http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (3): 416-421, 2006
ISSN 1812-5638
© 2006 Asian Network for Scientific Information

Hardware Implementation of Instruction Level Parallel Architecture Incorporating Special Functional Units for Image Processing Algorithms

M. Kannan and S.K. Srivatsa
Department of Electronics Engineering, MIT Campus, Anna University,
Chromepet, Chennai-44, Tamil Nadu, India

Abstract: Parallel processing is an efficient form of information processing with emphasis on the exploitation concurrent events in computations. Considering a sequence of assembly instructions for a specific problem it is found that many of the consecutive instructions are independent of each other, without any data dependencies between them. This work exploits such situations and it executes pairs of instructions, which do not have dependencies between them, on two different processing elements, thus enhancing the speed of operations. It is not always true that any two instructions taken from a sequence of instructions could go in parallel. The various types of dependencies that exist among the instructions are the bottleneck in executing instruction in parallel. The various possible data dependencies and control transfers are handled so that most of the instructions are run pairs. The ILP(Instruction Level Parallelism) architecture designed here is to be used for image processing applications. Since specific hardware solutions are always faster that their software counterparts and we have dedicated hardware units for most frequently used image processing problems of finding DFT and DCT. The proposed architecture improves the performance with a speed up factor of more than 1.5 with lesser data dependencies, we can get a higher speed up factor, upper bounded by the value of 2 by the Amdahl's law.

Key words: ILP architecture, DCT, DFT, parallel architecture

INTRODUCTION

Parallel processing has been one of the hottest ideas of computing in which information processing is done with emphasis on the exploitation of concurrent events in computations. Not only architectures, but also compilers and operating systems have been striving for more than two decades to extract and utilise as much parallelism as possible to improve the speed of the systems (Hwang, 1990; Lilja, 1994).

Basically, parallelism is used in two different contexts, available parallelism and utilised parallelism. The available parallelism is present inherently in the problem solutions thus making the implementations easily parallel. The parallelism could be either functional parallelism, which comes from the logic of the solution, or data parallelism, which is due to the type of data structures used (like vectors and matrices).

There are various levels in which the inherent functional parallelism of the solution to the problem can be utilised to complete the solution in a fast manner. The different levels of parallelism are Program level (Coarse-grained parallelism), Procedure level (Middle-grained parallelism), Loop level (Middle-grained parallelism) and Instruction level (Fine-grained parallelism). Among these levels, the highest level of parallelism possible is in the program level, which requires the device of parallel algorithms. An optimising parallel compiler achieves procedure level parallelism and loop level parallelism. By exploiting concurrency between consecutive instructions, which is the processing in the instruction level, an immense speed up can be got. This fine grained processing gives the maximal through put without making changes to the original algorithm.

The implemented ILP processor has instruction set which is a subset of the DLX instruction set (Patterson and Hennessy, 1990), which supports the special instructions used specifically for image processing applications, DCT and DFT instructions. DLX is a simple load store architecture, which had been designed for pipelining efficiency. In order to achieve higher speed up, these frequently used instructions are given to dedicated hardware modules. The performance of the system improves very much when these dedicated hardware units are added.

Corresponding Author: M. Kannan, Department of Electronics Engineering, MIT Campus, Anna University, Chromepet, Chennai-44, Tamil Nadu, India Tel: +91-44-22237276/233

This work proposes the design of a super scalar, instruction level parallel architecture. There are two processing elements and dedicated hardware for DCT and DFT (Schlansker *et al.*, 1997, Buehrer and Ekanadham, 1987). The memory is divided into Data memory and program memory thus following the Harvard Architecture of Memory Organisation. For each clock tick, two instructions are fetched from the Program memory and a data dependency check is done on them (Sohi, 1990). If they are independent and both their data are ready, then they are issued to two different processing elements and hence these two instructions run in parallel. The dedicated units can also run in parallel with these two processing elements.

