Chapter 1

Reconfigurable Bit-Parallel Architecture

This section describes the basic idea of RBP, provides an argument of its advantage over a bit-serial organization, describes the Abacus PE architecture, and some RBP arithmetic implementations.

1.0.1 The Basic Idea of RBP

The basic idea of RBP is the concatenation of several processing elements into a processing site. The advantages of the technique can be viewed from several angles. First, if an algorithm requires many data bits per pixel, then the silicon area of the ALU is small compared with total data area, and therefore adding a few more ALUs will improve performance linearly, with a tiny increase in total area. This interpretation assumes that all data is kept on chip and physically near the ALU.

Alternately, RBP may be viewed as a method of adding the appropriate amount of registers to a processing site to minimize the frequency of expensive off-chip accesses. The

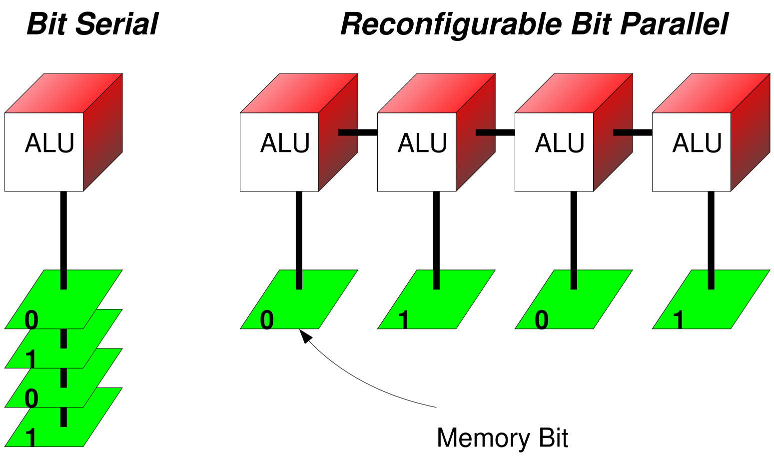

Figure 1-1: A bit-serial organization compared to a reconfigurable bit-parallel one. Each of the RBP processors is also capable of acting as a bit-serial unit, if appropriate.

improvement in performance is due mostly to decreased stall time rather than additional ALU bits.

1.0.2 RBP Performance Model

This section quantifies the regime in which the RBP organization is superior to a bit-serial one. We assume that a processing site, be it a bit-serial PE or a group of RBP PEs, requires a certain amount of on-chip storage to operate without frequent off-chip memory accesses. If this requirement is not met, performance will degrade catastrophically due to the limited off-chip bandwidth.

In the following discussion, let B be the number of bits in a data word; W, the number of words in the register file; V, the area of the non-memory overhead (ALU and network) component of the PE; and S, the number of cycles required by the bit parallel operation. Area measurements are normalized to the size of one memory bit. A bit-serial organization requires B cycles to step through each bit of a data word and occupies $V + BW$ area. An RBP PS requires S cycles and occupies $B(V + W)$ area. The silicon efficiency can be expressed as performance per unit area, or simply as $1/AT$. The ratio of efficiency, bit-parallel to bit-serial, is:

$$R = \frac{V + BW}{S(V + W)}$$

For the vision algorithms we are interested in, computations on 8-bit pixels are done with 16 bits to preserve fractions, and as many as 64 words are frequently accessed. In our implementation of the PE, V was approximately 50. Our algorithms have S ranging from 3 to 6, averaging around 4. A graph of R versus W, for relevant values of B, S and V is shown in Figure 1-2.

The important conclusion is that not only does the RBP approach result in faster (lower-latency) computation, but that it is also more efficient in terms of computational power per silicon area. Of course, this can be seen as arguing for hardware bit-parallel.

