1991 International Conference on Parallel Processing

B-SYS: A 470-Processor Programmable Systolic Array[†]

Richard Hughey and Daniel P. Lopresti
Department of Computer Science
Brown University
Box 1910
Providence, RI 02912

Abstract

This paper presents an architecture for programmable systolic arrays that provides simple and efficient systolic communication. The Brown Systolic Array is a linear implementation of this Systolic Shared Register architecture; a working 470-processor prototype system performs 108 MOPS. A 32-chip, 1504-processor implementation could provide 5 GOPS of systolic co-processing power on a single board.

Keywords: systolic array, parallel processing, VLSI, SIMD, sequence comparison.

Introduction

The systolic array is perhaps the most significant architectural development inspired by the revolution in VLSI fabrication technology. By pumping data through a regular network of hundreds or thousands of simple processing elements, computationally intensive problems can be solved many times faster on systolic arrays than on traditional machines. Since the introduction of the systolic paradigm, systolic co-processors have been proposed to solve a wide variety of problems. However, new VLSI single-purpose processing elements can require many months to design, simulate, fabricate, and test. Because of this great effort required to construct new arrays, research has turned to programmable systolic arrays. In this paper, we present both a general architecture for programmable systolic arrays and a working implementation based upon it. This *Systolic Shared Register* (SSR) architecture preserves the simple communication of single-purpose systolic arrays while providing the user with a fully programmable systolic co-processor.

To evaluate the SSR model, the Brown Systolic Array (B-SYS) has been designed, simulated, fabricated, tested, and used. B-SYS is particularly well suited to combinatorial problems: those requiring only integers and simple operations. These problems present large potential gains from systolic processing since small processing elements are sufficient, and the smaller each individual processing element, the easier it is to construct massively parallel machines. Many combinatorial problems (including data compression, dynamic programming, graph, image processing, and other problems) can efficiently use hundreds of thousands or even

[†]This research was supported in part by NSF grants MIP-8710745 and MIP-9020570 and by ONR and DARPA under contract N00014-83-K-0146 and ARPA order no. 6320, Amendment 1.

millions of processing elements; to achieve such high processor densities in a reasonable amount space, small and simple processing elements must be used.

A B-SYS prototype system has been running since the early fall of 1990. Ten B-SYS chips are linked together to form a small one-board, 470-processor linear systolic array which has been used to run various combinatorial algorithms. This prototype performs 108 million 8-bit operations per second (MOPS). We estimate that an easily expandable, low-cost single-board system of 1504 processing elements could perform five billion 8-bit additions each second (5 GOPS). The two major sections of this paper are devoted to the Systolic Shared Register paradigm and the Brown Systolic Array, respectively.

Systolic Shared Register Architectures

Perhaps the most critical issue in the design of programmable systolic arrays is that of systolic communication; the simple methods of single-purpose systolic systems should be maintained, but the architecture must be fully programmable. To address this issue (and others), the Systolic Shared Register architecture features a regular topology, broadcast instructions, systolic shared registers, and a stream model of data flow. The first two items follow directly from the algorithmic requirements of the class of systolic algorithms. The third provides efficient systolic communication and more flexible dispositions of computation and memory throughout the VLSI chip. Finally, the stream model of data flow is an elegant method for programming systolic co-processors.

Regular Topology

To support systolic algorithms, all of which use a regular topology by definition, the systolic co-processor itself should have a regular topology [1]. A fixed-degree regular network with nearest-neighbor connections (such as a line, square mesh, or hexagonal mesh) provides for the orderly and extensible mapping of algorithms to the systolic array.

Broadcast Instructions

Per-processor programmability currently requires too much area for inclusion in high-density systolic arrays, a result of both the size of the program store and the complexity of the instruction sequencer. To form arrays with hundreds of thousands of processing elements, there should be many processing elements on each chip; otherwise, the physical space used by the co-processor would be excessively large. Typical independently programmed systolic processing elements, such as iWarp, only have one processing element per chip [2]. Additionally, when large numbers of MIMD processors (hundreds or thousands) are used for systolic algorithms, the programs stored in each processor tend to be similar: by definition, systolic algorithms require only a small number of cell programs [1]. Thus, the

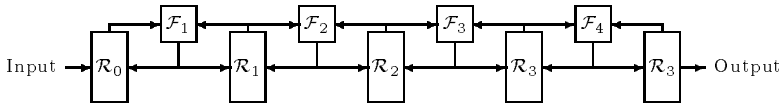

Figure 1: Linear SSR processor array with data stream.

inherent power of a MIMD machine is often not used for systolic algorithms.

