IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

PICAP3 – A coarse–grain linear SIMD–array

Per–Erik Danielsson, Björn Lindskog and Jan Segerström

Linköping University
Department of Electrical Engineering
S–581 83 Linköping, SWEDEN

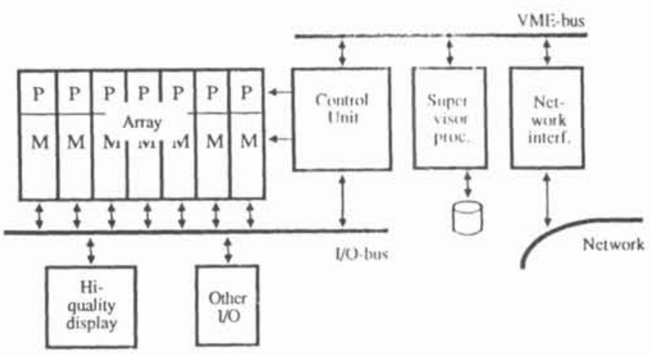
ABSTRACT

We give a brief description of the overall architecture of PICAP3. The enhanced processor module has full floating–point arithmetic and a 32 module machine will achieve a peak performance of 320 MFLOP. We show how the linear organization and the local address modification can be used efficiently for algorithms like FFT, Transposition, Histogramming, Convolution and Binary image processing. PICAP3 is orders of magnitudes faster than most commercially available systems.

INTRODUCTION

PICAP3 is a hardware/software low–budget project in parallel processing. Its roots are buried in a rather long tradition in computer design and architecture, most of it related to image processing and computer vision. However, the architecture of PICAP 3 has very little in common with its predecessors [1], [2], [3], [4]. The most salient features of PICAP3 are the following [5], [6]. See Figure 1.

- **SIMD** (Single Instruction Multiple Data stream). This was a natural decision since the main target applications were in low and medium level image processing.
- **Coarse granularity.** We deliberately abstained from customized VLSI–design. Therefore, to get as much processing power as possible per chip, each processing module is a 32–bit parallel ALU made from off–the–shelf components.
- **Direct processor–memory interconnect.** The power of all parallel architectures stems from a high memory bandwidth. We don't want to degrade this bandwidth by introducing an interconnection switching network. More importantly, such a network destroys the perfect modularity.
- **Linear processor – processor connection.** This is inexpensive and rather fast as long as the number of processors (P) is reasonably small. End–around word–wide bidirectional links constitute a quite powerful transportation highway as will be shown below.
- **High speed I/O–communication.** The linear organization makes it possible to attain input/output bandwidths equal to the processor/memory bandwidth. In the present implementation we are satisfied with a 40 MB/s data rate which typically occupies 5–10 % of available memory bandwidth.
- **A powerful, carefully designed control unit.** The control unit design is crucial. It has to supply the processor array with a constant flow of microinstructions, addresses and constants. The present design is capable of translating a logical three–component address into a physical address at full memory cycle rate.
- **Local address modification.** The globally issued address from the control unit can be modified locally. Table look–up functions is just one of several exploitations of this feature. It considerably enhances the performance of the SIMD–system and enlarges the space of amenable applications. Since it



has not been implemented in bit-serial SIMD arrays, the power of this feature is not generally well understood.

- **A large memory**, typically 4 MB/processor.
- **Moderate physical size**. A 32 module array fits on 8 PC-boards, the control, supervisory processor and network interface on another pair of PC-boards.

The present operational PICAP3 prototype is described in [5]. It contains only two processors. Currently we are designing an upgraded version of the processor module which will be described in the next section. For this design performance numbers from simulated execution will be given for a few algorithms.