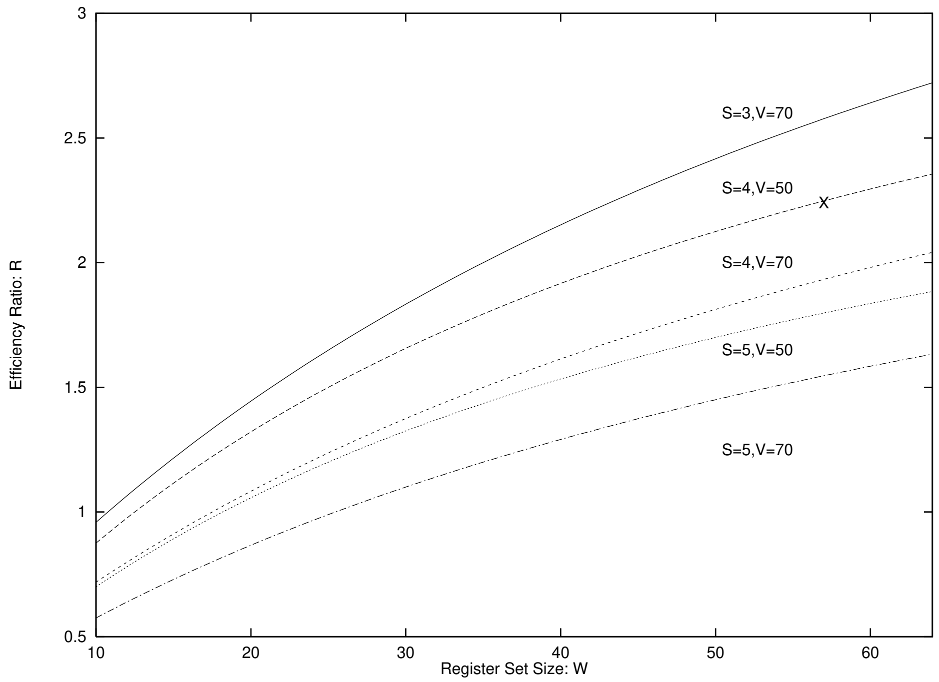

Figure 1-2: Efficiency ratio R vs register file size W, for various values of non-memory overhead V and algorithmic inefficiency S. The 'X' indicates the design point of our implementation.

1.0.3 Abacus PE Architecture

The Abacus processing element (PE) consists of 64 1-bit registers organized into two banks. There are two 3-input ALUs, each of which takes two inputs from its memory bank and one input from the other bank. The four available data bits allow complex boolean functions, and two result bits can be written in each cycle, one bit to each bank.

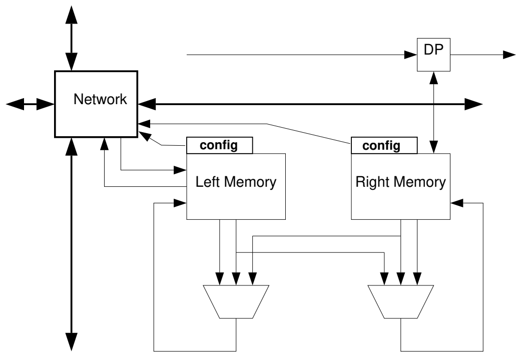

Figure 1-3: Processing Element Block Diagram.

A PE also has a 1-bit network interface and background I/O interface. Seven of the registers are used for control, leaving 57 general-purpose registers.

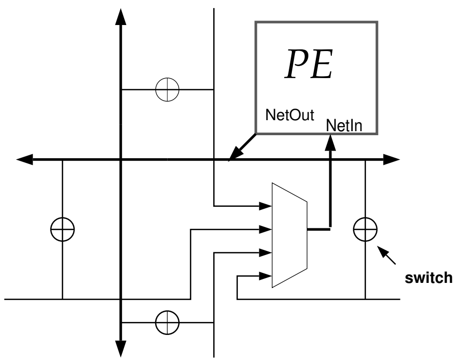

Figure 1-4: Network

Activity Register Conditional execution of an `if (cond) then ... else ...` sequence on SIMD arrays operates by disabling those PEs for which `cond` is true, executing the true branch, disabling the other set of PEs, and executing the false branch. One of the Abacus PE registers serves as the `activity` bit which, when cleared, disables computation by inhibiting the write of the result to the destination register.

The ability of the ALU to serve as a multiplexer greatly reduces the use of the activity register. For example, the sequence `if` A `then` $B \Leftarrow C$ incurs two cycles of overhead if implemented with the activity register. Alternately, the operation can be expressed as $B \Leftarrow$ `mux(`A`,`$C|B$`)`, eliminating all overhead.

Network The reconfiguration network serves for both bit-slice interconnection and inter-word communication. The network is a wired-OR reconfigurable mesh. As in a conventional mesh topology, each PE listens to the output node of one its four mesh neighbors. Each PE is provided with a configuration register that specifies which neighbor to listen to. Unlike a mesh, each PE can connect its output node to the selected neighbor's output node. Shorted nodes behave as a wired-OR bus: if any PEs in a connected chain send a 1, all PEs listening to the chain receive a 1.