Shared Registers for Communication

In a programmable systolic array, data cannot be moved automatically since different applications require different types of data movement; data movement must be explicitly specified. Several machines provide this ability with special instructions and special queues or registers for data movement [2, 3]. In an SSR architecture, however, data moves through the array as a natural result of the user's program.

Figure 1 shows a segment of a linear SSR array. Each functional unit ($\mathcal{F}_i$) is adjacent to two register banks. Each functional unit can access data values from the register banks directly to its west and east for both input and output. Input to and output from the array can take place at either end. Systolic communication in an SSR architecture uses these shared register banks. There are no internal registers in the functional units, though a small number of flag bits may be present. This separation of computation and storage is a hallmark of the SSR architecture.

The use of broadcast instructions and relative addressing of register banks prevents contention; dual-ported memory is not required since no two functional units attempt to access the same register bank simultaneously. Simple switches between functional units and register banks can control the flow of information.

With Systolic Shared Registers, data flow takes place concurrently with computation. For example, the instruction

$$E_2 \leftarrow \min(W_0, W_1)$$

not only performs a minimization but also moves data through the array from west to east.

The SSR paradigm also does not arbitrarily limit the number of communication channels between processors; the user may decide to use the first few registers of each register bank for local storage, the next for transmission of data from west to east, the next for another data stream, and so on. In summary, shared registers provide a small, fast, flexible, and elegant implementation of systolic communication.

Stream Programming

Finally, SSR machines are programmed using a data stream paradigm, allowing the inputs and outputs at both sides of a linear array to be linked to file streams or functions by the host program. In mesh architectures, all edge processing

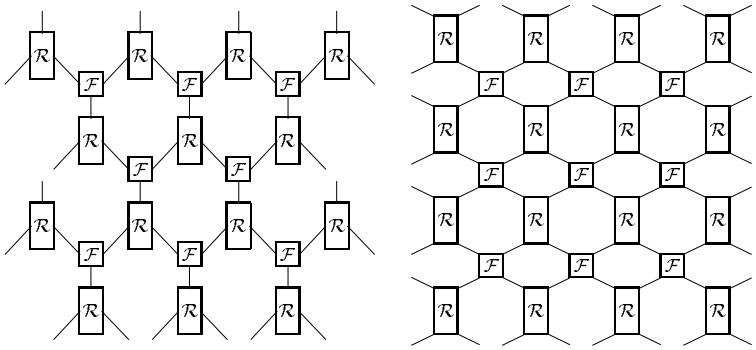

Figure 2: SSR networks.

elements, singly or in groups, may be linked to data streams. For example, a data stream of n-element vectors forming an $n \times n$ matrix could be linked to the entire west side of a mesh. The use of a *logical* data stream is crucial. It is possible that a single stream of data may use different physical registers at different times, changing register meanings to produce a more efficient program. This stream model of data flow allows the programmer to ignore such detail.

Other SSR Networks

The Systolic Shared Register paradigm is not restricted to linear arrays. Figure 2 shows abstract designs for hexagonal and octagonal mesh SSR machines. In the octagonal mesh, each functional unit can communicate with eight neighboring functional units through its four adjacent register banks. Although there are two boundary edges on corner memory nodes in the hexagonal mesh and three in the octagonal mesh, just one I/O interface for each boundary register bank is needed since only one value is written to any specific register bank during the execution of an instruction. This is a result of using SIMD broadcast instructions and relative addressing of the register banks.

The Brown Systolic Array

As mentioned above, the Brown Systolic Array is a working implementation of a linear SSR machine. Its original purpose was to provide a programmable string comparison co-processor suitable for the Human Genome Project, which is analyzing the three billion nucleotides of human DNA [4]. The need for inexpensive yet powerful hardware for this project is clear, and B-SYS is well suited to the task of performing many different comparison algorithms for the computational biologist. As a side advantage, B-SYS (being programmable) is able to perform any combinatorial problem suitable for a linear systolic array.

