

EDITOR-IN-CHIEF

Marcin Paprzycki
Department of Computer Science
and Statistics
University of Southern
Mississippi
Hattiesburg, MS 39406-5106,
USA
pdcp@orca.st.usm.edu
http://orca.st.usm.edu/pdcp

BOOK REVIEWS EDITOR

Yi Pan
Computer Science Department
University of Dayton
Dayton, OH 45469-2160, USA
pan@cps.udayton.edu

SOFTWARE REVIEWS**EDITORS**

Hong Shen
School of Computing and
Information Technology
Griffith University
Nathan, QLD 4111
Australia
hong@cit.gu.edu.au

Domenico Talia
ISI-CNR
c/o DEIS, Università della

EDITORIAL BOARD

Peter Arbenz, Swiss Federal Inst. of Technology, Zürich, arbenz@inf.ethz.ch
Dorothy Bollman, University of Puerto Rico, bollman@jacobi.upr.clu.edu
Luigi Brugnano, Università di Firenze, brugnano@pontedera.pisof.it
Bogdan Czejdo, Loyola University, New Orleans, czejdo@beta.loyno.edu
Frederic Desprez, LIP ENS Lyon, Frederic.Desprez@inria.fr
David Du, University of Minnesota, du@cs.umn.edu
Yakov Fet, Novosibirsk Computing Center, fet@ssd.ssc.ru
Len Freeman, University of Manchester, freeman@ma.man.ac.uk
Ian Gladwell, Southern Methodist University, gladwell@seas.smu.edu
John Gustafson, Ames Laboratory, gus@ameslab.gov
Peter Kacsuk, KFKI-MSZKI Research Institute, kacsuk@sunserv.kfki.hu
David Keyes, Old Dominion University, keyes@cs.odu.edu
Vadim Kotov, Hewlett-Packard Laboratories, kotov@hpl.hp.com
Thomas Ludwig, TU, München, ludwig@informatik.tu-muenchen.de
Marek Machura, Bournemouth University, mmachura@bournemouth.ac.uk
Svetozar Margenov, CLPP BAS, Sofia, margenov@parallel.bas.bg
Veljko Milutinovic, University of Belgrade, emilutiv@etf.bg.ac.yu
Oscar Naim, University of Wisconsin-Madison, naim@cs.wisc.edu
George Papadopoulos, University of Cyprus, george@cs.ucy.ac.cy
Lalit Patnaik, Indian Institute of Science, lalit@micro.iisc.ernet.in
Viktor Prasanna, University of Southern California, prasanna@usc.edu
Stewart Reddaway, Cambridge Parallel Processing Ltd., sfr@cppuk.co.uk
Siang Wun Song, University of São Paulo, song@ime.usp.br
Boleslaw Szymanski, Rensselaer Polytechnic Institute, szymansk@cs.rpi.edu
Alejandro Teruel, Universidad Simón Bolívar, teruel@lde.usb.ve
Roman Trobec, Jozef Stefan Institute, roman.trobec@ijs.si
Carl Tropper, McGill University, carl@cs.mcgill.ca
Pavel Tvrdik, Czech Technical University, tvrdik@sun.felk.cvut.cz
Marian Vajteršic, Slovak Academy of Sciences, marian@iris.savba.sk
Jan Van Katwijk, Technical University Delft, J.vanKatwijk@twi.tudelft.nl
Mladen Vouk, North Carolina State University, vouk@adm.csc.ncsu.edu
Lonnie Welch, University of Texas, Arlington, welch@ohio.uta.edu
Janusz Zalewski, University of Central Florida, jza@ece.engr.ucf.edu

Parallel and Distributed Computing Practices

Volume 3, Number 2, June 2000

TABLE OF CONTENTS

Editorial: Tool Support for Distributed Real-Time Software <i>J. Zalewski</i>	v
Introduction <i>Yakov Fet and Peter Kacsuk</i>	111
SPECIAL ISSUE: UNCONVENTIONAL PARALLEL ARCHITECTURES	
Reaction-Diffusion and Excitable Processors: A Sense of the Unconventional <i>Andrew Adamatzky</i>	113
A Scalable Multiprocessor for Real-Time Signal Processing Applications <i>Hans Eberle and Daniel Scherrer</i>	133
Conciliating Connectionism and Parallel Digital Hardware <i>Bernard Girau</i>	145
KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications <i>Bertil Schmidt and Manfred Schimmmler</i>	163
The Effect of Interword Connectivity in Associative Processing <i>C. Konstantinopoulos, A. Svolos, D. Serpanos and D. Maritsas</i>	179
RESEARCH PAPER:	
Design, Implementation and Evaluation of a Parallel Object-Oriented Language A-NETL <i>T. Baba, T. Yoshinaga, Y. Iwamoto and K. Ootsu</i>	199
BOOK REVIEWS: <i>Yi Pan</i>	221

SUBSCRIPTION INFORMATION 2000

Volume 3 (4 issues) \$125

Nova Science Publishers, Inc.

227 Main Street
Huntington, New York 11743
Tel: (631)424-NOVA(6682)
Fax: (631)425-5933
E-Mail: Novascience@earthlink.net
Web Site: http://www.nexusworld.com/nova

Nova Science Publishers, Inc.

Huntington NY
2000

Production Staff: Publisher, Frank Columbus; Vice President, Nadya Gotsiridze; Senior Editors: Susan Boriotti and Donna Dennis; Coordinating Editor: Tatiana Shohov; Office Manager: Annette Hellinger; Graphics: Wanda Serrano; Book Production: Matthew Kozlowski, Jonathan Rose and Jennifer Vogt; Circulation: Cathy DeGregory and Ave Maria Gonzalez

Copyright © 2000 by Nova Science Publishers, Inc. All rights reserved. Printed in the United States of America. No part of this journal may be reproduced, stored in a retrieval system or transmitted in any form or by any means: electronic, electrostatic, magnetic, tape, mechanical photocopying, recording or otherwise without permission from the publishers. The publisher assumes no responsibility for any statements of fact or opinion expressed in the published papers.

KPROC – AN INSTRUCTION SYSTOLIC ARCHITECTURE FOR PARALLEL PREFIX APPLICATIONS

BERTIL SCHMIDT* AND MANFRED SCHIMMLER†

Abstract. The KPROC (KiloPROCessor) architecture is the first implementation of a parallel computer with 1024 floating-point processors on a single chip. It strictly follows the concept of an instruction systolic array. The modular organisation allows for either building large arrays of many KPROC chips or speeding up small machines with a single KPROC as a coprocessor. This paper presents concept of this parallel computer model as well as the architectural details of the processor design. It is shown that this computer model allows for efficient implementation of parallel prefix computations. A large variety of applications from different areas is presented to demonstrate how parallel prefix computations can be used as key operations for deriving efficient implementations on the KPROC.

