Architecture of the PSC: A Programmable Systolic Chip

Allan L. Fisher, H. T. Kung, and Louis M. Monier
Department of Computer Science, Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Yasunori Dohi*
Department of Computer Engineering, Yokohama National University
Tokiwadai, Hodogaya-ku, Yokohama, 240 Japan

Abstract

In recent years, many systolic algorithms have been proposed as solutions to computationally demanding problems in signal and image processing and other areas. Such algorithms exploit the regularity and parallelism of problems to achieve high performance and low I/O requirements. Since systolic algorithms generally consist of a few types of simple processors, or systolic cells, connected in a regular pattern, they are less expensive to design and implement than more general machines.

This advantage is offset by the fact that a particular systolic system can generally be used only on a narrow set of problems, and thus design cost cannot be amortized over a large number of units. One way to approach this problem is to provide a programmable systolic chip (PSC), many copies of which can be connected and programmed to implement many systolic algorithms.

The systolic environment, by virtue of its emphasis on continuous, regular flow of data and fairly simple per-cell processing, imposes new design requirements for programmable processors which are quite different from those found in a general-purpose system. This paper describes the CMU PSC, a single-chip microprocessor suitable for use in groups of tens or hundreds for the efficient implementation of a broad variety of systolic arrays. The processor has been fabricated in nMOS, and is undergoing testing.

1. Introduction

This paper describes the architecture and applications of a single-chip microprocessor, the CMU *programmable systolic chip* (PSC). Large numbers of these chips are to be used in the implementation of a broad spectrum of *systolic algorithms*[11], which typically achieve high performance by exploiting regularity and parallelism in the computations which they perform. The PSC is tailored to the I/O and computational requirements of this family of algorithms.

The PSC project, initiated in October of 1981, had several motivations. Among these are:

- *Exploration of a new region of the computer design space:* The I/O and computation demands placed on a building-block processor for systolic arrays are quite different from those placed on conventional processors. Together with the emerging feasibility of sizable memories and significant computational power on single chips, these factors lead to new resource tradeoffs.
- *Programmable implementation of systolic arrays:* Because they consist of regular layouts of simple cells, systolic arrays are generally easier to design and implement than more general-purpose machines. However, since a particular systolic system is limited to a narrow set of problems, its development cost cannot usually be shared among many units. It is therefore desirable to have a programmable building-block, as shown in Figure 1, which can be used to construct many different types of arrays, reducing development costs to microcoding and circuit board layout. The PSC, in its current form, represents an efficient and flexible means of implementing a class of high-performance special-purpose devices. More specialized PSCs (e.g., tailored for signal processing or data processing) could be even more efficient. In addition to aiding in the actual application of systolic arrays, PSCs will also be useful in design of and experimentation with new systolic algorithms.

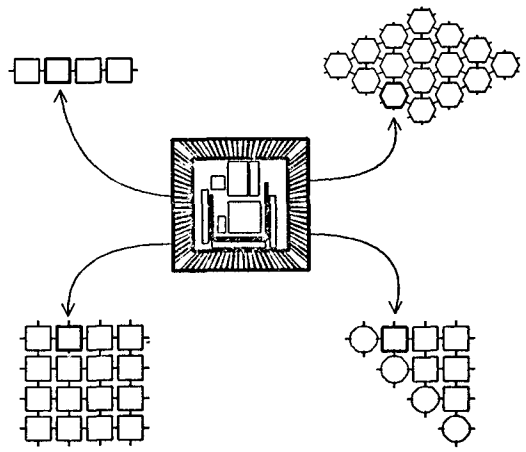

Figure 1: PSC: A building-block chip for systolic arrays.

- *Rapid implementation of novel architectures:* The development of the PSC design, from the initial idea to a silicon layout, took slightly less than a year in a university environment. The design has been simulated at the register transfer and layout levels. Demonstration arrays should be operational, allowing for two silicon runs, in an additional eight months. This project demonstrates the exciting opportunities presented by custom LSI and VLSI—fairly complex experimental architectures can actually be built and tested with a reasonable amount of time and effort. The process of

*Work performed while visiting CMU.

designing the PSC has also helped in evaluation of the design tools used at CMU, and has suggested directions for improvement.