IMPLEMENTATION ILP ARCHITECTURE

The instruction level parallel architecture designed here is a Super Scalar processor (IEEE, 1998 a,b). Two instructions are issued per clock since there are two pipelines in the design. Parallelism between consecutive instructions in the program sequence is exploited in this design. The overall architecture of the processor is shown in Fig. 1. It consists an instruction pre fetch unit that fetches instructions from an interleaved instruction memory, a scheduler which examines the instructions (Arvind and Nikhil, 1986; Pratt, 1976), resolves data dependencies among them and prepares them for execution, a tagged data memory and tagged register file with tags associated with each word to indicate the presence or otherwise of valid data , a set of data structures used by the scheduler, namely the Ready Queue, the Deferred Instruction Queue and the Tag Unit, a set of Processing Elements (PEs) to execute the instructions for which data are available. These PEs can be homogeneous or each could be a specialized functional unit. This design implements the lower order interleaved memory organisation. The memory modules are byte addressable. The processor is 32 bit processor with a register file of 32 registers, supports minimal number of addressing modes to make the design as simple as possible. The addressing modes supported are immediate addressing, Register addressing and Based Indexed addressing. The memory is organized as interleaved memory.

To facilitate simplicity again, DLX basically supports only 3 formats of instructions. They are I Type instructions, R Type instructions and J Type instructions and their formats are given in Fig. 2.

DLX supports the list of simple operations that is supported by almost all the processors. The instructions may be broadly classified as Load Store, ALU, Branches

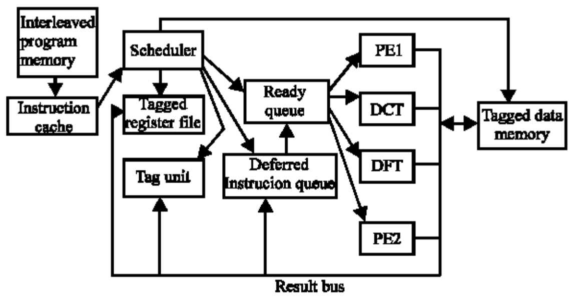


Fig. 1: Overall instruction level parallel architecture

I Type:

Opcode (6)	SRC 1(5)	SRC 2(5)	Immediat (16)

R Type:

Opcode (6)	SRC 1(5)	SRC 2(5)	DST (5)	Function (11)

J Type:

Opcode (6)	Offset (26)

Fig. 2: DLX instruction formats

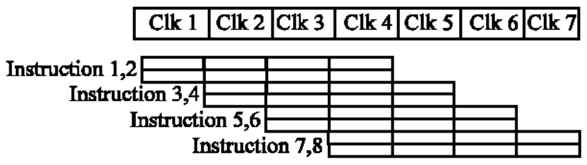


Fig. 3: Pipelined instruction execution in the ILPA

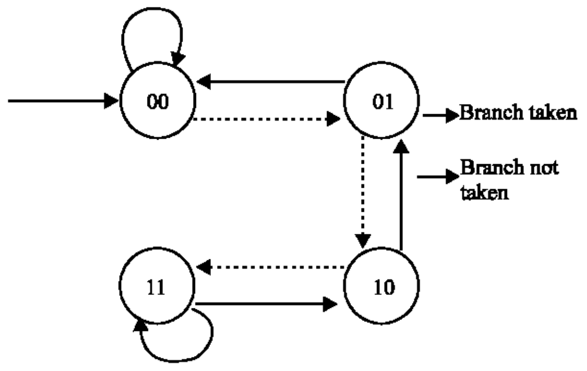


Fig. 4: Branch prediction state diagram

and Jump instructions. For the ILPA, there is one more type of instruction possible, which is the special instruction type for the dedicated hardware units. The opcodes are given in Table 1.

In a program, instructions sometimes depend on each other in such a way that a particular instruction cannot be executed until a preceding instruction or even

Table.1: Opcode of DLX processor (Main Opcodes)

	00	01	02	03	04	05	06	07
00	Special		J	JAL	BEQZ	BNEZ		
08	ADDI		SUB					
10			JR	JALR	ANDI	ORI	XORI	NOP
18	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI		
20	LB	LH	LW	LF	LD	LBU	LHU	LHI
28	SB	SH	SW	SF	SD	SLLI	SRLI	SRAI
Special opcodes (Main Opcode = 00)								
00	01	02	03	04	05	06	07	
00	ADD		SUB		AND	OR	XOR	
08	SEQ	SNE	SLT	SGT	SLE	SGE		
18	SLL	SRL	SRA		DFT	DCT		

two or three preceding instructions have been executed. There are three types of dependencies possible. They are Data dependencies (subsequent instructions are dependent on each other because of data), control dependencies (a conditional jump statement is executed) and resource dependencies (the same resource is being used by both instructions). All these dependencies are taken care appropriately in the scheduler design (Sohi, 1990; Arvind and Iannuei, 1986; Buehrer and Ekandham, 1987).