THE NEW PROCESSOR MODULE

The new processor module is shown by Figure 2. All data paths are 32 bit and data transfers take place via the multi-ported register file RF (64 32-bit words). Global addresses and constants are pumped out from the control unit and enters the PE over the Global bus. This bus is bidirectional and may also be used to get values *from* the PEs. Once the address is inside RF, it may be modified by the ALU or directly used to access the memory. After two cycles data is available in the communication register (CR) and may be optionally transferred (in one extra cycle) to the left or right neighbor.

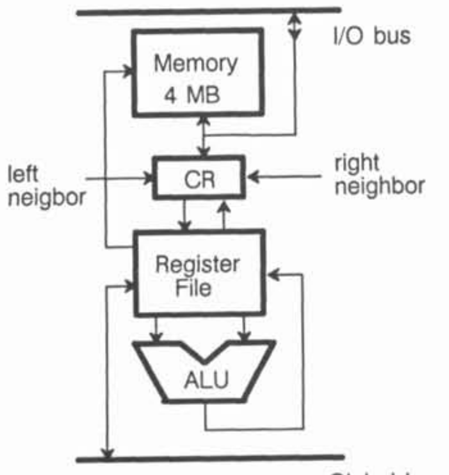


Figure 2

The arithmetical unit consists of a floating-point subunit, an integer subunit and an integer multiplier-accumulator, all three operating on 32-bit data. Both the register file and the ALUs are non-pipelined devices so that a full integer or floating-point operation can always be performed in one cycle time. The multi-ported register file serves the purpose of widening the data path from memory (5 MW/s) into a full seven bus system (70 MW/s). It also enables concurrent computation and memory accesses. By proper allocation of input, intermediate and output operands this gives a processing speed of 10 MIPS or 10 MFLOPS per PE

FFT

The basic operation in most FFT operations is the radix-2 butterfly. With each input being a two-word complex floating-point number (like the twiddle factor ω) the butterfly is executed in 10 cycles (four multiplications and six additions).

To bring the operands out and the result back to memory takes 16 cycles (four read and four write operations). However, this presumes that the coefficient ω is brought forward as a global constant from the control unit. If not, we need two more memory cycles to get it from the local memory. This gives 20 cycles.

Clearly, even if we use optimal overlap of memory and processor activity we are bounded by the memory access cycles when computing one butterfly at a time. When more than two points are included in each computation, the situation improves. An eight-point FFT is formed by 12 butterflies which takes 120 cycles. The intermediate results can be kept in the register file so that only 16 memory reads and 16 memory writes are needed. To pick up the ωs takes an extra 24 reads and a total of 112 cycles are needed for memory accesses.

The situation is now processor-bounded so that we obtain exactly

1 M Butterfly/s (1 MBUTT) per PE

Let us now assume that all input data for an N-point FFT reside in the local memory of the individual PE. A 1024 complex input data FFT then takes 5.12 ms. Assuming a **32 processor system** we obtain an efficient throughput of

1024 complex input FFT in 0.16 ms

The 2D FFT on a 512 x 512 complex data image is decomposed into two passes of 1D FFTs over the columns of the image. The second pass is preceded by a transposition (see below) and the total time is

$$2 \times 36.9 + 19.7 = 93.5 \text{ ms}$$

TRANSPOSITION. CORNER TURNING.

Evidently, there are numerous cases in a parallel system where data has to be rearranged in order to proceed with the computation in the most efficient way. One common operation is the so called corner turning which is illustrated by Figure 3 with an 8 x 8 matrix of data points. To begin with the leftmost PE contains data points 0, 8, 16, 24, 32, 40, 48, 56. After the transposal it contains 0, 1, 2, 3, 4, 5, 6, 7.

By proper use of local indexing we can move data over the left/right neighbor network in a rather efficient manner.The main trick is to bring out those data points at the same time that are to be moved the same horizontal distance [4], [5]. This is made possible by local indexing. Note that the average distance data has to travel is P/4.

N 32-bit words can then be corner-turned in a 32 PE machine using

$$N/32\ (4 + 32/4) = N \cdot 3/8 \text{ cycles}$$

where each word requires four cycles for memory acceses. This formula translates directly to 19.7 ms for the complex 512 x 512 image in the previous section.