The architecture of an array segment is shown in Figure 3. As can be seen, each 8-bit functional unit has local flags (8) and buffers as well as access to two 16-

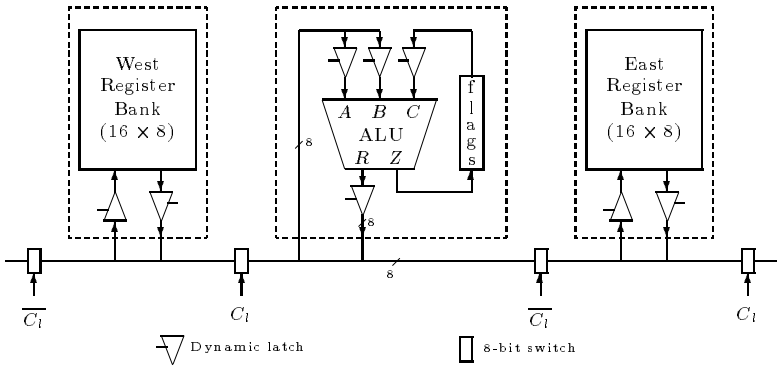

Figure 3: Segment of the B-SYS array.

word register banks via complementary 8-bit switches. A context flag can control conditional instruction execution. The B-SIM behavioral simulator (with both TEX snapshots and array animation using the Tango system [5]) proved crucial for algorithm development and early evaluation of this architecture.

The fully custom Brown Systolic Array chip was designed using Berkeley's Magic design suite and the COSMOS simulator [6, 7]. The 86-pin, 6.9 mm×6.8 mm chip was fabricated by MOSIS using a 2 μm CMOS process in May, 1990.

As is illustrated by the floor-plan (Figure 4), the B-SYS chip contains 47 functional units, each layed out with its west register bank (identical to the structure shown in the expanded processor) and one additional register bank. Each row of the chip has its own control circuitry, including a finite state automata (FSA) for control signal generation and two decoders, one for the 16-element register banks and one for the eight flags. Also visible are row buffers for the 16 bits of function information associated with each instruction.

Because of the simplicity of the SSR architecture, the 1730 transistors of the functional unit and register bank pair are densely packed in a 999 μm × 607 μm cell. The modestly sized chip with 47 functional units has 85 000 transistors. A larger die with a smaller technology could fit well over 500 functional units and register banks on a single chip.

Because sending information on and off chip is relatively slow, it is not practical for end functional units to read operands from adjacent chips. For this reason, the register bank shared by the rightmost processing element of one chip (block 47 in Figure 4) and the leftmost of the next (block 1) is duplicated. This performs the function of a write-through cache so that chip I/O does not take place during the loading of instruction operands. Whenever a result is written to the right, the value is sent both to the shadow register bank (above the B-SYS logo) and off-chip. At the same time, an input is received (either from an adjacent chip or