Section 2 of this paper discusses the demands placed on processors to be used for implementing systolic arrays and the design issues which arise from these demands. Section 3 describes the architecture of the PSC. Sections 4 and 5 discuss the applications of the chip and compare it with alternative means of implementation. Section 6 discusses possible refinements and specializations of the PSC concept. Details of the implementation of the chip are available in a companion paper[6].

2. Design Goals and Issues

The primary goal of the PSC architecture was to support reasonably efficient implementation of a very broad variety of systolic algorithms. To this end, a set of *target applications* was chosen. An obvious candidate was the field of signal and image processing; problems in this area often demand real-time solution and many systolic algorithms for such problems have already been designed[9, 12]. Another choice was error detection and correction coding, Reed-Solomon coding[14, 16] in particular. In addition, sort/merge of large files was chosen as a representative of data-processing applications.

Given these goals, the following design issues were considered:

- Locality and flexibility of control: For some systolic arrays, including many signal processing and matrix arithmetic algorithms, simple global control would be sufficient. However, many other algorithms require different actions in different phases (e. g., loading of coefficients), as well as data-dependent actions within each systolic cycle.
- Chip count: Keeping a systolic cell on a single chip has two advantages. First, it allows the functional blocks of the processor to operate together without paying the time and pinout penalty of off-chip communication. Second, it allows systolic arrays to be constructed with small chip count. The possibilities for putting *more* than one cell on a chip seem limited for the near future; given fairly complex processors, advances in miniaturization will probably be more usefully spent on increased word size and functional capability. This situation obviates the need for on-chip configurability—system configuration (or reconfiguration) is done at the board level.
- Primitive operations: The primitive arithmetic, logical and control operations which a processor can perform are critical to its efficiency. In particular, fast multiplication is needed for the effective implementation of most signal and image processing algorithms. Provisions for multiple-precision arithmetic can extend a processor's utility.
- Intercell communication: A principal feature of systolic arrays is the continuous flow of data between cells. Efficient implementation of such arrays requires wide I/O ports and data paths. Provision must also be made for the transmission of pipelined, or *systolic control* signals.
- Internal parallelism: Partition of a processor's function into units which can operate in parallel enhances performance.
- Control structure: Horizontally microprogrammed control structures provide flexibility in programming new applications and promote parallelism within a processor. The usual drawback of horizontally microprogrammed architectures, difficulty in programming, is eased by the fact that systolic algorithms have very simple basic cycles and hence short microprograms.
- Word size: Individually programmed processors are subject to a tradeoff in word size: small word sizes lead to an imbalance between the hardware devoted to control and that devoted to data paths, and large words lead to large chips with large pinout and low yield.

3. PSC: A Programmable Systolic Chip

Based on the considerations discussed above, we have designed and laid out in nMOS (using Mead-Conway design rules[15]) the PSC, a single-chip programmable systolic processor. Since this processor is an experimental prototype, rather than a production version, we set the goal of keeping the design simple and general, albeit at the cost of pincount and ultimate performance. Nonetheless, as discussed in Section 5, the chip as designed already represents a cost-effective means of implementation for many systolic algorithms.

Processor structure

The processor consists of a collection of communicating functional units, all of which may operate in parallel. This collection is made up of a microcode RAM and microsequencer, a register file, an ALU, a multiplier-accumulator (MAC), and three input and three output ports. As schematized in Figure 2, data communication among the units takes place on three independent buses; control and status lines are separate. Each bus can be written by one of eight sources and be read by any of ten destinations. This organization supports a significant amount of on-chip parallelism; a multiplication with accumulation, an addition, a memory fetch, and interchip I/O can take place concurrently in one instruction cycle. The parallelism-limiting effect of having only three buses, as opposed to eight, is alleviated by the fact that a value on a bus is often used more than once, and by the ability of each of the functional units to hold its inputs over more than one cycle.

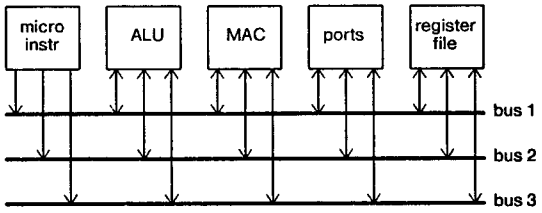

Figure 2: Bus structure of the PSC

Word size

In order to keep the chip small and hence keep pinout and yields reasonable, a modest word size of eight bits for arithmetic was chosen. Most data paths in the chip, however, are nine bits wide. The ninth bit can be used to tag data, to store control information, or as the most significant bit of a number modulo 257 for coding applications. In order to support arithmetic on larger numbers, facilities are also provided for multiple-precision computation: the multiplier and ALU have provisions for setting a carry in to the low-order bit, and the ALU can cycle its carry out into its carry in for the next operation.