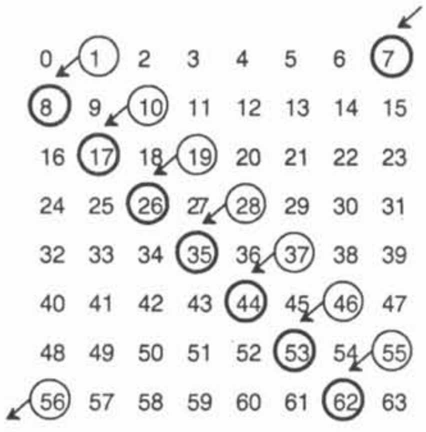


Figure 3

IMAGE PROCESSING

We will briefly describe three algorithms; *Histogramming, Convolution and Binary operations.*

Histogramming is performed as follows. In the first step all processors compute their local histograms. Two memory reads and one write are needed per pixel. Thus, to collect 32 local histograms of a 512 x 512 image it takes

$$6 \cdot 512 \cdot 16 = 49152 \text{ cycles or } 4.92 \text{ ms}$$

The second step is to merge these data into a single histogram. Assume a simple case of four PE's and 8 entries in the table as shown in Figure 4.

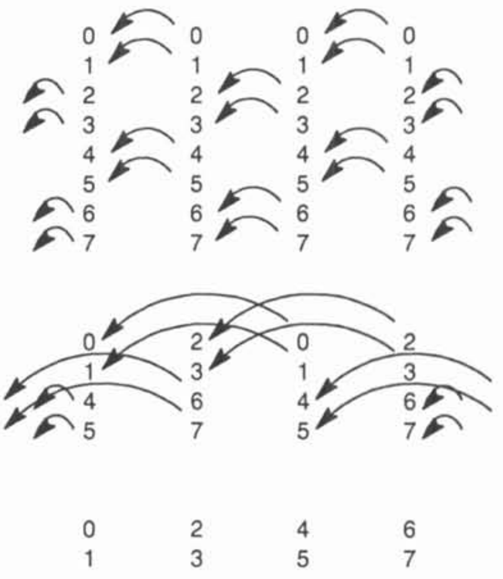


Figure 4

In the first pass, half of the 8 entries are moved one step and accumulated, in the next pass half of the remaining 4 entries are moved two steps and accumulated etc. With 32 PEs the total number of memory reads is 31N/16, memory writes 31N/32, and shift steps 5N/2. This totals

$$N \cdot 133/16 = 2128 \text{ cycles or } 0.21 \text{ ms}$$
$$\text{for } N = 256 \text{ (8 bit/pixel)}$$

We note that the merging time is negligible.The effectiveness of this merge procedure can be exploited in many other algorithms besides histogramming.

Convolution. We will here show the very common 3 x 3 filter.

The image is distributed in slices over the PE's. The neighborhoods for the border pixels overlap to the neighboring PE's but the time penalty for fetching from left or right is negligible. We may now proceed as shown in Figure 5 by keeping an input buffer holding the nine latest used input pixels.

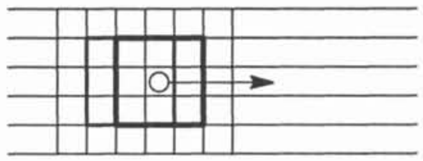


Figure 5

With this scheme, we see that there is only three memory reads and one memory write per result. With

9 multiply/accumulate and one final adjustment operation using the integer multiplier-accumulator we will be able to do one 3 x 3 kernel operation in 10 cycles. Since the memory accesses take 8 cycles, the operation is obviously processing bounded and the total execution time for a 512 x 512 image is

$$512 \cdot 16 \cdot 10 = 81920 \text{ cycles or } 8.2 \text{ ms}$$

Binary image processing. Using the integer ALU, the image data is *packed* ie a sequence of 32 vertical pixels is stored in one word. Without giving a full analysis, this can be done as fast as the pixels can be read out from the memory [5].