The ILP architecture is a pipelined architecture like DLX. This design implements a four-stage pipeline, which is different from the standard DLX architecture. The memory access stage in the standard DLX design is removed in this design, since it is considered redundant. The four stages of the pipeline are fetch stage, decode stage, execute stage and write back stage.

As shown in the Fig. 3, initially, in the first clock, the first two instructions are fetched. In the next clock, they are decoded and the fetch unit fetches the next two instructions. In the next clock, the first two instructions are executed in the corresponding execution units, the instructions 3 and 4 are decoded and instructions 5 and 6 are fetched. In the fourth clock, the results of instructions 1 and 2 are written back, instruction 3 and 4 are executed, instruction 5 and 6 are decoded and the instruction 7 and 8 are fetched.

The ILP architecture design implements the branch prediction logic as given below in Fig. 4. Whenever a conditional branch instruction is encountered, it initially assumes that the branch would be taken and starts issuing the instruction from the 'target address' sequence of the program to the instruction pipeline, rather than the 'sequential address' of the program. There are two bits associated with each and every branch instruction that occurs in a program out of which the first bit says if the branch would be taken or not. '0' indicates the branch would be taken and '1' indicates it would not be taken.

When the execution of the branch instruction is over and if the branch is taken as predicted, then the bits

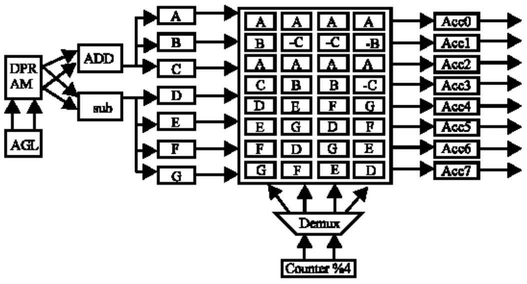
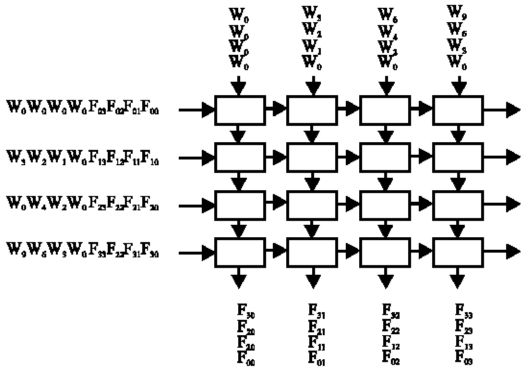


Fig. 5: Architecture of the DCT processor



$W_0 W_0 W_0 W_0$
$W_3 W_2 W_1 W_0$
$W_6 W_4 W_2 W_0$
$W_9 W_6 W_3 W_0$
$W_0W_0W_0W_0F_{03}F_{02}F_{01}F_{00}$
$W_3W_2W_1W_0F_{13}F_{12}F_{11}F_{10}$
$W_6W_4W_2W_0F_{23}F_{22}F_{21}F_{20}$
$W_9W_6W_3W_0F_{33}F_{32}F_{31}F_{30}$
$F_{30} F_{20} F_{10} F_{00}$
$F_{31} F_{21} F_{11} F_{01}$
$F_{32} F_{22} F_{12} F_{02}$
$F_{33} F_{23} F_{13} F_{03}$

Fig. 6: Systolic array representation of DFT

remain at "00". If the branch is not taken, but was predicted that it would be taken, then the bit transition occurs. So, the current status of the branch is used as history to predict the future branches.

Consider the following example, which adds r5 6 times into r1. In the first clock, 1 and 2 are fetched. In clock 2, they are decoded and 3 and 4 are fetched. In the third clock, 1 and 2 are executed, 3 and 4 are decoded and in the mean time, the fetch unit is free which now uses the branch predicted 'target address' and fetches the instructions 1 and 2 again.

