Long Operand Arithmetic on Instruction Systolic Computer Architectures and Its Application in RSA Cryptography

Bertil Schmidt[1], Manfred Schimmler[2], and Heiko Schröder[3]

[1] Lehrstuhl für Informatik I, RWTH Aachen,
52056 Aachen, Germany,
bes@i1.informatik.rwth-aachen.de

[2] Inst. f. Datenverarbeitungsanlagen, TU Braunschweig,
38106 Braunschweig, Germany,
Schimmler@ida.ing.tu-bs.de

[3] Department of Computer Studies, Loughborough University,
Loughborough, LE11 3TU, England,
H.Schroder@lboro.ac.uk

Abstract. Instruction systolic arrays have been developed in order to combine the speed and simplicity of systolic arrays with the flexibility of MIMD parallel computer systems. Instruction systolic arrays are available as square arrays of small RISC processors capable of performing integer and floating point arithmetic. In this paper we show, that the systolic control flow can be used for an efficient implementation of arithmetic operations on long operands, e.g. 1024 bits. The demand for long operand arithmetic arises in the field of cryptography. It is shown how the new arithmetic leads to a high-speed implementation for RSA encryption and decryption.

1 Introduction

Instruction systolic arrays (**ISAs**) provide a programmable high performance hardware for specific computationally intensive applications [5]. 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. The ISA model is a mesh connected processor grid, which combines the advantages of special purpose systolic arrays with the flexible programmability of general purpose machines [3].

In this paper we illustrate how the capabilities of ISAs are exploited to derive efficient parallel algorithms of addition, subtraction, multiplication and division of long operands. The demand for long operand arithmetic arises in the field of cryptography, e.g. RSA encryption and decryption. Their implementations on Systola 1024 show that the concept of the ISA is very suitable for long operand arithmetic and results in significant run time savings.

The ISA concept is explained in detail in Section 2. Section 3 gives an overview over the architecture of Systola 1024. It is documented how the ISA has been integrated on an low cost add-on board for commercial PCs. The new ISA algorithms for long operand arithmetic are explained in Sections 4 to 6. The implementation of RSA based on these arithmetic routines is given in Section 7. Section 8 discusses its performance and concludes the paper.

2 Principle of the ISA

The basic architecture of the ISA is a quadratic $n \times n$ array of identical processors, each connected to its four direct neighbours by data wires. The array is synchronized by a global clock. The processors are controlled by instructions, row selectors and column selectors. 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 each clock cycle. In clock cycle $k+1$ processor $(i+1, j)$ and $(i, j+1)$ execute the instruction that has been executed by processor (i, j) in clock cycle k.

The selectors also move systolically through the array: row-selectors horizontally from left to right, column-selectors vertically from top to bottom (Fig. 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 very efficient solutions for a large variety of applications.

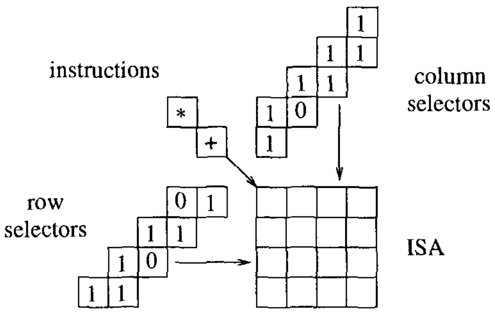

Fig. 1. Control flow in an ISA

Every processor has read and write access to its own memory. Besides that, it has a designated communication register (**C-register**) that can also be read by the four neighbour processors. Within each clock phase reading access is always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own C-register with the contents of the C-register of their neighbour. This convention avoids read/write conflicts and also creates the possibility to broadcast information across a whole row or column with one single instruction:

Row broadcast: Each processor reads the value from its left neighbour. Since the execution of this operation is pipelined along the row, the same value is propagated from one C-register to the next, until it finally arrives at the rightmost processor. Note that the row broadcast requires only a single instruction.

The **period** of ISA programs is the number of their instructions, which is $2n-2$ clock cycles less than their execution time. The period describes the minimal time from the first input of an instruction of this program to the first input of an instruction of the next program. In the following the period of ISA programs is used to specify their time-complexity. This is appropriate because they will be used as subroutines of much larger ISA programs in Section 7.

3 Architecture of Systola 1024

The ISATEC Systola 1024 parallel computer is a low cost add-on board for standard PCs. The ISA on the board is a 4×4 array of processor chips. Each chip contains 64 processors, arranged as an 8×8 square. This provides 1024 processors on the board. In order to exploit the computation capabilities of this unit, it is necessary to provide data and control information at an extremely high speed. Therefore, a cascaded memory concept, consisting of interface processors and board RAM, is implemented on board that forms a fast input and output environment for the parallel processing unit (see Fig. 2).