Key words. single chip parallel computer, instruction systolic array, parallel prefix computations, image processing, cryptography

1. Introduction. Instruction systolic arrays (ISAs) have been developed to provide a programmable high performance hardware for specific computation intensive applications [7, 9]. Typically, such an array is connected to a sequential host; thus operating like a coprocessor which solves only the computationally intensive tasks within a global application [14, 15, 16, 17, 18]. The ISA is a mesh connected processor grid, where each processor is connected to its four direct neighbours by two unidirectional data wires. The processors are controlled by three streams of control information: instructions, row-selectors and column-selectors. Communication among adjacent processors occurs via designated communication registers. The whole system is globally clocked. Since data and control information is propagated only to physical neighbours, clock skew problems cannot occur.

KPROC is an ISA with 1024 processors integrated on a single chip. The first part of the paper describes the architecture and the design of the circuit. Due to the strict limitations in area and power consumption the functionality of the individual processor had to be restricted to a small number of simple machine instructions. The choice of this reduced instruction set is tailored towards an efficient implementation of complex operations like double precision floating point addition or 1024-bit integer multiplication.

In the ISA the execution of the instructions is pipelined in the two dimensions of the mesh. This skewed instruction execution leads to a significant advantage in comparison to a conventional SIMD mesh: Operations which need information propagation from left to right or from top to bottom in the array can be computed with a single instruction, whereas the SIMD mesh requires $\Omega(n)$ where n is the edge length of the mesh. In this paper it is shown that parallel prefix computations belong to these favourable operations. Parallel prefix computations are well-known key operations in many standard algorithms. Therefore, to implement efficient applications on the KPROC it is useful to design algorithms that take advantage of parallel prefix computations. We demonstrate this technique for several applications in numeric, graph theory, cryptography, and image processing.

The concept of instruction systolic arrays is explained in the second section. Section 3 gives an overview over the global architecture of the single processor. The KPROC machine consists of 1024 of these processors integrated on one die using a 0.25-micron digital CMOS process. The array allows off-chip communication with a data rate of 800 Mbytes per second between host computer and the KPROC-chip. In Section 4 it is proven that parallel prefix computations can be computed with only a single ISA instruction. Section 5 contains a number of examples how to convert a given algorithm in order to exploit parallel prefix computations. Section 6 concludes the paper with a collection of ideas of a future ISA machine.

2. Principle of the ISA. The ISA is a quadratic array of identical processors, each connected to its four direct neighbours by single data wires. The array is synchronised by a global clock. The processors operate bit-serially, i.e. one bit per clock cycle. The size of the internal registers

*School of Computer Engineering, Nanyang Technological University, Singapore 639798, (asbschmidt@ntu.edu.sg).

†Institut für Datenverarbeitungsanlagen, TU Braunschweig, Hans-Sommer-Str. 66, 38106 Braunschweig, Germany (masch@ida.ing.tu-bs.de).

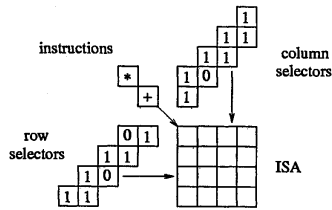


FIG. 2.1. Control flow in an ISA.

is 16 bits and each instruction produces a 16-bit result. Therefore, one period of 16 subsequent clock cycles is called an instruction clock cycle. The processors are controlled by instructions, row- and column selectors. In every instruction clock cycle one instruction, 32 row-selector bits and 32 column-selector bits enter the array.

The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. This guarantees that within each diagonal of the array the same instruction is active during a single clock cycle.

The selectors also move systolically through the array: the row selectors horizontally from left to right, the column selectors vertically from top to bottom (Fig. 2.1). The selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one. This construct leads to a very flexible structure, which creates the possibility of efficient solutions for a large variety of applications.

Formally, a program diagonal is a triple $\langle I, R, C \rangle$, where I is an instruction from a finite instruction set, R and C are words over $\{0, 1\}$. A program is a sequence of program diagonals $\langle I(k), R(k), C(k) \rangle_{k=1, \dots, m}$. An instruction $I(k)$ is executed in processor $P(i, j)$ at time $t = i + j + k - 2$ if and only if $R^{(k)} = 1$ and $C^{(k)} = 1$. m is called the *period* of the program. Since execution of each takes the same time (one instruction clock cycle), the period is a reasonable complexity measure for ISA programs.

Every processor has read and write access to its own memory. Besides that, it has a designated *communication register* that can also be read by the four neighbour processors. Two adjacent processors can exchange data by writing on their own communication register and afterwards reading the communication register of the other in two subsequent clock phases. Within one instruction clock phase the reading access is always performed before the writing access. This convention on the one hand avoids read/write conflicts, on the other hand it creates the possibility to broadcast information across a whole row or column with one single instruction.

Complex operations (like multiplication with long operands) are composed of two or even more subsequent instructions. Since every instruction is propagated to the neighbour processor directly after its execution, the processors do not require their own control memory.

3. KPROC architecture. The goal of integration of 1024 processors on a single chip has created a number of requirements for the implementation:

- The whole system should dissipate less than 5W. Since it consists of 1024 processors the individual processor was limited to slightly above 4mW.
- The chip area should not exceed 1.2 cm². Having 1024 processors plus padframe and clock distribution on one chip the area for one processor was limited to about 0.1 mm².
- 64-Bit and 32-Bit floating-point formats of IEEE should be supplied as well as all kinds of integer and word formats.
- The internal global clock rate should be 400 MHz.

Due to the limited area the processor had to be very compact. In fact, this was the reason for the choice of a bit serial data organisation. The size of the local memory was limited to 64 general-purpose registers plus one communication register. The wordlength of the data items was chosen to 16 bits as a compromise between the requirements of different application areas like image

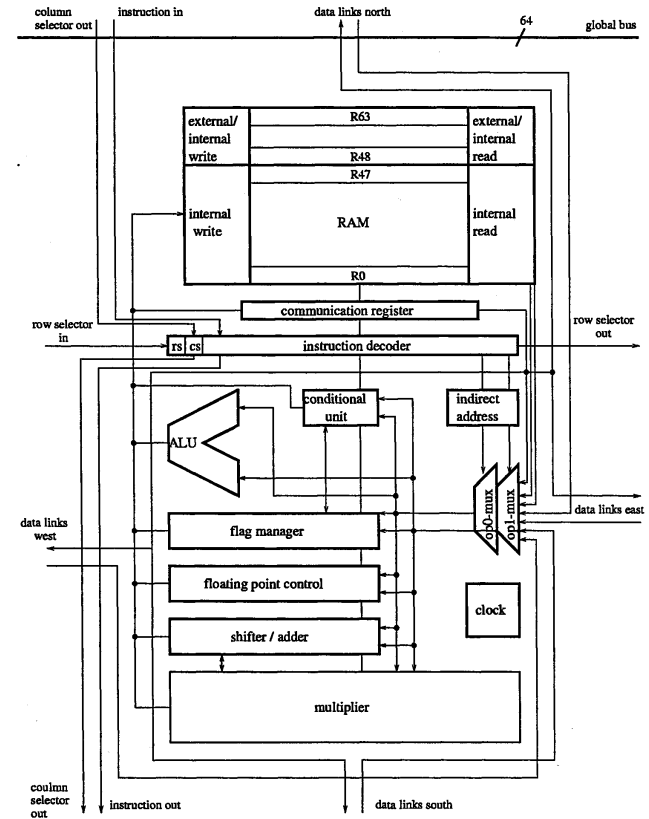


FIG. 3.1. Architecture of one processor.

processing (favourable 8 bits) and numeric (64 bits). The instruction set and the internal data formats were defined such that arbitrary formats for fixpoint and floating point arithmetic could be handled. Fig. 3.1 shows the global architecture of the single processor.