I/O ports

Each of three input and three output ports contains nine bits, eight of which are data, with the ninth available for data or control. The ninth bit of each input port is available as a condition code for microprogram branching. This is often used as a tag bit for variable-length data or as a systolic control bit for loading and unloading of stored values. The ninth bit of each output port may be set as a literal, as the most significant bit of a nine-bit bus value, or as the most recent value of an incoming ninth bit.

In order to keep the clock period short, interchip communication is pipelined (or overlapped) with instruction execution. Thus a value which is computed in a given cycle can be used on the same chip in the next cycle, but not until the cycle after that on a neighboring chip. The effect on system performance is to increase latency (in clock cycles, though not necessarily in real time) but to reduce the time needed for each pipeline stage.

Control part

The microprogram memory consists of 64 60-bit words of dynamic (for circuit density) RAM. The 60 bits in a horizontal microinstruction are divided as follows:

- 9 bits to set bus contents.
- 20 bits for control of functional unit input registers.

- 9 bits for off-chip control registers.
- 10 bits for control of ALU, multiplier, and register file.
- 3 bits to control program branching.
- 9 bits to provide a literal branch address, literal data to the bus, or condition code selection for branching.

The microsequencer is able to fetch instructions in sequence, branch to a literal address, branch to one of four locations depending on any two of twelve condition code bits, and push and pop addresses to and from a subroutine stack.

Arithmetic

The ALU performs standard arithmetic and logical operations: addition, subtraction, logical AND, etc.. It also allows its carry in bit to be set, either as part of its opcode or as its previous carry out, as an aid to multiple-precision computation. A typical complement of condition code outputs is supplied to the microsequencer. Arithmetic is performed in twos complement notation, except that an "unsigned subtraction" operation is available for eight-bit character comparisons and normalization of numbers modulo 257.

The multiplier-accumulator multiplies two eight-bit numbers and adds them to a 16-bit accumulator, a third eight bit input, or zero to produce a sixteen bit output. The numbers used may be signed (in twos complement) or unsigned, independently.

Register file

The register file consists of 64 nine-bit words of dynamic RAM. One word may be read or written in each cycle. In order to remove the RAM's delay from the chip's critical path, the register file is pipelined one cycle behind the rest of the chip; thus an address must be supplied one cycle before the addressed value is needed. This does not appear to pose a performance penalty, since register file accesses seem in general to be uniform and predictable.

Microcode loading

All microcode is loaded through a shift register, which can also be used for functional testing of the chip.

In summary, then, the design of the PSC responds to the issues discussed in the previous section as follows:

- Locality and flexibility of control: The microprogrammed control described provides a very high degree of flexibility.
- Chip count: The processor is designed and implemented as a single chip.
- Primitive operations: The processor has hardware which allow it to perform multiplication, other arithmetic operations (including multiple precision), and branching and subroutine control very efficiently.
- Intercell communication: The three sets of I/O ports provide high-bandwidth data flow between neighboring cells, and efficiently support the passing of systolic control bits.
- Internal parallelism: The ports, ALU, multiplier, program memory and data memory all function simultaneously. The parallel bus structure allows a number of disjoint computations to proceed simultaneously.
- Microprogrammed control: The horizontal organization of the microcode allows all of the functional units to be utilized in a single cycle.
- Word size: The eight/nine bit format chosen represents a reasonable compromise in terms of size, yield and utility. As section 4 shows, eight-bit PSCs can already be very useful. Longer words will be used for future PSCs, especially as silicon feature sizes shrink, in order to make them useful to a broader class of applications.

As we discuss in Section 5, existing microprocessors fall short of meeting these criteria in several ways.

4. Applications

The PSC can be used as the basic systolic cell for many systolic systems in many application areas. In this section we give a flavor of how it is applied in two target applications, and discuss its cost-effectiveness with respect to these applications. These and other applications have been implemented in microcode; this code has been simulated by ISPS[1] simulators and used to evaluate architectural specifications of the chip. We estimate that a commercial nMOS implementation of the PSC could operate with a cycle time of 200 ns; our first implementation should run within a factor of two of this speed. We shall assume the 200 ns period in our performance estimates.

