance for Systolic Communication. Proceedings of the 15th Annual International Symposium on Computer Architecture, June, 1988.

10. Lopresti, D. P. "P-NAC: A Systolic Array for Comparing Nucleic Acid Sequences". *Computer Magazine 20*, 7 (July 1987), 98-99.

11. McCanny, J. V, and McWhirter, J. G. "Some Systolic Array Development in the United Kingdom". *Computer Magazine 20*, 7 (July 1987), 51-63.

12. Nash, J. G. and Petrozolin, C. VLSI Implementation of a Linear Systolic Array. Proc. ICASSP '85, Tampa, Florida., March 26-29, 1985, pp. 1392-1395.

13. Pomerleau, D. A., Gusciora, G. L., Touretzky, D. S. and Kung, H. T. Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second. Submitted to the IEEE Second International Conference on Neural Networks, April, 1988.

14. Seitz, C. L. "The Cosmic Cube". *Comm. ACM 28*, 1 (January 1985), 22-33.

15. Symanski, J.J. Systolic Array Processor Implementation. Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV, Society of Photo-Optical Instrumentation, August, 1981, pp. 27-32.

16. Tanenbaum, A. S. "Network Protocols". *Computing Surveys 13*, 4 (1981), 453-489.