Two operand multiplexers choose two registers of the own local memory or of the communication registers of the neighbours (incoming data wires). The operands are propagated to the units required for execution of the actual instruction. These units can be the multiplier, the shifter, the floating point control, the conditional unit, or the ALU. The result of the execution is given to the result bus and from there written into the local registers or the communication register or both.

During the execution of one instruction the processor receives the next instruction together with a column selector bit from the upper neighbour and a row selector bit from the left neighbour. The selectors are interpreted. If both are 1 then the execution of the instruction is prepared. Is one of the selectors 0 then the execution of a no-operation is initialised. Instruction and column selector are propagated bit serially to the lower neighbour and the row selector to the right neighbour during

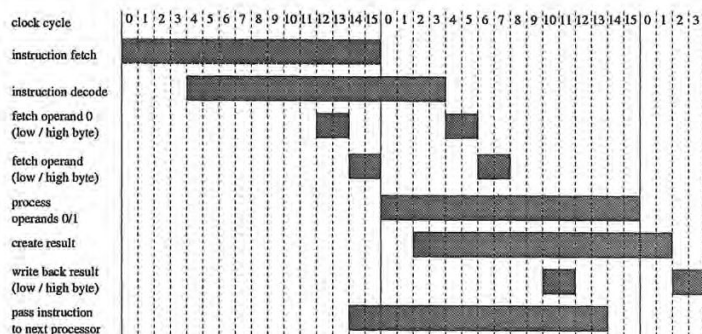


FIG. 3.2. Bit-level pipelined execution of an instruction

the execution of the instruction.

The local memory must be available for reading (of two operands) and writing (of one result) access within one instruction clock cycle. Although it is organised as single ported static RAM this could be achieved by reading and writing data fractions of 8 bits at 6 different phases of the instruction execution. This required an additional parallel-serial conversion in the read unit and a corresponding serial parallel conversion in the write unit of the RAM. Fig. 3.2 shows the pipelined structure of the memory access within each processor.

Architecture of the computation units

ALU The unit for integer arithmetic and logical operations is a standard ALU consisting of a full adder with feedback of the carry output to one of the inputs and a collection of logic gates for implementing AND, OR, NAND, NOR, XOR, EQV and arithmetical and logical shift operations. Since there are no new ideas in the design we do not describe the details here. The interested reader is referred to [3].

Multiplier The multiplier is a parallel-serial design operating in the following way: in the first instruction cycle one operand is loaded as multiplicand into the unit. In the following cycle (or cycles if operand lengths are greater than 16) the second operand is serially broadcast to all bits of the multiplicand, where a bitwise multiplication is performed. Simultaneously, the result bits are accumulated and propagated along the line of bits of the multiplicand. In order to perform a multiply-add in the same time as a multiplication it is possible to input a third operand (16 bits in each instruction cycle) into the multiplier. This third operand is added to the result on line. If the multiplicand is longer than 16 bits then the same procedure has to be performed more than one times, using the sum input to add the intermediate results of lower significance to the actual multiplication of higher precision.

Conditional unit Unfortunately, one disadvantage of SIMD-architectures holds also for ISAs: There is no possibility of conditional branches in the control flow of the machine. The KPROC solves this problem by means of a conditional unit. This unit allows assignments that depend on flags set by one of the preceding instructions. A conditional assignment has the form

```
if flag then A:=B else A:=C;
```

A branch of the form

```
if condition then statement1 else statement2;
```

can be equivalently transformed to

```
B:=statement1;
```

```
C:=statement2;
```

```
if condition then setflag;
```

TABLE 3.1
Tag bits to encode floating point numbers.

case	tag3	tag2	tag1
positive number	0	0	0
negative number	1	0	0
+0	0	0	1
-0	1	0	1
+infinity	0	1	0
-infinity	1	1	0
NaN (Not a Number)	X	1	1

TABLE 3.2
Computation of result tags in the case a multiplication (x: normal number, INF: infinity, NaN: Not a Number).

	x	0	INF	NaN
x	x^1	x	INF	NaN
0	INF	0	NaN	NaN
INF	INF	NaN	INF	NaN
NaN	NaN	NaN	NaN	NaN

if flag then A:=B else A:=C;

Of course, this requires both the execution of statement1 and statement2 (in all processors of the array). The original branch would execute only one of them. But practical applications show that the decrease in performance is subdominant with respect to the total execution time.

Floating point unit Several decisions on the architectural level were made because of power reasons. One example might illustrate this technique: In a conventional floating point processor the internal format is the one known as IEEE standard [1]. The IEEE 64-Bit format for example consists of one sign bit, 11 exponent bits and 52 bits of the significant (mantissa). The special cases occurring in floating point arithmetic (infinity, zero, not a number) are encoded as certain values of the significant and mantissa. This representation implies that in every floating point operation there are a large number of possible special cases (special operations to be performed) depending on the state of the operands. But the interpretation of the state of the operands needs to inspect all 64 bits. Obviously this implies large circuitry and a high power consumption in floating point computations.

The solution in the KPROC processor is very simple and efficient: instead of the IEEE standard, internally an extended standard is used. The significant is extended to 64 bits (which even gives additional precision) and three bits are added to the exponent. All necessary information about the special cases is encoded in these three bits. Therefore, the switching effort in every floating point instruction is minimised in comparison to the IEEE representation. These three tag bits encode the cases as shown in Table 3.1.

The disadvantage of this technique is some additional storage requirements (for the internal representation) and the additional conversion of every input floating point number into the internal format and every output floating point number into IEEE format.

Floating point multiplication is performed in the usual way: Exponents are added and mantissas are multiplied using the integer units discussed above. Interpretation of the tags during the addition of the exponents allows detecting all special cases. Table 3.2 shows how the tags of the result depend on the operand tags. Special cases for the result are unique except for the case where both operands are normal numbers. In this case an underflow of the exponent changes the result to 0 and an overflow to infinity. If no underflow or overflow occurs then the result is normal number.

Floating point addition is a bit more complicated: First, the exponents of the operands have to be compared. Then the mantissa of the operand with smaller exponent has to be rightshifted according to the exponent difference. After that the mantissas have to be added.