Background I/O Each PE contains a data plane (DP) register, used for background off-chip data transfers. The DP registers of PEs in a column of the array form a 32-bit shift register. At the edge of the PE array, the 32 shift registers are connected to an off-chip memory port. These registers can be shifted without interfering with PE computation. Although the DP register is not used in arithmetic operations it is essential in hiding the latency of external memory accesses.

1.0.4 RBP Algorithms

RBP arithmetic algorithms resemble circuits more than bit-serial programs, since they are unrolled in both space and time. During each time step, each PE emulates part of the logic comprising an arithmetic circuit, reconfiguring connections if necessary. In the examples of this section, PSs are shown organized as a line in order to resemble a circuit. In reality, clusters are often arranged as squares or rectangles (a square organization minimizes the average inter-cluster communication distance), and topologically form a ring. In arithmetic circuits, boundary logic cells such as the LSB and MSB are configured differently from the middle cells. Similarly, PEs corresponding to cluster edges are marked as MSB or LSB. In addition, each PE is labeled with its bit position within the cluster. The LSB/MSB bit can be computed from this position, as can network configuration bits.

Data Representation RBP architectures can use a wide variety of number representations. The usual representation is the conventional binary one. In the case of the accumulate operation, the redundant carry-save representation is preferred, as carry computation can be postponed until a non-redundant operation is required. The MGAP designers are proponents of the redundant radix-4 representation where each processor stores a digit in the range of $-3..3$. This is not a very memory-efficient representation, as it requires five bits to store a two bit number.

Logical Shift. The simplest RBP arithmetic operation is the shift. As shown in Figure 1-5, each PE simply replaces its bit with its neighbor's. For logical shifts, the MSB or LSB is cleared; for arithmetic shifts the MSB is retained.

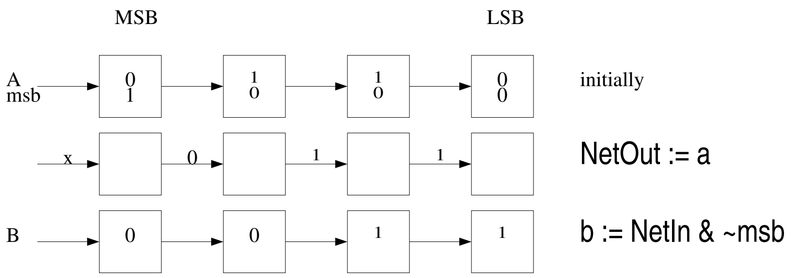

Figure 1-5: Logical shift right. The number 6 is shifted down to become 3.

Sum Accumulation. Accumulating a sum is a very common operation. The sum of a sequence of n numbers can be evaluated in $\Theta(n)$ cycles with a carry-save (CS) algorithm. The CS adder computes the sum and carry bits separately for each addition. The computation is purely local, and the delay of each CS stage is independent of word size. At the end of a summing sequence the carry and sum values must be summed with a full carry-propagation adder, but this delay is amortized over many summations. Many computations usually expressed as additions can be reformulated as accumulations, improving the performance of algorithms such as region summing. It is interesting to note that accumulation is an inherently simpler and faster operation than addition, and yet conventional processors do not make the distinction.

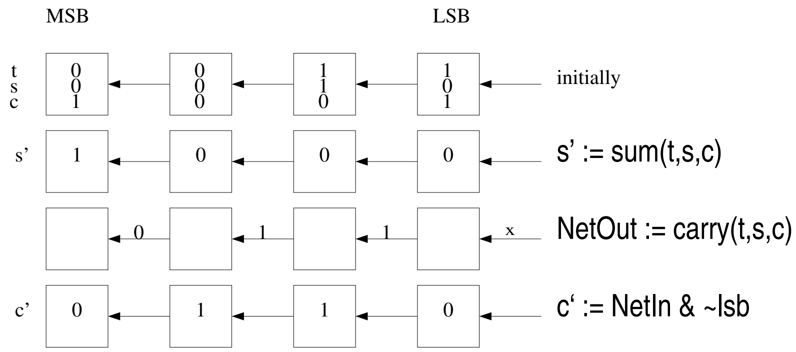

Figure 1-6: Accumulate. The addition of the number 3 to a sum of 11 represented as 2+9. After the operation, the sum is 14, represented as 6+8. All operations are purely local or nearest neighbor. Steps 1 and 2 can happen during the same cycle if a copy of t is present in both memory banks.