Note that for each of these applications it is always possible to develop a more special-purpose, and thus more efficient, chip than the PSC. The point of the PSC project, however, is to show that with a carefully chosen architecture, a rather general-purpose chip such as the PSC can still achieve very competitive performance results for each individual problem in a wide variety of application areas.

Because of space limitation, in this section we describe only two applications: a digital filtering algorithm is described in some detail, and a system for Reed-Solomon coding is described at a higher level of abstraction. A companion paper[6] briefly describes sort/merge and file fingerprinting applications. For more information and for other examples, the reader is referred to a forthcoming paper on applications.

4.1. Digital Filtering in Signal and Image Processing

Many digital signal and image processing applications require high-speed filtering capabilities. Mathematically, a filtering problem with $h+k$ taps is defined as follows:

given the weights $\{w_1,w_2,\ldots,w_h\}$, $\{r_1,r_2,\ldots,r_k\}$, the initial values $\{y_0, y_{-1},\ldots,y_{-k+1}\}$ and the input data $\{x_1,\ldots,x_n\}$,

compute the output sequence $\{y_1,y_2,\ldots,y_{n+1-h}\}$ defined by

$$y_i=\sum_{j=1}^{h} w_j x_{i+j-1}+\sum_{j=1}^{k} r_j y_{i-j}.$$

If the $\{r_i\}$ are all zero, then the problem is called a finite impulse response (FIR) problem, and otherwise an infinite impulse response (IIR) problem. It is known that both types of filtering can be performed by systolic arrays[8, 9, 11]. Figure 3 illustrates a systolic array for FIR filtering for $h=3$.

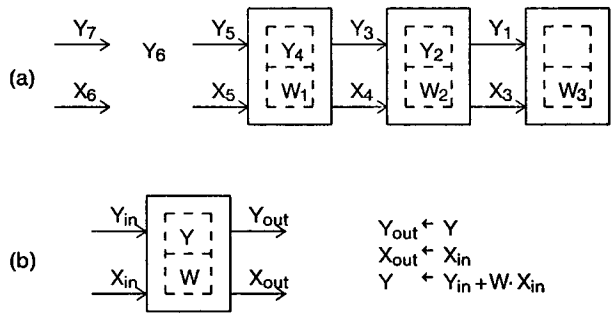

Figure 3: (a) Systolic FIR filtering array and (b) its cell definition.

Based on this scheme, digital filters (FIR or IIR) with eight-bit data and weights and m taps can be computed with a linear systolic array composed of m PSCs, taking one sample each 200 ns. Thus with 40 PSCs, a 40 tap filter can be computed at a rate of 400 million operations per second (MOPS), counting each inner product step (eight-bit multiply, 16-bit add) as two operations. This is equivalent to 600 MOPS for pure eight-bit arithmetic.

We now describe a particular example in which the PSC is programmed as a systolic cell for a systolic FIR filtering array. The program takes only one instruction to implement the operations depicted in 3(b). After an initialization phase in which the weights are loaded, the inner loop of the algorithm uses only one PSC microinstruction, coded as follows:

```
Bus1=Sda, Bus2=Sdb, Bus3=Lo,
SdaOut=Val3, SdbOut=Val2,
MacX=Hold, MacY=Val2, MacZ=Val1, MacOp=AddZ,
Jump=OnCC0, CC0=Sca, ScaOut=Pass.
```

The lines of this microinstruction have the following effects, all in a single cycle:

1. Bus 1 carries Y_{in}, read from systolic data port A, bus 2 carries X_{in}, read from port B, and bus 3 carries Y_{out} from the previous operation, available as the output of the MAC.
2. Output port A receives Y_{out} from the previous operation (the value on bus 3), and port B receives $X_{out}=X_{in}$.
3. The MAC holds the cell's weight in its x register, sets its y register to X_{in} (the value on bus 2), and sets its z register to Y_{in} (the value on bus 1). It then computes $x \cdot y + z$, or $W \cdot X_{in} + Y_{in}$. This value will be sent to output port A during the next cycle.
4. The instruction loops in place until systolic control signal A arrives, meaning it is time to reinitialize. Control then passes to the next intruction, and the control bit is sent on to the neighboring cell.