¹except the case of an exponent underflow or overflow, then the result tag is 0 or INF, respectively

TABLE 3.3
Computation of result tags in the case of an addition.

x	x	0	INF	NaN
x	x^2	x	INF	NaN
0	x	0	INF	NaN
INF	INF	INF	INF ³	INF
NaN	NaN	NaN	NaN	NaN

Finally, the greater exponent has to be chosen as the exponent of the result. Again, during the comparison of exponents the special cases are detected. The type of the result is shown in Table 3.3. The result of the addition of two normal numbers might be a normal number, 0, or infinity. Addition of plus and minus infinity results in not a number: Which of these cases hold is known after the addition of mantissas. Therefore, the result tags are written in the final step, where the greater exponent is chosen. Shifting of the mantissa is done by a combination of a configurable shift register and an address displacement. Concurrently with the addition of the mantissas the number of leading digits of the result are counted in order to normalise the floating point number after addition.

In order to keep to full precision in a sequence of floating point operations every result has to be normalised. This means the mantissa has to be shifted such that the first digit following the binary point is a 1 for positive numbers and a 0 for negative numbers in 2-complement. For normalisation again the shifter is used together with an address displacement as in floating point addition. The shifting distance is given by the value of the leading digit counter. After shifting the mantissa the exponent has to be corrected by subtracting the contents of the leading digit counter.

New features of the KPROC architecture In contrast to the existing commercial ISA implementations [4, 20], the KPROC architecture has a number of new features, that allows for more efficient parallel prefix operations:

- Random access data exchange with the external memory is possible, because KPROC has no interface processors at all. Instead, every processor has two memory banks, from which one is connected to the outside, while the other serves as local storage. These memory banks are switched with a special instruction.
- Addressing modes are generalised: Besides arbitrary 3-address instructions in the KPROC also indirect addressing for the two source operands and for the destination have become possible.
- A number of specialised machine instructions allows for faster 8-bit oriented image operations.
- Every processor has one designated ready flag. This can be used to control conditional instructions in order to run iterations and optimisations with a data-dependent termination condition.

In Section 5 we will show how these features help to speed up several parallel prefix computations.

4. Parallel prefix computation. The *parallel prefix computation* [8] can be recursively defined as to be given inputs $x_0, \dots, x_{n-1}$ compute outputs $y_0, \dots, y_{n-1}$ with $y_0 = x_0$ and $y_i = y_{i-1} \otimes x_i$ for $1 \leq i \leq n-1$ where $\otimes$ is an associative binary operator. The following proposition holds for the calculation of parallel prefix computations on the ISA:

PROPOSITION 4.1. *If the operator $\otimes$ can be executed in a single ISA processor within a single instruction, the corresponding parallel prefix computations $y_0, \dots, y_{n-1}$ can be computed in an ISA processor row (column) of length n with period 1.*

Proof. Let the C-register of the i^{th} processor of the ISA processor row (column) initially store the input value x_i , $0 \leq i \leq n-1$. The instruction $C := C \otimes C[\text{WEST}]$ ($C := C \otimes C[\text{NORTH}]$) performs the parallel prefix computation, i.e. each processor i , $0 \leq i \leq n-1$, has calculated y_i in its C-register after the execution of this instruction. This holds because of the pipelined instruction flow.

²except the case that the result mantissa is 0, then the result tag is 0

³except the case that the signs of the two operands are different, then the result tag is NaN

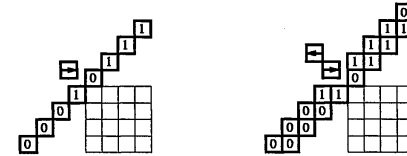


FIG. 5.1. Left: Broadcast along the first row of the array. Right: Ringshift of a data item in the first row of the array.

Of course, which parallel prefix computations are efficiently executable on the ISA depends on the instruction set of the individual processor. In contrast to the calculation of a parallel prefix computation on an ISA processor row of length n with constant period its calculation on a mesh-connected SIMD processor array requires $\Omega(n)$.

Parallel prefix computations can be used to solve a number of problems efficiently. Ladner and Fisher show how they can be used to parallelize the computation of an arbitrary Mealy machine [8]. Leighton discusses a number of different problems that can be solved using prefix computations [11]. In the following section the ability of the ISA to perform parallel prefix computations in an extremely efficient way is exploited as a key operation to derive efficient implementation on the KPROC for several computationally intensive applications in cryptography, image processing, and multimedia.

5. Applications.

5.1. Broadcast and Ringshift. The simplest example of a parallel prefix computation is broadcasting, i.e. $x \otimes y = x$. Thus, only one instruction is needed in order to propagate a data item from left to right along a row of processors, just as if there were a broadcast line connecting all processors of a row (or column, respectively). This instruction is $C := C[\text{WEST}]$ (read the contents of the C-register of the western neighbour and write it into the own C-register), denoted by an arrow symbol in Fig. 5.1 (left). Sending a data item from right to left along a row takes $2n$ instructions. So one would write programs in a way that whenever possible propagating data is done from left to right or from top to bottom.

The fictitious broadcast line can be used as a *wrap-around connection* of the array in order to realise ringshift operations. If the C-register contents of processors $2, \dots, n$ of a row are simultaneously read by their left neighbours, a left shift of the data of this row is performed. However, the data item in the first processor would be lost. Therefore, this data item is broadcast to the last processor, yielding a ringshift of the data items. Fig. 5.1 (right) shows the program for a 4×4 array.

Broadcast and ringshift provide elegant and efficient solutions for the problem of interprocessor communication on the ISA. While allowing communication along a whole row or column, these operations use only local communication and take only one or two instructions. But the number of instructions is the natural measure for the time complexity of an ISA program. The following programs are examples for the use of broadcast and ringshift operations.

Matrix Multiplication: The problem is to compute the product of two $n \times n$ matrices $F = A \cdot B$, defined as $F_{ij} = \sum_{k=1, \dots, n} A_{ik} \cdot B_{kj}$.

We first assume that $n = 32$ (thus, fitting into the KPROC array). Suppose that matrix A and matrix B have been loaded into the array. Then the first column of A , which is in the first column of the array, is broadcast along the rows and stored intermediately in some double register H . Similarly, the first row of B is broadcast along the columns. Now each processor $P(i, j)$ has elements A_{i1} and B_{1j} available and can compute $F_{ij} := F_{ij} + A_{i1} \cdot B_{1j}$. The next step is to ringshift A by one column to the left in order to get the second column of A , elements A_{i2} , into the first column of the array, from where they can be broadcast along the rows. Similarly, B is ringshifted by one row to the top, in order to be able to broadcast elements B_{2j} along the columns. After n of such steps, registers F contain the product of A and B .

If $n = 32 \cdot k$, $k \geq 2$, A , B , and F are partitioned into 32×32 submatrices $A[p, q]$, $B[p, q]$, $F[p, q]$, $0 \leq$

$p, q \leq k-1$. Each submatrix $F[p, q], 0 \leq p, q \leq k-1$, is computed in k steps, where in each step $r, 0 \leq r \leq k-1$, the two submatrices of $A[p, r]$ and $B[r, q]$ are multiplied as described above. The submatrices $A[p, q+1]$ and $B[q+1, p]$ can be loaded into the processor array concurrently to the processing in step $r, 0 \leq r \leq k-2$.

The same routing techniques as described for matrix multiplication can be applied for the ISA implementation of the solution of linear equation with positive definite matrices and the computation of the transitive closure of a graph on the ISA [4].