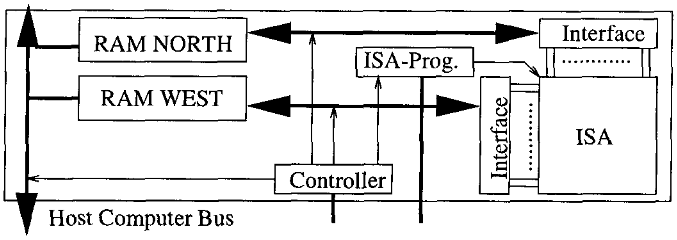

Fig. 2. Data paths in the parallel computer Systola 1024

4 Addition and Substraction of Long Operands

The difficulty of partitioning an addition in an array where only neighbours can talk to each other is the carry propagation. 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. Therefore, we use an **accumulation technique** based on the generate and propagate signals of a *carry-look-ahead-adder*.

If every processor knows, whether it has to propagate an incoming carry from the left to the right, 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 C-register. With the accumulation operation

(*) **if** `zeroflag` **then** `C:=C[WEST]`

all carry-bits travel to the processor one position left of their destinations. This is again because of the skewed instruction execution: 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[WEST]` all carries are at the place where they are needed. Note that the carry propagation requires only two instructions, although the carry-bits can travel over a distance of up to $n-1$ processors. 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.

The operands to be added are distributed over the rows of the ISA. If the length of the operands is 512/1024 bits, every processor in a row of 32 processors gets 16/32 bits. The LSBs are stored in the leftmost and the MSBs in the rightmost processor of the row. 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:=carry;
```

store the sum of **RA** and **RB** in **RSUM** and the carry-bit in **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. Thus, we can complete the long addition as pointed out above with

```
if (RSUM + 1=0) then C:=C[WEST]; RSUM:=C[WEST] + RSUM;
```

Table 1. Addition of two 24-bit numbers in a row with 6 processors

Processor	1	2	3	4	5	6	
A B	0101 0001	1100 0011	1010 0101	0010 1010	1101 1010	1000 0100	Every pr. gets 4 bits of A and B. LSB/MSB in leftmost/rightmost bit in pr. 1/6.
SUM	0100	1111	1111	1001	0000	1100	SUM:=A + B
C	1	0	0	0	1	0	C-register stores the carry-bit
Zeroflag	0	1	1	0	0	0	if (SUM+1=0) then zeroflag:=1 else :=0
C	1	1	1	0	1	0	C after prop.: if zeroflag then C:=C[WEST]
SUM	0100	0000	0000	0101	0000	0010	SUM:=SUM + C[WEST]

The **subtraction** works in principle in the same way as the addition, using **RDIFF:=RA-RB**, the propagation condition **if (RDIFF=0)** and the carry-bit subtraction **RDIFF:=RDIFF-C[WEST]**.

Due to the systolic control flow the n-bit addition/subtraction is possible using $O(n)$ ISA processors with constant period. The complete 512/1024-bit addition/subtraction in one Systola 1024 processor-row requires only 5/8 instructions. However, having 32 processor-rows, 32 different additions/subtractions can be calculated in parallel in the same time.

5 Multiplication of Long Operands

The idea behind the multiplication on the ISA is related to the *school method for integer multiplication*: One operand is cut into pieces and all pieces are multiplied with the other operand in parallel. The results are then shifted with respect to each other in an appropriate way and added to form the final result.

The ISA-program for the $m \times n$-bit ($m = 16 \cdot p$, $n = 16 \cdot q$) multiplication in each processor-row is explained below. Every processor stores 16 bits of the first operand in register **RA**, the least significant 16 bits in the leftmost and the most significant 16 bits in the rightmost processor. The complete second operand is stored in $\mathbf{RB}_0, \ldots, \mathbf{RB}_{q-1}$ in each processor. At the end the most significant m bits of the result will be distributed over the rows in RP_q. The least significant n bits will be stored in the first processor of each row in $RP_0, \ldots, RP_{q-1}$.