For applications requiring more accuracy, a filter with 16-bit data and eight-bit coefficients and m taps can be computed with m PSCs, taking one sample each 1.2 μs. Thus with 40 PSC chips, a 40 tap filter can be computed at a rate of 67 MOPS, counting each inner product step as two operations. This is equivalent to 200 MOPS for eight-bit arithmetic.

If higher performance is needed, one-dimensional systolic filtering arrays can be stacked together to form two-dimensional arrays. In this case the system host must have sufficiently high I/O bandwidth that multiple data streams can be fed into a two-dimensional systolic array simultaneously. Similar comments apply to other one-dimensional systolic arrays discussed below.

4.2. Error-Correcting Codes

Among various error-correcting codes, Reed-Solomon codes are most widely used today for deep space communications, where burst errors occur frequently[14, 16]. A popular Reed-Solomon code, which for example has been adopted by the European Space Agency, is a scheme in which a codeword consists of 224 message symbols followed by 32 parity symbols. This code can correct up to 16 symbol errors per codeword through a decoding process.

Each symbol is defined over a finite field of 257 elements, denoted by $GF(257)$, and thus is encoded with 9 bits. Any number less than 256 is represented as usual by 8 bits, and the number 256(=-1) by 9 bits as 100000000_2. We call this representation "normalized form". Arithmetic is performed on numbers in normalized form, and gives a result in normalized form. The basic pattern is to test if one of the operands is equal to 256—in which case a special treatment is applied—then operate on 8 bits and normalize the result.

Before a message is transmitted, it is first encoded. It is well known that encoding a message, i.e., obtaining parity symbols from the given message symbols, is equivalent to a polynomial division. It can therefore be carried out with the systolic division array described in[10]. Since in this case the divisor—the *generator polynomial* in the terminology of error-correcting codes—is monic, no cell in the systolic array needs to perform a numerical division. Each cell is basically the same as used in the systolic filtering array discussed above, except that integer arithmetic modulo 257 is used.

Decoding, which is much more complex than encoding, is usually done in four steps. We will not describe the steps in detail here; the reader is referred to the texts on error-correcting codes cited earlier. In the following we simply point out that each step corresponds to some polynomial computation over the finite field $GF(257)$ that can be effectively carried out by systolic arrays.

- *Syndrome computation* is to compute the syndrome polynomial $S(x)$ of degree 31. The problem is equivalent to that of evaluating a polynomial of degree 255 at 32 points. Using Horner's rule, this can be done by a systolic array where each cell holds one of the 32 points and accumulates results as the coefficients of the polynomial flow through the array[9].
- *Solution of the key equation* is to find two polynomials, the error locator polynomial $\omega(x)$ and error evaluator polynomial $\sigma(x)$, with $\deg\omega < 16$ and $\deg\sigma \le 16$, such that the key equation,

$$\omega(x) \equiv \sigma(x) S(x) \ [\bmod x^{32}],$$

  is satisfied. This problem can be solved by a systolic array for computing extended greatest common divisors[3].
- *Error location* is to find the roots of $\sigma(x)=0$, which will identify the locations of errors in the received message. Finding the roots is most efficiently done by simply evaluating $\sigma(x)$ at every point in $GF(257)$, since the size of the field is small. Again as in the syndrome computation, based on Horner's rule we can use a systolic array for carrying out the polynomial evaluation. Now since the degree of the polynomial is small and the number of the points where the polynomial is to be evaluated is large, we use a "dual" systolic design where each cell holds one coefficient of the polynomial and the points flow along the array.
- *Error evaluation* is to evaluate the amplitude of each error found in the previous step, by evaluating $\omega(x)/\sigma'(x)$ at the roots of $\sigma(x)=0$. This again calls for polynomial evaluation.

Figure 4 illustrates that all the steps mentioned above can be implemented with appropriate systolic arrays made up of the same PSCs, with different microcode for each array. We estimate that by using a linear array of 112 PSCs, Reed-Solomon decoding can be performed with a throughput of 8 million bits per second.