- Loop: Add r1, r1, r5
- Addi r6, r6, 1
- Slei r4, r6, 6
- Bnez r4, loop

The sequence proceeds in the above said manner for 6 times. When the instruction 3 resets the register r4 when r4 becomes greater than 6, the branch would not be taken, which would have been predicted to be taken. In that case, instructions 1 and 2 would have gone into the pipeline already. So, we have to flush the pipeline this time and the status for this branches moves from "00" to "01".

When the different units of ILP Architecture are considered, the Scheduler takes care of scheduling instructions onto available processing elements by coordinating with other units. This includes preparing the instructions and putting in the Ready Queue (RQ) when the data is available. If data is not available the instruction is transferred to Deferred Instruction Queue (DIQ) and transferred from DIQ to RQ once the data is available.

The Tag unit acts as a reservation station for the register file. Whenever a write to a register is initiated, an instance for that register is made in the tag unit. Thereafter, instead of addressing the register by its number, it is addressed by its tag number or the instance number. When one or more of the input operands are not ready at the time of decoding, then that particular instruction has to be deferred. This data structure DIQ is used to store the deferred instructions. The architecture has an extensive set of registers organised as a register file with 32 GPR s. Each Register in the register file has a busy bit associated with it. If the busy bit is set, this indicates that a write into the register has been initiated and an instance created in the tag unit. If it is not set, then the data in the register is valid and that data could be read by the instructions.

The Ready Queue maintains the list of all instructions, which are currently ready to get executed. The Ready Queue gives the instructions to be executed to the scheduler and the scheduler schedules the instructions to the processing elements when it finds the PEs free. On the rise of every clock, the scheduler takes a look at the status of the execution units. If one or more of them are free, then the instructions from the RQ are scheduled to the execution units and the locations in the RQ are made free for further instructions to come.

Related to the Data Dependency Check if the busy bit is set for particular register, the tag unit is searched for the latest tag entry of that register (Espara and Valero, 1997). The corresponding tag index is entered as tag number of the source operand to the DIQ along with the instruction. If the busy bit was reset for the corresponding source register, the data is retrieved from the register file and stored in the corresponding slot of the DIQ. This is repeated for both the source operands. That is, if for one of the sources, the corresponding register has valid data item, this data is stored along with the instruction in the DIQ until the other register gets that data. If both s1 and s2 are available, the instruction is scheduled to any available computational unit and executed. In case of the destination register, a free tag is allocated to this register indicating that an instance has been created for generating this value.

Processing element: The core of any architecture is its processing element. The ILPA processes 32 bit operands. The processing element of the ILP architecture consists of Carry Look Ahead adder, Carry Save Multiplier, Logic units, Comparator, Barrel Shifter etc. (Patterson and Hennessy, 1990). The other two functional units are the DCT and DFT (Pratt, 1976).

Discrete cosine transform: The Discrete cosine transform is a sequence of multiply-accumulate operations as given in the equation. The 2D DCT is a separable function and can be easily achieved by two 1D DCTs operating in a pipelined manner. The fast algorithm for the 8 point DCT is discussed below. The one-dimensional DCT is expressed mathematically as follows.

$$Y(k) = \sqrt{\frac{2}{N}}C(k)\sum_{i=0}^{N-1} x(i)\cos\frac{(2i+1)k\pi}{2N}$$

where:

$$C(k) = \frac{1}{\sqrt{2}} \quad \text{when k=0}$$
$$= 0 \text{ Otherwise}$$

For an eight-point DCT, the matrix shown here gives the transform coefficients.

$$\begin{pmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \\ Y(5) \\ Y(6) \\ Y(7) \end{pmatrix} = 1/2 \begin{pmatrix} A\ A\ A\ A \\ B\ C - C\ B \\ A - A - A\ A \\ C - B\ \mathbf{B} - C \\ D\ E\ F\ G \\ E - G - D - F \\ F - D - G\ E \\ G - F\ E - \end{pmatrix} \begin{pmatrix} X(0)+x(7) \\ X(1)+x(6) \\ X(2)+x(5) \\ X(3)+x(4) \\ X(0)+x(7) \\ X(1)+x(6) \\ X(2)+x(5) \\ X(3)+x(4) \end{pmatrix}$$