Addition. Fast addition relies on quick computation of the carry bit. A RBP algorithm based on the the well-known logarithmic-time technique was described in [**?**]. The algorithm "looks-ahead" at the carry by computing the eventual carry into each one-bit adder. It can be shown that the carry c_k into bit k can be expressed as $c_{k+1} = g_k + p_k c_k$, where g_k is the *generate* bit and p_k is the *propagate* bit. Since the computation of successive p and g values can be expressed in terms of a binary associative operator, it can be performed in time logarithmic with the number of bits. This technique forms the basis of many hardware implementation of adder circuits.

The Abacus architecture uses a different approach, one based on the hardware implementation of Manchester carry computation. In this technique, the carry bit propagates electrically down the line of bits unless prevented by a switch controlled by the p signal. Although computation time is linear with the number of bits, the constant factors result in faster operation than software emulation of the logarithmic-time circuit.

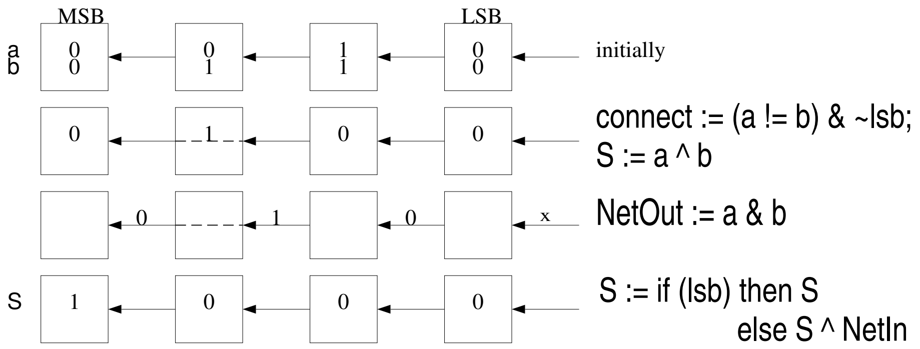

Figure 1-7: Manchester-carry addition. A NOP cycle occurs after step 2 to allow the carry bit to propagate. Steps 1 and 2 are actually merged into one cycle, and the `S` bit can be computed during the NOP cycle.

Match. The match operation compares two bit patterns for equality. First, all PEs form a wired-OR bus. Then, all PEs write `a != b` to the bus. If any bits differ, the bus becomes a 1; if all bits are identical, the bus remains at 0.

Comparison. The comparison algorithm is based on the observation that the most significant differing bit (MSDB) between two words determines which word is greater. Thus if the MSDB occurs in position i, $A > B$ when $A_i > B_i$, or equivalently $A_i = 1$ and $B_i = 0$. In the first step of the algorithm, all PEs with identical bits bypass themselves and do not participate in the rest of the computation. Then, all PEs send $A \cdot \overline{B}$ towards the MSB PE. Now, the MSB PE knows the value of $A > B$ for the rest of the word, and merges the result with its own data bits to produce the final answer. This entire process is equivalent to performing a Manchester-carry subtraction and testing for the borrow bit at the MSB.

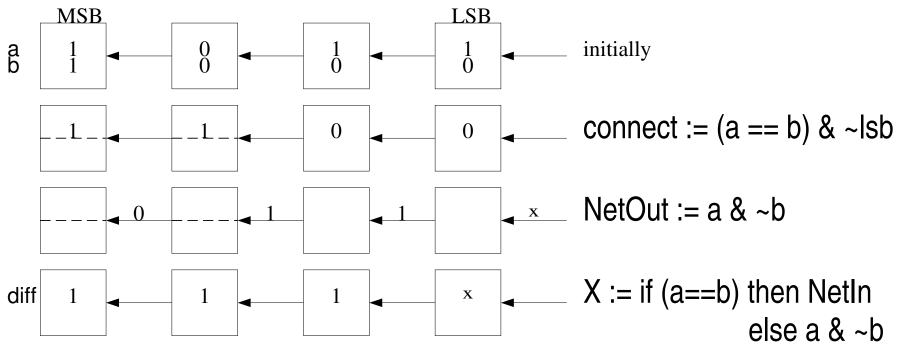

Figure 1-8: The `compare` operation. After the broadcast, the MSB PE receives A¿B for the rest of the word. It then merges the result with the MSB bit result.