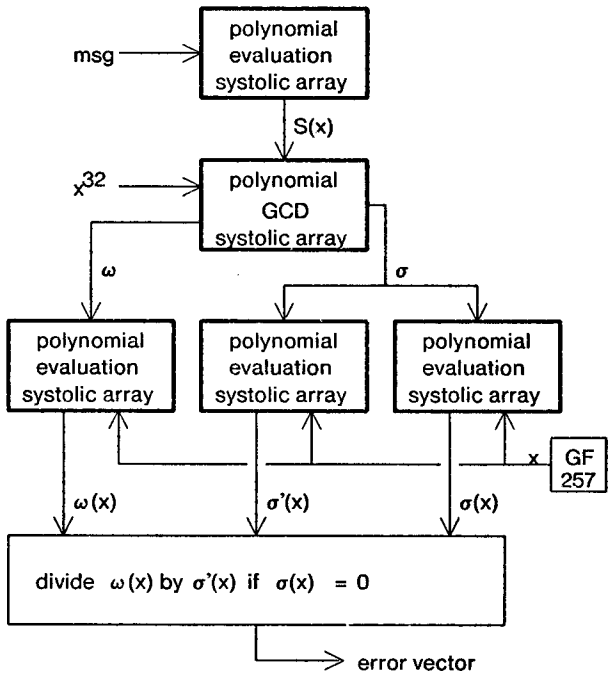

Figure 4: Systolic decoder for Reed-Solomon code

Encoding is much easier; it requires only about 16 PSCs to achieve the same throughput. As far as we know, the fastest existing Reed-Solomon decoder with the same mathematical characteristics uses about 500 chips but achieves a throughput of no more than 1 million bits per second.

5. Implementation Alternatives

The PSC represents only one possible means of implementation of systolic algorithms. Alternatives include board-level implementation from existing parts, full custom LSI implementation, and implementation using existing microprocessors. Existing systolic array implementations span this spectrum. Early test implementations[7, 13] were full custom single-purpose devices, as is the GEC correlator chip[5]. In an intermediate range of flexibility are the ESL systolic processor[2, 19] and a forthcoming ESL systolic chipset for floating-point matrix computations, both of which are programmable for a range of signal processing tasks. At the very general end of the spectrum is the Naval Ocean Systems Center systolic array testbed[4, 17, 18], which is assembled out of general-purpose microprocessors and can use both one- and two-dimensional array communication structures.

Board-level and full custom LSI implementations may be preferable to the PSC approach where performance requirements are very stringent. The disadvantages of these approaches are mainly in design cost (especially for custom LSI) and, for board-level implementations, costs of packaging, power dissipation, and physical size.

The alternative approach closest in spirit to the use of PSCs is the use of existing microprocessors. Conventional microprocessors cover such a wide range of applications that it may seem a good idea to use them for building systolic arrays. In fact this turns out to be false, since microprocessors would perform very poorly in this context—in many cases, the PSC works an order of magnitude faster. The reasons for this failure reside in the differences between the PSC and conventional μPs: I/O facilities, on-chip program memory, internal parallelism, parallel multiplier and systolic control bits. No commercially available μP that we are aware of combines all of the above properties. Instead, we can find a large variety of machines, all called "microprocessors," that have only one or two of those features.

If we try to implement systolic algorithms using an existing microprocessor, we come up with many possibilities:

- *Most common microprocessors*, either on 8 bits (Z80, 6800, 8080 families) or larger (68000, 16000, iAPX 432 families), have no internal program or scratch pad RAM (usually a few registers), no hardware multiplier, and at most two ports: an address port and a data port. In order to be used in systolic arrays, each processor would need external memory chips, I/O devices and a few components for inter-chip communication. Such devices could be built, but at high cost in chip count, power dissipation, and space. Also, conventional microprocessors provide large instruction sets and complex addressing schemes, but cannot be programmed at the microcode level; as a result, even simple operations take several cycles. Finally, I/O is a serious bottleneck since the same ports are used for fetching operands and instructions and for inter-chip communication.
- *Signal processing microprocessors* (NEC 7720, TMS320, Hitachi HSP) usually have an on-chip program memory in ROM, a small data memory, and (for the most recent ones) a parallel multiplier. They also show some amount of internal parallelism since they can perform a multiplication-accumulation and increment an address register at the same time. However, their structure is often too specialized to use them in applications other than signal processing, and the I/O problem is also present; none have the off-chip bandwidth necessary for systolic array communication.
- *Single-chip micro-computers* (8020, 8749) are, in some respects, superficially similar to the PSC. They have an on-chip ROM program memory, a small RAM for data, and up to three bidirectional ports (used serially, however, not simultaneously). They can be used with few external components, as opposed to microprocessors that need an external memory and I/O devices in order to function. However, they are serial machines with simple instruction sets (the instruction "multiply" is absent), and suffer from the same I/O bottleneck as traditional microprocessors.
- *Bit-slice microprocessors* form a last family. Of course, one could make a PSC out of several of them and many external parts, and this could have been an alternative to the implementation of a chip. However, the result would be uneconomical in most applications because of the large chip count and accompanying size, fabrication, and power costs.