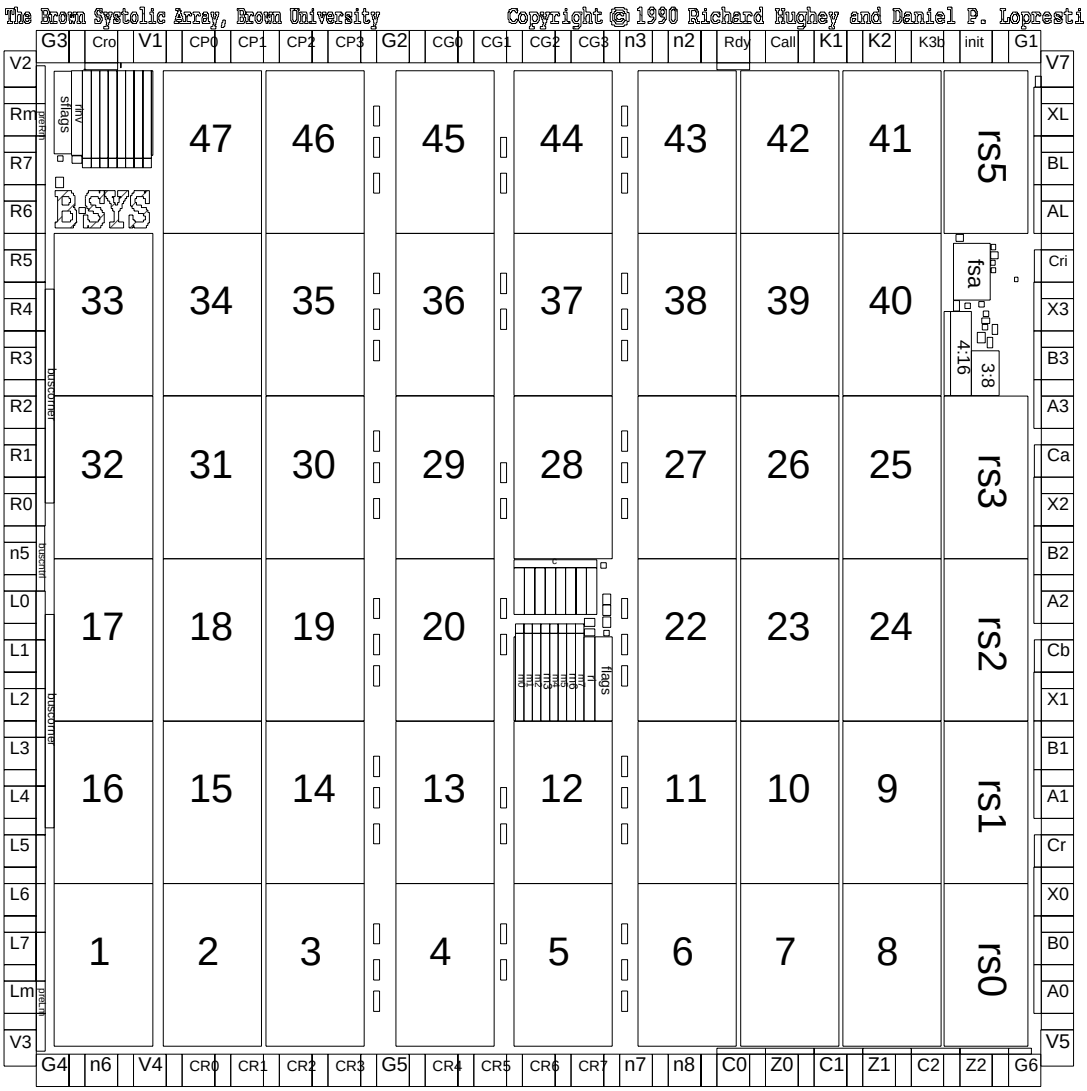

Figure 4: Floor-plan of the Brown Systolic Array.

the board) for the leftmost register bank (part of block 1 in Figure 4).

The B-SYS ALU is loosely based on the OM1 ALU described in Mead and Conway [8]. Logic for calculating arbitrary functions computes the propagate and generate signals which control the Manchester carry chain. The carry chain requires approximately 4 ns to evaluate.

The 38-bit B-SYS instruction word is shown in Figure 5. Eight bits specify the 3:1 function of the result logic block (CR) and four bits specify 2:1 functions for each of the carry logic blocks (CG, CP). Register bank addresses require five bits

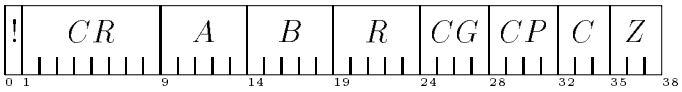

Figure 5: B-SYS Instruction word.

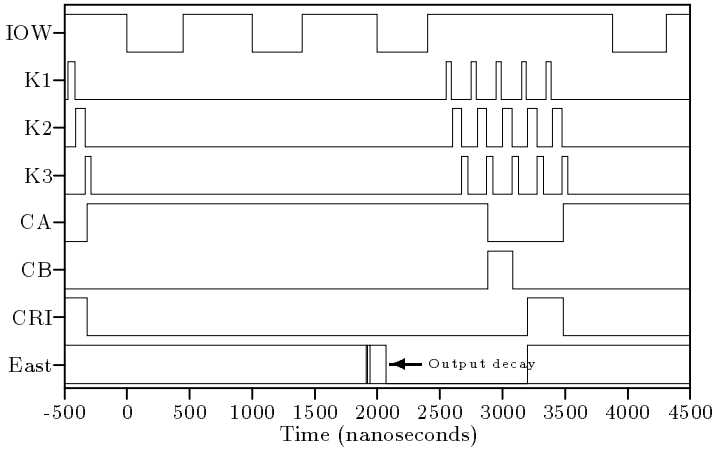

Figure 6: B-SYS clocking for sequence comparison

to specify a register bank (east or west) in addition to the register number (A, B, R), while flag addresses require three bits (C, Z). A single bit indicates whether or not the context flag should be obeyed (!).