Multiplication. RBP integer multiplication is composed of iterating through a sequence of shifts, broadcasts and carry-save adds. These operations can be partially overlapped so that an iteration requires 7 cycles. Thus, a 16 by 16 multiply requires 112 cycles, plus about 10 cycles of overhead. These numbers are for unsigned multiplication with a double-length result. If only the low word of the result is required, an iteration requires only 6 cycles. Two's complement multiplication is implemented with a software version of the Baugh-Wooley multiplier, and requires one additional iteration. While hardware multipliers frequently use the modified Booth algorithm to halve the number of iterations, this algorithm is not very efficient on the Abacus architecture since it requires the broadcast of three bits.

Mesh Move. In a mesh move each cluster rearranges itself into several shift registers in the direction of movement. The move operation is relatively expensive, as k bits must be transferred through a $\sqrt{k}$-wide bus, requiring $\sqrt{k}$ time steps. Further, unlike arithmetic operations, mesh moves cross chip boundaries and therefore suffer an additional 1-cycle latency cost.

Scan Operations. A set of computational primitives known as *parallel prefix* or *scan* operations have been found to be useful programming constructs across a variety of machine architectures. These operations take a binary associative operator $\oplus$, and an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ and return the ordered set $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots a_{n-1})]$. Common scan operations include *or-*, *and-*, *max-*, *min-*, and *+-scan*. There are also corresponding *reduce* primitives that reduce the values in a vector to a single value. For example, *+-reduce* calculates the sum of a vector, as shown in Figure 1-9.

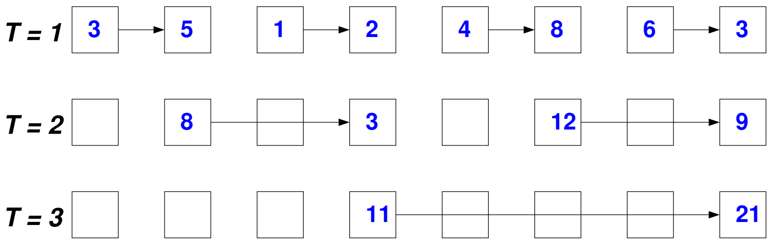

Figure 1-9: A +-reduce operation.

The bypass capability of the Abacus network can be used to implement fast scan operations. A message can propagate across a chip in two cycles. Crossing a chip boundary requires an additional cycle of latency. The prototype is a 16 by 16 chip array, so that 48 cycles are required for a bit to cross the array. Assuming a 4 by 4 bit PS organization, propagation of a 16-bit cluster requires 200 cycles. Since each stage of a logarithmic scan doubles propagation time, an entire scan operation requires approximately 500 cycles, or 4 microseconds. This can be optimized further by pipelining bit transmission.

1.0.5 Performance Summary.

The performance of a single Abacus chip on the operations described in this section is given is in Table 1. The left half of the table gives cycle counts, while the right gives the MOPS rating assuming 1024 PEs per chip and a 125 MHz clock rate.

These ratings are somewhat pessimistic for two reasons. First, they include an additional one-cycle penalty for reconfiguration. For example, if a right shift instruction is followed by an add instruction, PEs must be reconfigured from reading from the MSB direction to

	Cycle Count			MOPS		
Operation	8-bit	16-bit	32-bit	8-bit	16-bit	32-bit
Add, Compare	4	4	5	4000	2000	700
Shift	3	3	3	5300	2700	1300
Accumulate	4	4	4	4000	2000	1000
Mesh Move	5	6	8	3200	1300	500
Multiply	66	126	235	242	63	17

Table 1.1:

reading from the LSB direction. This penalty does not occur during every instruction, and a clever compiler can group instructions with identical configurations to reduce reconfiguration costs.

Second, as discussed earlier, microcode for several arithmetic operations can be overlapped. By the table entries above, the sequence $a = b + 2d + 2c$ performed on 16-bit values appears to require 14 cycles (two shifts, an accumulate, and an add). Yet a handcoded program performs the same operations in only 9 cycles by eliminating three reconfiguration steps, overlapping two broadcasts, and duplicating a data bit in both memory banks.

A high fraction of peak performance has been observed on simulations of vision algorithms.