As an example of the effects of serial execution and data access, consider the filtering example of Figure 3. A microprocessor equipped with a multiplier and the ability to perform an I/O operation to either of its neighbors in a single cycle would need at least 10 instruction cycles to perform this inner loop (4 for I/O and 6 for branching, arithmetic, and data movement). Counting the overhead of instruction fetching and decoding, it is extremely unlikely that such a processor, implemented in similar technology, would have a smaller instruction cycle time than the PSC; hence the PSC's organization allows it to operate at least ten times faster in this case. For Reed-Solomon encoding, a standard μP would execute at least four times as many instructions as the PSC. Most systolic algorithms seem to produce instruction ratios in the range between 4:1 and 10:1.

We see from these considerations that current microprocessor architectures are not well-suited to the construction of systolic arrays. The key points, again, that favor the PSC structure are:

- The large number of ports (including systolic control) used in parallel.
- Internal program and data memory: I/O bandwidth is fully available for transfer of data between chips. In our examples, the ratio (number of input + output)/(number of instructions) is high, often larger than 1. Also, typical programs are short and can fit into a small control memory.
- Horizontal microprogramming: large microinstructions provide effective internal parallelism. This is possible only because of the rich interconnection pattern (3 internal global buses).
- Systolic control: this is indispensable for systolic algorithms, and is costly in instruction cycles and I/O bandwidth for conventional μPs.
- Multiplier: this is critical for high performance in numerical applications.

6. Concluding Remarks

This paper reports an initial investigation of the architecture space of *programmable* systolic arrays and processors, which seem to represent an important point on the generality-performance-cost tradeoff curve. The PSC prototype will be extremely useful in further research in this area, by supporting detailed experimental applications studies which are underway.

One promising area of further investigation is the design of more specialized PSCs. The present design is not, for example, optimally suited to the sorting problem—the large multiplier sits idle, and off-chip memory addressing is somewhat cumbersome. Thus it might be reasonable to design a small family of PSCs, each well-suited to a particular range of applications.

Another logical step is the design of more powerful PSCs. Larger word size and hardware floating point capability would greatly extend the PSC's utility in signal and image processing. Assuming that a 16-bit multiplication takes twice as long as an eight-bit multiplication (true for most bit-parallel multipliers), it is possible to build a 16-bit PSC with the same I/O-computation balance as the current system by multiplexing I/O through eight-bit ports. This alleviates the pinout problem often associated with larger word sizes.

Yet another task is the refinement of the existing PSC design. The current multiplier uses the naive bit-parallel design, and its layout could be improved. Since the architecture was settled, several possible enhancements have suggested themselves. One is the addition of small FIFOs at the I/O ports, which would serve to reduce bus usage within chips for data that need to be delayed. Another is the augmentation of the register file to two or more ports; some multiple-precision computations seem to be memory-bound. It might also be possible to provide

more effective bus bandwidth by providing specialized buses for high-utilization paths. All of these issues will need further detailed applications studies to resolve them.

The programming task presents a higher-level problem. As for most horizontally microprogrammed machines, writing optimal or near-optimal programs for the PSC is a difficult exercise. We hope that microcode compilers for the PSC will be easier to produce than for some other machines, since they will be able to take advantage of the brevity and simplicity of typical systolic cells' programs.

Finally, there remains the possibility of different organizations for programmable systolic arrays. Despite the fact that it is a building-block for highly specialized, non-conventional systolic architectures, the design of the PSC itself is based mostly on standard principles. It is possible that alternative structures, such as bit-serial arrays, could also serve as building-blocks for many systolic algorithms. This avenue of research deserves exploration.

Acknowledgments

The PSC project has profited greatly from the assistance of Monica Lam, Onat Menzilcioglu, Hank Walker, and John Zsarnay in particular, and the VLSI design community at CMU in general. The dynamic RAM used for the control store and register file was designed by Hank Walker.