A 3 x 3 binary operation could now be executed as follows. See Figure 6 with the buffering method similar to Figure 5. Around the central 32 pixel word we put together a new 32-pixel word shifted one bit up and another one shifted one bit down. For each new set of 32 3 x 3 neighborhoods these two operations have to be carried out which take four cycles [5].

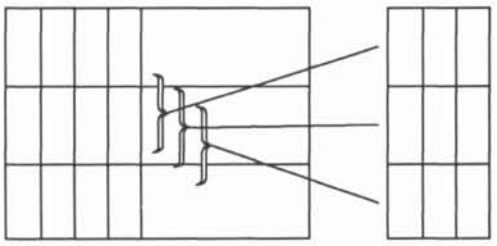


Figure 6

Then, we are ready to use the logic operations in the ALU. Ordinary expand (dilate) or shrink (erode) take 8 cycles in total. This is enough to overlap the memory cycles so that the time for a 512 x 512 expand/shrink becomes

$$512/32 \cdot 512/32 \cdot 8 = 2048 \text{ cycles}$$
$$\text{or } 0.20 \text{ ms}$$

A thinning operation is more complex and requires approximately 1.5 ms per step.

Effectively, we are using the 32 PE's as 32 x 32 = 1024 separate units for logic evaluation. It is therefore not surprising that the performance is as good or even better than for bit-serial SIMD arrays.

DISCUSSION AND SUMMARY

In the design of PICAP 3 we have tried to exploit two important trends in high speed computing: Very Large Scale Integration and Parallel Processing. We save cost, time and effort by using off-the-shelf components, we get simplicity in operation and control from the Single Instruction Multiple Data stream mode and we get high I/O-bandwidth from the linear organization.

The best illustration of these and other features of the machine is the following list of *estimated* processing times. All numbers are given for a 32 PE machine. Previous experience indicates that the real execution times are about 10 % slower mainly due to deficiencies in the control unit.

FFT		
1024 complex data	0.16	ms
512 x 512 complex data	94	ms
Transposition 512 x 512 8 bit data	9.8	ms
Histogramming 512 x 512		
8 bit pixel	5.1	ms
12 bit pixel	8.3	ms
Convolution 512 x 512, 16 bit pixel		
general 3 x 3 kernel	8.2	ms
Sobel	3.3	ms
Binary 512 x 512, packed data		
Expand, shrink per step	0.20	ms
Thin, per step	ca 1.5	ms

REFERENCES

[1] Kruse, B.: "A Parallel Picture Processing Machine", IEEE Transactions on Computers, Vol. C-22, No. 12, p. 1075, 1973.

[2] Kruse, B., Danielsson, P.E. and Gudmundsson, B.: "From PICAP I to PICAP II", in Special Computer Architectures for Pattern Processing, K.S. Fu and Ichikawa (eds.), CRC Press Inc., 1982.

[3] Danielsson, P.E.: "Vices and Virtues of Image Parallel Machines", in "Digital Image Analysis", Levialdi, S. (ed.), Pitman, 1984.

[4] Danielsson, P.E. and Ericsson, T.: "LIPP - Proposals for the Design of an Image Processor Array", Chapter 11 in "Computing Structures for Image Processing", Duff. M. (ed.), Academic Press, 1983.

[5] Lindskog, B.: "PICAP 3. An SIMD Architecture for Multidimensional Signal Processing", Linköping Studies in Science and Technology, Dissertations, No. 176, ISBN 91-7870-305-0, 1988.

[6] Lindskog, B. and Danielsson, P.E.: "PICAP 3. A Parallel Processor Tuned for 3D Image Operations", Proc. 8th International Conference on Pattern Recognition, Paris, France, 1986.