Convolution: Convolution is a fundamental image processing operation. It focuses on the question how to combine the pixels of a local neighbourhood. The elementary combination of the pixels in the neighbourhood is given by an operation which multiplies each pixel in the range of the filter mask with the corresponding weighting factor of the mask, adds up the products (multiply-add), and writes the sum to the position of the centre pixel:

$$g_{i,j} = \sum_{n=-M}^M \sum_{m=-M}^M h_{n,m} \cdot f_{i-n,j-m} = \sum_{n=-M}^M \sum_{m=-M}^M h_{-n,-m} \cdot f_{i+n,j+m} \quad i, j = 0, \dots, N-1$$

This equation assumes an odd-sized mask $h_{n,m}$ with $2M+1 \times 2M+1$ coefficients and an $N \times N$ image $f_{i,j}$. In the following we will use the *cyclic convolution*, i.e. pixels $f_{i-n,j-m}$ with $i-n$ or $j-m \notin \{0, \dots, N-1\}$ are defined by the physically existing pixels $f_{(i-n) \bmod N, (j-m) \bmod N}$.

Again, we first assume that $N = 32$ and $2M+1 \leq N$. Suppose that $f_{i-M,j-M}$ and $h_{i+M,j+M}$ have been loaded into processor $P(i, j)$. The basic idea of the ISA algorithm is to consider a superposition of $(2M+1)^2$ frames of the input image and the filter mask. In iteration step $n, m, -M \leq n, m \leq M$, each processor (i, j) has the elements $f_{i+n,j+m}$ and $h_{i-n,j-m}$ available by using two row ringshifts or two column ringshifts and the coefficient $h_{-n,-m}$ is broadcasted from the upper left processor to all other processors. Afterwards each processor $P(i, j)$ computes the multiply-add operation $g_{i,j} := g_{i,j} + h_{-n,-m} \cdot f_{i+n,j+m}$ in parallel. After $(2M+1)^2$ such steps the cyclic convolution of f and h is performed.

In image processing applications typical values for N and M are $N = 1024$ and $M = 7$. Because the data volume of the input image (1 Mbytes for 8-bit pixels) is larger than the capacity of the KPROC (128 Kbytes) the input image has to be partitioned and mapped onto the processor array. There are a lot of possibilities of such mappings [19]. As an example, an entire row of 1024 pixels can be mapped onto the processor array in a row-major way. Since it is possible to store more than one pixel in each processor, several rows of the image can be processed simultaneously with the additional benefit that local operations in the columns of the image can take place in one processor. However, the introduced routing technique can be used for both mappings.

Similar routing techniques can be also applied in the case of other neighborhood operations in image processing, as e.g. morphological operators, median filter, and wavelet transform [19].

Fast Fourier Transform (FFT): Fig. 5.2 shows the signal flow graph of the FFT of a sequence of n points, here for $n = 8$. It consists of $\log(n)$ stages. In each stage, a complex multiplication with some primitive root of unity, a complex addition, and a complex subtraction is performed at each of the n points.

The output of each stage is permuted by interchanging the two halves of subsequences of length 2^k before being input to the next stage. At the beginning, the sequence of the n points is permuted according to the so called bit reversal permutation.

The implementation of a 2-dimensional FFT of size $n \times n$ on the ISA consists of two subsequent passes: first the rows are transformed and then the columns. Each row is mapped onto $\frac{n}{64}$ rows of the ISA such that each processor stores two values.

In the case of $n = 256$, for instance, four rows of the processor array are needed for one row of data. The complex roots of unity are precomputed and are loaded to the processor array when needed. This load operation takes only a small constant number of instruction cycles since the values can be broadcast along the columns of the ISA. The computation in each stage requires 4 (real) multiplications and 6 (real) additions in each processor.

The implementation of the permutation between every two stages involves routing of data, which is performed efficiently either within each row of the ISA (in the case of interchanging the two

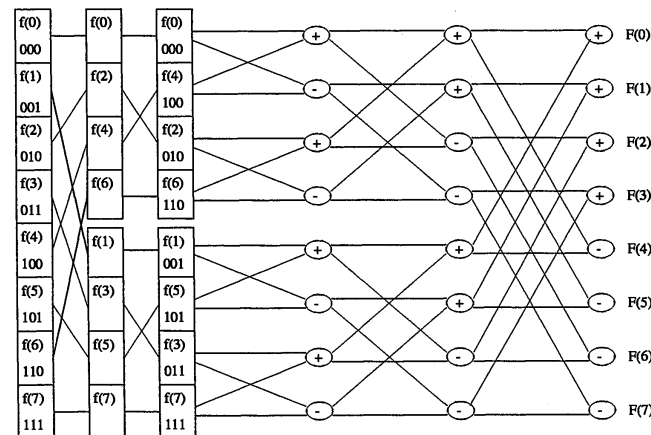


FIG. 5.2. Signal flow graph of an 8-point FFT.

halves of subsequences of length ≤ 64) or by interchanging complete rows of the ISA (in the case of interchanging the two halves of longer subsequences). The interchanging of the C-register contents of two halves of m processors is implemented by $\frac{m}{2}$ ringshift operations. The initial bit reversal permutation of the input values can be realised efficiently by combining it together with the input of operands into the processor array.

Because of the two passes needed for transformation of rows for columns, the performance of 2D FFT in the conventional ISA implementations [4] was significantly affected by the time required for input and output of intermediate results. But this fraction of computing effort can be reduced on the KPROC to almost zero by the bank mechanism described in Section 3.

The similar routing technique as described for FFT can also be used for the implementation of the Discrete Cosine Transform (DCT) on the KPROC.

Filtered Backprojection: Tomographic reconstruction deals with the reconstruction of an object from its projections [13]. The technique of CT scanners consists of passing a series of X-rays through an object, and measuring the attenuation in these rays by placing a series of detectors on the receiving side of the object. These measurements are called projections. For the collection of projection data in the 2D plane two different modes are used: *parallel beam scanning* and *fan beam scanning* (see Fig. 5.3). The goal of tomographic reconstruction is to obtain an image of a cross section of the object from such a set of profiles. If this process is repeated at various heights along the object, we obtain several 2D cross-sectional images, which can then be stacked one on the top of another to get internal 3D volume visualisation of the object.

Because a finite number of angles and rays are taken from the CT scanner it is impossible to reconstruct the analysed object exactly. This problem is mathematically described as the *Inverse Radon Transform* [6]. Thus, only discrete approximations can be computed. The filtered backprojection algorithm is currently the most used reconstruction method. It consists first of a filtering operation and a following backprojection operation.

Filtering is needed to undo blurring effects due to the low pass nature of the backprojection line integrals. Thus, a 1D filter is applied on each profile (projections of the same angle). Filtering of parallel beams is performed by taking the Fast Fourier Transform (FFT) of each projection, multiplying by the filter frequency response, and taking the inverse FFT. The implementation of the FFT on the KPROC is described above.