B-SYS' 84 pins are grouped into five categories: clocking (4), instruction (38), data (9 for each side, including a control line for masking), diagnostic output (6), and source voltages (12). The remaining 6 pins are unused. B-SYS is controlled with three clocks (K1–K3) and an initialization signal (init). The clocks correspond to decoding the memory address and precharging memory (or evaluating the carry chain), accessing the register bank, and clearing the register selection. Figure 6, a transcription of logic analyzer data, illustrates clocking on the prototype. Lowering the init signal sends the chip through the three phases of reading two operands (CA and CB) and writing the result (CRI). The majority of time is spent copying the instruction to the board, as indicated by the IOW line. The decay of B-SYS output from the previous instruction in 2 ms can also be seen.

Timing simulation produced the conservative estimate that an entire instruction evaluation could be accomplished within 320 ns (20 ns clocks with various times between pulses). The chips were tested with a Hewlett Packard 16500A logic analysis system. The simplicity of the SSR architecture led to a fast testing scheme. Of the twenty-four chips delivered, ten performed satisfactorily at 400 ns per instruction (and could be run at faster speeds), though the complete array is run at a slower speed (Figure 6). The remaining chips suffered assorted fabrication

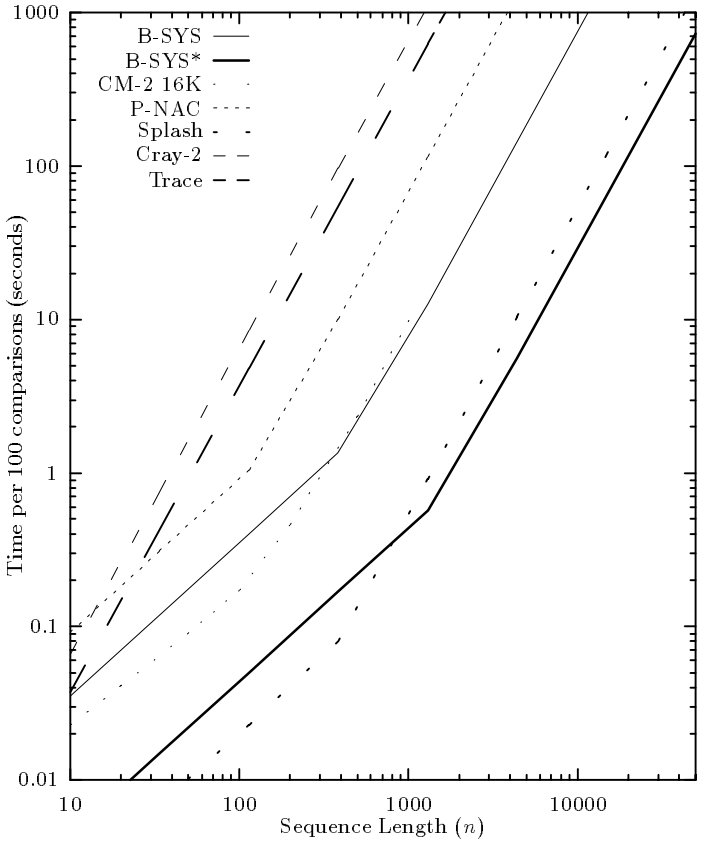

Figure 7: Many-against-one sequence comparison.

defects which were individually pinpointed to specific rows, processing elements, and bits.

A wire-wrap ISA board buffers instructions between the B-SYS chips and the Intel 80386 (25 MHz) host. Each instruction requires three 16-bit ISA I/O bus cycles (Figure 6); this instruction transfer time dominates the prototype's 108 MOPS performance. Simple board modifications could increase the array speed to 200 MOPS, while a truly sophisticated board with its own controller could run the B-SYS array at full speed, increasing performance to 400 MOPS, or perhaps even closer to the chip's maximum of 3.3 MOPS per functional unit (1.8 GOPS for a 470-element array).