This research was supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. A.L. Fisher was supported in part by an NSF graduate fellowship and in part by an IBM graduate fellowship.

References

1. Barbacci, M.R., "Instruction Set Processor Specifications (ISPS): The Notation and Its Application," *IEEE Transactions on Computers*, Vol. C-30, No. 1, January 1981, pp. 24-40.

2. Blackmer, J., P.Kuekes and Frank, G., "A 200 MOPS Systolic Processor," *Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV*, The Society of Photo-optical Instrumentation Engineers, August 1981.

3. Brent, R.P. and Kung, H.T., "Systolic VLSI Arrays for Polynomial GCD Computation," Tech. report, Carnegie-Mellon University, Computer Science Department, May 1982.

4. Bromley, K., Symanski, J.J., Speiser, J.M., and Whitehouse, H.J., "Systolic Array Processor Developments," *VLSI Systems and Computations*, Kung, H.T., Sproull, R.F., and Steele, G.L., Jr., eds., Computer Science Press, Inc., Computer Science Department, Carnegie-Mellon University, October 1981, pp. 273-284.

5. Corry, A.and Patel, K., "A CMOS/SOS VLSI Correlator," *Proceedings of 1983 International Symposium on VLSI Technology, Systems and Applications*, 1983.

6. Fisher, A.L., Kung, H.T., Monier, L.M., Walker, H.and Dohi, Y., "Design of the PSC: A Programmable Systolic Chip," *Proceedings of the Third Caltech Conference on VLSI*, California Institute of Technology, March 1983.

7. Foster, M.J. and Kung, H.T., "The Design of Special-Purpose VLSI Chips," *Computer*, Vol. 13, No. 1, January 1980, pp. 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. A preliminary version of the paper, entitled "Design of Special-Purpose VLSI Chips: Example and Opinions," also appears in *Proceedings of the 7th International Symposium on Computer Architecture*, pp. 300-307, La Baule, France, May 1980

8. Kung, H.T., "Let's Design Algorithms for VLSI Systems," *Proceedings of Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, California Institute of Technology, January 1979, pp. 65-90, Also available as a CMU Computer Science Department technical report, September 1979.

9. Kung, H.T., "Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI," *Proceedings of the SPIE, Vol. 241, Real-Time Signal Processing III*, The Society of Photo-Optical Instrumentation Engineers, July 1980, pp. 76-84.

10. Kung, H.T., "Use of VLSI in Algebraic Computation: Some Suggestions," *Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*, Wang, P.S., ed., ACM SIGSAM, August 1981, pp. 218-222.

11. Kung, H.T., "Why Systolic Architectures?," *Computer Magazine*, Vol. 15, No. 1, January 1982, pp. 37-46.

12. Kung, H.T. and Leiserson, C.E., "Systolic Arrays (for VLSI)," *Sparse Matrix Proceedings 1978*, Duff, I.S. and Stewart, G.W., eds., Society for Industrial and Applied Mathematics, 1979, pp. 256-282, A slightly different version appears in *Introduction to VLSI Systems* by C.A. Mead and L.A. Conway, Addison-Wesley, 1980, Section 8.3.

13. Kung, H.T. and Song, S.W., "A Systolic 2-D Convolution Chip," *Multicomputers and Image Processing: Algorithms and Programs*, Preston, K., Jr. and Uhr, L., ed., Academic Press, 1982, pp. 373-384, An extended abstract appears in *Proceedings of 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, November 11-13, 1981, pp. 159-160

14. MacWilliams, F.J. and Sloane, N.J.A., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, Holland, 1977.

15. Mead, C.A. and Conway, L.A., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachusetts, 1980.

16. Peterson, W.W. and Weldon, E.J., Jr., *Error-Correcting Codes*, MIT Press, Cambridge, Massachusetts, 1972.

17. Symanski, J.J., "A Systolic Array Processor Implementation," *Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV*, The Society of Photo-Optical Instrumentation, August 1981.

18. Symanski, J.J., "Progress on a Systolic Processor Implementation," *Proceedings of SPIE Symposium, Vol. 341, Real-Time Signal Processing V*, The Society of Photo-Optical Instrumentation, May 1982, pp. 2-7.

19. Yen, D.W.L. and Kulkarni, A.V., "Systolic Processing and an Implementation for Signal and Image Processing," *IEEE Transactions on Computers*, Vol. C-31, No. 10, October 1982, pp. 1000-1009.