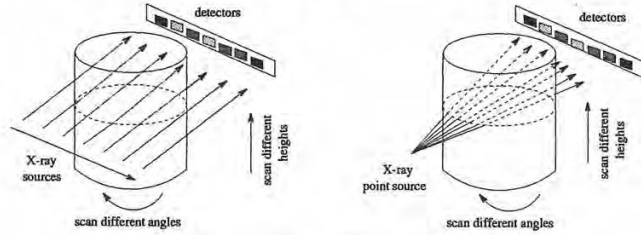


FIG. 5.3. Different scanning modes in CT: parallel beam (left) and fan beam (right).

```

for t := 0 to T - 1 do
  for i := 0 to N - 1 do
    for j := 0 to N - 1 do
      p := i · cos θt + j · sin θt
      g(i, j) := g(i, j) + Pθt([p]) + ([p] - p) · (Pθt([p]) - Pθt([p]))

```

FIG. 5.4. Algorithm Back.

Backprojection of parallel beams reconstructs each pixel by summing up one filtered projection value of each angular sample. To reconstruct the pixel (x, y) the projection value $P_\theta(x \cdot \cos \theta + y \cdot \sin \theta)$ has to be added for the angle θ . In general, $x \cdot \cos \theta + y \cdot \sin \theta$ is a non-integer. Thus, the needed value has to be determined by interpolation between the two physically existing neighbouring projections. Usually, linear interpolation provides the best trade-off between reconstruction quality and execution time [6].

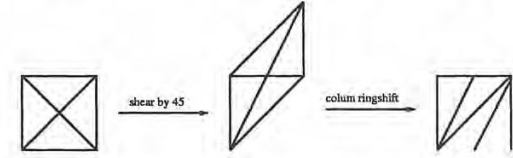
Filtered backprojection of fan beams can be reduced efficiently to the filtered backprojection of parallel beams using the *rebinning algorithm* [5]. In order to produce a parallel implementation of backprojection of an $N \times N$ image on an $N \times N$ ISA each processor reconstructs one pixel in parallel. In Algorithm Back (see Fig. 5.4) we perform the accumulation operations for the complete image slope by slope. In each iteration step t only the projection values of one profile (θ_t) have to be considered. In the ISA implementation these projections are broadcasted through the processor array such that each processor gets the relevant projection values corresponding to its pixel. Afterwards each processor can compute the interpolation and accumulation in parallel.

The easiest case is the backprojection of horizontal projections. Each projection $P_\theta(i), i = 0, \dots, N-1, \theta = 0^\circ$, is broadcast and accumulated along processor row i in parallel. By means of shearing of the image all other projection angles can be reduced to horizontal backprojection, such that the efficient row broadcast operation can always be applied.

Shearing is an operation frequently used in scan line image processing [12]. The idea is to reshape the image such that the actual scan line of arbitrary slope appears to be horizontal. This is achieved by shifting the columns with increasing shift distances. Fig. 5.5 depicts the effect of shearing: A straight line of 45° slope appears to be horizontal if the image is sheared by 45° .

Shearing an image by an angle θ , with $0^\circ \leq \theta \leq 45^\circ$, is calculated by shifting each column j by the distance $j \cdot \theta$. The implementation of shearing an $N \times N$ image on an $N \times N$ ISA has to be done in place taking only integer values as shifting distances. Thus, shifting of column j is implemented by ringshifting column j with the distance $[j \cdot \theta]$.

For an efficient implementation on the ISA ringshifting along large distances has to be avoided. Thus, different projection angles are handled in increasing order and the ringshifting distance only needs to incorporate the relative shear between neighbouring angles $[j \cdot \theta_t] - [j \cdot \theta_{t-1}]$ for each column j . The distances for relative shearing are known in advance. Thus, they are precomputed and are loaded into the processor array when needed using the column broadcast operation.

FIG. 5.5. The central image depicts the left image sheared by 45° . The right image depicts the sheared image stored in place using column ringshift.

Assuming, projections were generated using N rays for each angle and the reconstruction image is of size $N \times N$, the distance between every two rays of an angle θ , with $0^\circ \leq \theta \leq 45^\circ$, is $\frac{1}{\cos(\theta)}$ with respect to each image column. Because row broadcasting is always applied for backprojection, exactly one projection value has to exist between two neighbored pixels of each image column. Thus, projection values between every two rays of an angle θ , with $0^\circ \leq \theta \leq 45^\circ$, are calculated with distance 1 with respect to each image column. This operation is computed by linear interpolation between the existing projections and takes only a negligible amount of work. Now, the indices for linear interpolation between neighbouring projections of an angle θ turn to a constant value for each image column j : $\text{frac}(j \cdot \theta)$. Thus, they can be loaded into the processor array using the column broadcast operation.

The accumulation step for each angle θ , with $0^\circ \leq \theta \leq 45^\circ$, loads the projection values $P_\theta(i-1)$ and $P_\theta(-N+i-1)$ into each processor row i , for $i = 0, \dots, N$, using row broadcast. The interpolation indices $I_\theta(j)$ are loaded into the processor columns j , for $j = 0, \dots, N$, using column broadcast. Afterwards each processor gets also the projection values from its southern neighbour using the column ringshift operation. Because the implementation of shearing was done in place there are pixels in some processor rows $i, i = 0, \dots, N-1$, needing the projections $P_\theta(-N+i), P_\theta(-N+i-1)$ instead of $P_\theta(i), P_\theta(i-1)$. These are exactly the pixels, which lie above the 0^{th} image row in the original sheared image (see Fig. 5.5). Each processor stores this information in a flag register. Depending on this flag each processor (i, j) decides either to take the projections $P_\theta(i), P_\theta(i-1)$ or $P_\theta(-N+i), P_\theta(-N+i-1)$ for the linear interpolation and accumulation:

```

for all processors (i, j) do in parallel
  if flag then P0:=Pθ(i); P1:=Pθ(i-1) - Pθ(i);
  else P0:=Pθ(-N+i); P1:=Pθ(-N+i-1) - Pθ(-N+i);
  accu[i, j] := accu[i, j] + P0 + Iθ(j) · P1;

```

Because column ringshifting on the ISA is efficient only in one direction, the above implementation can only be used for projection angles θ , with $0^\circ \leq \theta \leq 45^\circ$. Angles θ , with $-45^\circ \leq \theta \leq 0^\circ$ are backprojected by mirroring the image columns and applying the same ISA program. The angles between 45° and 90° (-45° and -90°) are calculated by applying the *reflected ISA program* (interchanging row and column selectors, changing west and north, east and south) for 0° to 45° (-45° to 0°).

The implementation of backprojection on the KPROC has to partition the implementation for an $N \times N$ ISA on a 32×32 ISA. In order to reduce computing time each processor can reconstruct 2×2 pixels instead of one. Thus, a 64×64 subimage is reconstructed in parallel in one step. The complete $N \times N$ image is reconstructed by $\frac{N}{64} \cdot \frac{N}{64}$ iteration steps.

The shearing technique together with accumulation can be also applied for the implementation of the *Hough transform* (HT) and *Morphological Hough transform* (MHT) [16], which is a frequently used method for line detection in image processing. The MHT is an example how a classical algorithm (HT) is modified in order to improve its performance, while fitting of the ISA architecture.