```
$RP_0..RP_q$:= RA · $RB_0..RB_{q-1}$ // compute $16 \times 16 \cdot q$-bit multiplication
C:=$RP_0$                         // load least significant result in C-register
for i:=1 to q do                  // add $RP_0..RP_q$ along processor-row in $q$ steps:
begin                                add 16 bit more significant word in C[EAST]
    $RP_{i-1}$:=C                        and $RP_i$. Sum is stored in C and $RP_i$
    C:=C[EAST]+$RP_i$+carry              This addition is repeated up to the most
end                                  significant word
$RP_q$:=C                         // store most significant result
C:=carry
if $RP_q$+1=0 then C:=C[WEST] // propagate the carry-bit of the last addition
$RP_q$:=C[WEST]+$RP_q$                   as described in Section 4
```

The implementation on Systola 1024 requires 223/507 instructions for one single $512 \times 512/1024 \times 1024$-bit multiplication in one row of 32 processors.

6 Division of Long Operands

Division can be efficiently reduced to multiplication and subtraction by using the *Newton-Raphson-method.* For a value B the reciprocal value is computed by an iteration that converges rapidly towards $\frac{1}{B}$. To achieve a fast convergence ($2m$ bits precision in m iteration steps), $0.5 \leq B < 1$ must hold for B. The iteration is given by the equations

$$x_0 := 1 \quad , \quad x_{i+1} := (2 - B \cdot x_i) \cdot x_i \tag{1}$$

Obviously, any division $Q = \frac{A}{B}$ can be computed by multiplying A with $\frac{1}{B}$. For division on the ISA we assume B to be in the range between 0.5 and 1. If this is not the case, B is initially shifted in the ISA such that the most significant one is the MSB of the rightmost word of B.

In order to understand the choice of intermediate operand lengths in our division method it is useful to consider the behavior of the convergence in the Newton-Raphson-algorithm. Let x_i differ from the precise value of $\frac{1}{B}$ by some ε. Then the following holds for x_{i+1}:

$$x_{i+1} = (2 - B \cdot x_i) \cdot x_i = (2 - B \cdot (\tfrac{1}{B} - \varepsilon)) \cdot (\tfrac{1}{B} - \varepsilon) = \tfrac{1}{B} - B \cdot \varepsilon^2 \tag{2}$$

Since $B < 1$, this means that the precision of x_{i+1} (i. e. number of correct leading digits) is at least double the precision of x_i. This means that we can restrict the operand lengths to 2^i bits for B and x_{i-1} while computing the i^{th} iteration x_i. E.g. the first four iterations can be performed in one processor, since the required precision of the operands is ≤ 16. Only the last iteration step needs full precision of m bits. The subtractions and multiplications are performed efficiently on the ISA as explained above.

7 Parallel Implementation of RSA Encryption

For obtaining a high-speed implementation of RSA encryption a fast *modular exponentiation* ($M^e \bmod N$) is necessary. To achieve a sufficient degree of security the operand-lengths have to be relatively large (currently ranging from

512 up to 2048 bits). Modular exponentiation can be accomplished by iterating *modular multiplications* using the *square-and-multiply algorithm* [2].

The difficulty in implementing an efficient modular multiplication ($A \cdot B$ mod N) is the modular reduction step. In the case of RSA encryption the modulus N is known in advance to each modular multiplication. Thus, the precomputation of $\frac{1}{N}$ requires only a negligible amount of work. Now, the division x div N can be replaced by the more efficient multiplication $x \cdot \frac{1}{N}$. Of course, we cannot produce $\frac{1}{N}$ precisely. We instead produce a number μ such that $x \cdot \mu$ is close enough to x div N; i. e. the result needs only a small correction by at most 4 subtractions with N. Thus, the modular multiplication for RSA can be implemented efficiently on the ISA by three multiplications and at most four subtractions.

One single n-bit modular multiplication for $n = 512/1024$ requires 581/1709 instructions in one processor- row of Systola 1024.

8 Performance Evaluation and Conclusions

The performance of the parallel implementation is compared with an optimized sequential implementation on a PC (see Table 2). The current ISA-prototype Systola 1024 is a machine based on technology far from being state-of-the-art. Extrapolating to technology used for processors such as the Pentium II 266 MHz would lead to a speedup of the ISA by a factor of at least 80. Thus resulting in encryption/decryption speeds of more than **500 Kbit/s** for 1024-bit RSA (without taking advantage of the *Chinese Remainder Theorem*). Such performance can be expected from the already announced next generation Systola board **Systola 4096**.

Table 2. Performance of RSA encryption

	Systola 1024	Pentium II 266	Speedup
512-bit RSA with full length exponent	128 KBit/s	18 KBit/s	7
1024-bit RSA with full length exponent	56 KBit/s	6 KBit/s	9

We have presented ideas for fast implementations of addition, subtraction, multiplication and division on operands that are too long to be handled within one processor. We take advantage of the ability of the ISA to implement accumulation operations extremely efficiently. We have used the results for finding a high-speed instruction systolic implementation of RSA encryption and decryption. This leads to efficient software solutions with respect to performance and hardware cost. We also used the long operand arithmetic routines for the implementation of a prime number generator for RSA keys on Systola 1024[1].

References

1. Hahnel, T.: The Rabin-Miller Prime Number Test on Systola 1024 on the Background of Cryptography. Master Thesis, University of Karlsruhe (1998)
2. Knuth, D.E.: The Art of Computer Programming: Seminumerical Algorithms. Volume 2, Reading, Addison-Wesley, second edition (1981)
3. Kunde, M., et al.: The Instruction Systolic Array and its Relation to other Models of Parallel Computers. Parallel Computing **7** (1988) 25–39

4. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Comm. of the ACM **21** (1978) 120–126
5. Schmidt, B., Schimmler, H., Schröder, H.: Morphological Hough Transform on the Instruction Systolic Array. Euro-Par'97, LNCS 1300, Springer (1997) 798–806