B-SYS was evaluated with a one-against-many sequence comparison algorithm appropriate to the Human Genome Project. The B-SYS prototype performs 470-character one-against-many sequence comparison 81 times faster than its 80386 host. Figure 7 compares B-SYS to several systolic co-processors and supercom-

puters. P-NAC, Splash, Cray-2, and Trace timings are extrapolated from experimental 100×100 data [9]. CM-2 timings are from experimental data. P-NAC is a single-purpose nMOS co-processor. Splash is a 2-board linear array of field-programmable gate arrays. B-SYS* is a proposed 32-chip (1504-element) board with its own controller. The Splash programmable hardware system is twice as fast as B-SYS* when $n < 400$, but the higher processor density on the single-board B-SYS* system lets it perform 1.8 times faster than Splash when $n > 1500$. With a redesigned 500-processor B-SYS chip, a 32-chip (16000-element) B-SYS system would perform 2500 times faster than P-NAC and 20 times faster than Splash when $n > 16000$.

Splash is the only programmable competitor to B-SYS* but it does not feature full or simple programmability; even slight algorithmic changes require complete reconfiguration, often lowering the systolic cell density. For example, B-SYS performance will not change for 7-bit ASCII characters, while Splash cell density will drop by a third, moving Splash's bend between $O(n)$ and $O(n^2)$ performance to $n = 250$ and increasing the performance factor difference to 2.7 when $n > 1500$. Other variations, such as the affine cost functions used by computational biologists, will produce even starker contrasts.

Ease of programmability is also a concern. For this problem, P-NAC requires no lines of code, B-SYS requires tens of lines of code, the Connection Machine and other supercomputers require slightly more lines, and the Splash system requires nearly 3000 lines of code. With the development of programming tools for the automatic support of systolic streams, the programming and testing of new B-SYS applications requires little additional time beyond that required for selection of an appropriate systolic algorithm [10].

Conclusions

The Brown Systolic Array features regular interconnections for systolic data flow; SIMD broadcast instructions for control; shared registers for communication and computation; and data streams for the ease of programming. The working 470-processor prototype provides over 100 MOPS of systolic co-processing power.

Programs for several more B-SYS applications have been generated, including data compression, sequence comparison variations, and graph algorithms [10]. The power of B-SYS is its simple and efficient programmability and high concentration of processing elements. A more sophisticated co-processor board board could increase the Brown Systolic Array's performance to a limiting 5 GOPS for a 32-chip board, or even 50 GOPS with a redesigned, 500-element B-SYS chip. The Systolic Shared Register architecture forms a strong basis for the construction of small and powerful systolic co-processors.

References

[1] M. J. Foster and H. T. Kung, "The design of special-purpose VLSI chips," *Computer*, pp. 26–40, Jan. 1980.

[2] S. Borkar *et al.*, "iWarp: An integrated solution to high-speed parallel computing," in *Proc. Supercomputing '88*, pp. 330–339, IEEE, Nov. 1988.

[3] L. W. Tucker and G. G. Robertson, "Architecture and applications of the Connection Machine," *Computer*, pp. 26–38, Aug. 1988.

[4] J. D. Watson, "The Human Genome Project: Past, present, and future," *Science*, pp. 44–48, 6 Apr. 1990.

[5] J. T. Stasko, "Tango: A framework and system for algorithm animation," *Computer*, pp. 27–39, Sept. 1990.

[6] R. N. Mayo *et al.*, "1990 DECWRL/Livermore Magic Release," Research Report 90/7, Digital Western Research Laboratory, Palo Alto, CA, Sept. 1990.

[7] D. Beatty *et al.*, *User's Guide to COSMOS version 1.2.* Carnegie-Mellon University, Oct. 1988.

[8] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems.* Reading, MA: Addison-Wesley, 1980.

[9] M. Gokhale *et al.*, "Building and using a highly parallel programmable logic array," *Computer*, pp. 81–89, Jan. 1991.

[10] R. Hughey, *Programmable Systolic Arrays.* PhD thesis, Department of Computer Science, Brown University, Providence, RI, 1991.