5.2. Accumulation. If the instruction $C := C[\text{WEST}]$ is replaced by $C := C + C[\text{WEST}]$ (add the contents of the C-register of the western neighbour to the own C-register), an accumulation over an entire row can be performed with one instruction, which is identical to the parallel prefix computation defined by $x \otimes y = x + y$. If this is done in all rows simultaneously, followed by the instruction

$C := C + C[\text{NORTH}]$ executed in the last column, the sum of all C-register contents of the entire array is obtained in the lower right processor.

Motion estimation: Video data compression techniques rely on the fact that in general two subsequent frames are similar. Often the difference of the frames is that part of the first frame has moved. In order to detect such a motion, the second frame is decomposed into 16×16 -pixel macroblocks (MBs), and each MB is compared at all possible positions in its neighbourhood on the first frame. A motion vector (x, y) of a MB at position (i, j) in the second frame $F^{(1)}$ has been detected, if it matches at position $(i - x, j - y)$ in the first frame $F^{(0)}$, i.e. if the sum of absolute differences

$$\sum_{p=i}^{i+15} \sum_{q=j}^{j+15} |F_{p,q}^{(1)} - F_{p-x, q-y}^{(0)}|$$

is 0 or lies below a certain threshold.

An obvious implementation of this motion estimation technique is to store the two frames on the processor array in such a way that each processor contains corresponding pixels $F_{p,q}^{(1)}$ and $F_{p-x, q-y}^{(0)}$ of the two frames. Computing the absolute value of the difference of the pixels takes two instructions, summation within 16×16 -blocks takes another two instructions using the accumulation operation. The preceding computation procedure is repeated until all candidate MBs are compared and the final motion vector is obtained.

The KPROC architecture speeds up motion vector estimation in a threefold way: First, a designated instruction `absdiff` computes two absolute differences of 8-bit pixels each within one instruction clock cycle. Second, the ready flag allows stopping the process of motion estimation as soon as one good enough candidate for the motion vector is found. Third, loading of the new MB is speeded up by random input access. These three effects add up to a factor of at least 2.5 in performance (see Section 6).

Histogram: Computing a histogram is a basic operation in image processing and computer vision. It can provide useful information for segmentation and measuring the textual properties of an image. Given an $N \times N$ image with h grey level values, the histogramming problem is to compute the number of occurrences of these values. Although, computing the histogram is a simple task for sequential computers it cannot be efficiently implemented on conventional mesh-connected SIMD-type parallel machines. This holds also for the ISA in its previous implementations [4, 20]. Since image processing is a key application area for fine-grained parallel architectures it is necessary to overcome this problem.

KPROC solves it by combining indirect addressing with a two dimensional accumulation: Every processor computes a counter $R[I(i, j)]$ for each grey value $I(i, j)$. The operation `ADD R[I(i, j)], 1`, `R[I(i, j)]` increments the appropriate counter. After processing of all tiles, a two-dimensional accumulation operation for each grey value computes the sums of the counters and so the final histogram.

5.3. Carry propagation. The difficulty of partitioning an addition in an array where only neighbours can talk to each other is the carry propagation. The situation is easy as long as a generated carry has to be passed to the next neighbour only where it can be added without generating a new carry. But if it meets a sum value consisting of ones only, then an incoming carry produces a new carry in the most significant digit. In the worst case this can hold for all blocks of the partitioned addition, such that the carry-bit of the first block propagates through all other blocks up to the most significant block where it is finally absorbed. Of course, the addition on the ISA must be solved data independently, i.e. the number of instructions must not depend on the propagation distance of the carries. Therefore, a parallel prefix computation based on the generate and propagate signals of a *carry-look-ahead-adder* is used.

Suppose, every processor knows, whether it has to propagate an incoming carry from the left to the right. Then it can set a flag (e.g. the zero-flag) to one if and only if it propagates an incoming carry. Furthermore every processor can store the carry generated by itself in its communication register. With the (parallel prefix) operation

(*) if `zeroflag` then $C := C[\text{WEST}]$

all carry-bits travel to the processor one position left of their destinations. This is again because of the skewed execution of the instructions in the processor array. Suppose, processors $(i, j-1)$, (i, j) , and $(i, j+1)$ have set their zero-flag to one (in order to propagate a carry) and processor $(i, j-2)$ has generated a carry-bit one. The instruction (*) is executed in processor $(i, j-1)$ which replaces its carry-bit (0) with that of processor $(i, j-2)$. In the next clock cycle the same instruction is executed in processor (i, j) , where again the own carry (0) is replaced by the value of the left neighbour (which has been set to one in the previous instruction cycle). In the following cycle, processor $(i, j+1)$ replaces its carry with the one from the left neighbour, etc.

With the final assignment $C := C[\text{WEST}]$ in every processor all carries are at the place where they are needed. C denotes the own communication register and $C[\text{WEST}]$ the communication register of the left (western) neighbour. Note that the carry propagation requires only two instructions, although the carry-bits can travel over an arbitrary distance (of up to $n-1$ processors of an $n \times n$ -ISA).

Of course, this technique only works if no processor propagating a carry has generated a carry-bit on its own. But this is obviously impossible, since the propagation condition is a sum consisting of ones only, whereas the generating condition is a sum $\geq 2^m$ where m is the number of bits to be added in one blocks of the partition. If both conditions hold, the sum would be $2^{m+1} - 1$, which is impossible because the operands are both $\leq 2^m - 1$.

To achieve the optimal speedup, the operands to be added or subtracted are distributed over the rows of the KPROC. If the length of the operands is 512 bits, every processor in the row (of 32 processors) gets 16 bits. The least significant bits are stored in the leftmost and the most significant bits in the rightmost processor of the row. For longer operands (e.g. 1024 bits), every processor is responsible for more bits (e.g. 32 bits). If the length of the operand is smaller, it is useful to divide the rows into pieces where every piece is responsible for one addition or subtraction. It is not reasonable to store less than 16 bits per processor because the arithmetic units of the processors of the KPROC perform one 16-bit operation within one clock cycle.

Now the addition is performed in the following way. Every processor has 16 bits of the operand A in register RA and 16 bits of B in register RB . The operations

$RSUM := RA + RB;$

$C := \text{carry};$

store the sum of RA and RB in $RSUM$ and the carry-bit in the communication register C . Now every processor must find out whether it has to propagate an incoming carry. This is the case if and only if the value of $RSUM$ is consisting of ones only. If $RSUM$ has at least one zero, this will absorb an incoming carry-bit. If the propagate condition holds, $RSUM + 1$ is zero.

Now we can proceed as pointed out above with

if $(RSUM + 1 = 0)$ then $C := C[\text{WEST}];$

$RSUM := C[\text{WEST}] + RSUM;$

completing the long addition. Note that the carry-bit generated in the final addition is redundant.

In [17] is described how the carry propagation technique can be applied to the multiplication, division (using the Newton-Raphson method), and modular multiplication of long operands (e.g. 2048 bits) on the ISA. The demand for long number arithmetic arises in the field of public-key cryptography. Thus, the presented arithmetic leads to a high-speed implementation for RSA encryption and decryption on the KPROC. Another cryptographic application that can use the same arithmetic routines is prime number generation using the probabilistic Rabin-Miller test.

6. Conclusions. We have presented the architecture of the KPROC processor which is the central processing unit of the first massively parallel system integrated on a single chip. The underlying parallel model is the ISA which has been shown to be optimally suited for parallel prefix computations. A collection of practical applications show the relevance of operations of this type to a large variety of algorithms.

To show the effect of the new features of the KPROC architecture, Table 6.1 shows a quantitative comparison of the instruction count for the presented parallel prefix applications on the Systola 1024 PC add-on board ISA implementation [4] and the KPROC chip.

TABLE 6.1

Instruction count and speed up for parallel prefix applications on the Systola 1024 PC add-on board and the KPROC chip. Because both architectures comprise a 32×32 ISA of bit-serial processors with 16-bit word length, the table shows the effect of the new features of the KPROC.

Description of application	Instruction count		Speedup
	KPROC	S1024	
Matrix multiplication (64×64 matrices, IEEE single)	7160	7568	1.06
Convolution ($256 \times 256 \times 8$ -bit image, 3×3 filter mask)	4852	9024	1.86
FFT (256×256 image, IEEE single)	141824	242048	1.71
Backprojection (reconstructing a 256×256 grid using 256 rays over 256 angles, IEEE single)	1447116	1562886	1.08
Motion Estimation (352×288 CIF frame, ± 7 pixel full search, worst case)	144936	362763	2.50
Histogram ($256 \times 256 \times 8$ -bit)	4840	46080	9.52
RSA encryption (512-bit encryption per processor row)	472320	521472	1.10

Because of its remarkable processing power of up to 24 giga instructions per second (with respect to 16 bit operations) the main problem in the current KPROC implementation is the limited I/O-bandwidth. The reason is on the one hand the pin limitation and on the other hand the memory organisation. We can not do anything about the first but to cope with the latter it might be useful to introduce a memory hierarchy to gain performance by reuse of data. A level one cache could be integrated onto the KPROC chip. The cache must be global, i.e. every processor can access the cache for data exchange.

A second disadvantage of the KPROC is the fixed instruction set. Since parallel prefix operations can be executed fast only if their key instruction is contained in the instruction set, there are some problems that can not take advantage from this technique, e.g. [2]. A possible solution is the introduction of an instruction buffer. Instead of propagating each instruction immediately after execution it is stored for one (or a fixed number of) instruction(s) in the local memory of the processor. Now parallel prefix computations based on complex binary operations can also be implemented in a fixed number of instruction diagonals. Recently, a totally different type of ISA has been implemented as an accelerator unit for a videophone [20]. It will be interesting to observe, if new ideas concerning the processor design will come up from this approach.

REFERENCES

- [1] IEEE standard for binary floating point arithmetic, ANSI IEEE 754-1985, New York, 1985.
- [2] A. DITTRICH, H. SCHMECK, *Given's Rotation on the Instruction Systolic Array*, In: Wolf, G., Legendi, T., Schendel, U., eds.: PARCELLA '88, Mathematical Research, Vol. 48, Akademie Verlag, Berlin, 1988, pp. 340-346.
- [3] J. L. HENNESSY, D. A. PATTERSON, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman Publishers, 1996.
- [4] ISATEC SOFT-UND HARDWARE GMBH, *The ISATEC Parallel Computer Systola 1024*, Users Guide V. 3.0, 1998.
- [5] A. K. JAIN, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, 1989.
- [6] A. C. KAK, M. SLANEY, *The Fundamentals of Computerized Tomographic Imaging*, IEEE Press, New York, 1988.
- [7] M. KUNDE, H.-W. LANG, M. SCHIMMLER, H. SCHMECK, H. SCHRÖDER, *The Instruction Systolic Array and its Relation to other Models of Parallel Computers*, Parallel Computing, 7 (1988), pp. 25-39.
- [8] R. E. LADNER, M. J. FISCHER, *Parallel Prefix Computation*, Journal of the ACM, 27 (4) (1980) pp. 831-838.
- [9] H.-W. LANG, *The Instruction Systolic Array - a Parallel Architecture for VLSI*, Integration, the VLSI Journal 4, (1986) pp. 65-74.
- [10] H.-W. LANG, R. MAA, M. SCHIMMLER, *The Instruction Systolic Array - Implementation of a Low-Cost Parallel Architecture as Add-On Board for Personal Computers*, in High Performance Computing and Networking, Lecture Notes in Computer Science 797, Springer Verlag, 1994, pp. 487-488.
- [11] F. T. LEIGHTON, *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*, Morgan Kaufmann, 1992.
- [12] P. K. ROBERTSON, *Fast perspective views of images using one-dimensional operations*, Computer Graphics and Algorithms, 7 (2) (1987) pp. 47-56.

- [13] S. W. ROWLAND, *Computer implementation of image reconstruction formulas*, in Image Reconstruction from Projections, G. T. Herman, ed., vol. 32 in Topics in Applied Physics, New York, Springer Verlag, 1979, pp. 9-79.
- [14] M. SCHIMMLER, H.-W. LANG, *The Instruction Systolic Array in Image Processing Applications*, in Vision Systems: Sensors, Sensor systems, and Components, Europto 96, O. Löffel ed.: SPIE 2784, 1996, pp. 136-144.
- [15] H. SCHMECK, *A Comparison-Based Instruction Systolic Array*, in Parallel Algorithms and Architectures, Robert, Y. et al., eds., North Holland, 1986.
- [16] B. SCHMIDT, M. SCHIMMLER, H. SCHRÖDER, *Morphological Hough Transform on the Instruction Systolic Array*, in EuroPar'97, C. Lengauer, M. Griebl, S. Gorlatch, eds., Lecture Notes in Computer Science, 1300, Springer Verlag, 1997, pp. 798-806.
- [17] B. SCHMIDT, M. SCHIMMLER, H. SCHRÖDER, *Long Operand Arithmetic on Instruction Systolic Computer Architectures and its Applications to RSA Cryptography*, in EuroPar'98, Lecture Notes in Computer Science 1470, Springer Verlag, 1998, pp. 916-922.
- [18] B. SCHMIDT, M. SCHIMMLER, H. SCHRÖDER, *The Instruction Systolic Array in Tomographic Image Reconstruction Applications*, in PART'98, Adelaide, Australia, Springer Verlag, 1998, pp. 343-354.
- [19] B. SCHMIDT, *Techniques for Algorithm Design on the Instruction Systolic Array*, Ph.D. thesis, Loughborough University, UK, 1999.
- [20] B. SCHMIDT, M. SCHIMMLER, *A Parallel Accelerator Architecture for Multimedia Video Compression*, in EuroPar'99, Lecture Notes in Computer Science 1685, Springer Verlag, 1999, pp. 950-960.

Edited by: Yakov Fet and Peter Kacsuk

Received: June 3rd, 1999

Accepted: July 12th, 2000