Where,

$A = \cos\frac{\pi}{4}$; $B = \frac{\pi}{8}\cos$; $C = \sin\frac{\pi}{8}$

$D = \frac{\pi}{16}\ \cos\frac{3\pi}{16}$; $E = \cos$; $F = \sin\frac{3\pi}{16}$; $G = \sin\frac{\pi}{16}$

The block diagram of 1D DCT processing unit is shown in the Fig. 5. This consists of the multiplier unit, the register unit and the pre processing adder and subtractor units, with a post processing accumulator unit.

The input data from the Dual ported RAM is accessed in blocks of 8×8. Two values can be accessed simultaneously from the DPRAM. The first two values accessed are x(0) and x(7) where 0 and 7 indicate the position of the pixel in the current row under consideration. The address generation logic for accessing these values in a specific order as required by the algorithm is given as follows.

The values fetched from the DPRAM are given to the adder and subtractor. The algorithm requires the sum and difference of the two value pairs x(0) and x(7), x(1) and x(6), x(2) and x(5) and x(3) and x(4) for the computation of DCT coefficients Y(0),Y(1),... Y(7) as indicated by the matrix. The adder computes the sum of the two values and the subtractor computes their difference.

The output of the adder should be multiplied with the coefficients A, B and C. Parallel multipliers are used to do this. The algorithm also requires the difference multiplied by the coefficients D, E, F and G, which is also done simultaneously by using 4 multipliers.

Discrete fourier transform: The DFT is also a set of multiply-accumulate operations. The DFT of a two dimensional signal is given as follows:

$$F(u,v)=\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{vy}{N}\right)}e^{-j2\pi\left(\frac{ux}{N}\right)}$$

This mathematical function can be split into two separate MAC operations.

$$F'(x,v)=\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{vy}{N}\right)}$$

and

$$F(u,v)=\sum_{x=0}^{N-1} F'(x,v)e^{-j2\pi\left(\frac{ux}{N}\right)}$$

The multiplied values are accumulated into a register in the Processing element in the first part of the DFT (finding F' (x, v)) and in the next part, the accumulated values are multiplied and outputs are taken (finding F (u, v)). The DFT block is designed based on the Systolic Array topology.

The PEs are arranged in a 4×4 2D array. When the clock goes high, the process of multiply-accumulate begins. Initially all the registers of all the PEs are cleared. The control input is held low during the first half of the operations. The weights or the twiddle factors for the DFT are fed as vertical inputs whereas the horizontal inputs are the input matrix values.

Table 2: Synthesis results of some units of ILPA

Module	Cell count Combinational	Sequential	Total	Maximum speed
DCT Unit	965	192	1157	22.3 MHz
DFT Unit	124	16	140	34.6 MHz
Adder Unit	215	0	215	84 MHz
Multiplier Unit	1289	0	1289	45.9 MHz
Comparator Unit	177	0	177	29.4 MHz

For the first 4 clocks, the input values are multiplied with their corresponding weights and are stored into the registers Rij. The input value for a row is passed on to all the processors in that row simultaneously and the weight for a column is also broadcasted to all the processors in the column in the same way. When these 4 clocks are counted, the control signal is made high.

RESULTS

The blocks of the ILPA are simulated using Modelsim and the results are obtained. The above blocks are synthesised using Leonardo Spectrum and the results are given in Table 2.

The performance of the ILPA is measured and compared with a single processing systems for standard Matrix Multiplication. The number of instructions executed is 25. Using single processor systems it would have taken 25 clocks to run the same code of matrix multiplication. Since the ILPA takes advantage of parallelism, it could complete the same job with 16 clock cycles, with all data dependencies resolved and producing the same output. Hence a speed up of 1.56 is achieved for this particular program. If we write longer programs and with lesser data dependencies, we can get a higher speed up factor, upper bounded by the value of 2 by the Amdahl's law.

DISCUSSION

The ILP architecture designed here exploits the inherent parallelism in the program and appropriately speeds up the execution. When the design is used for the specific application of image processing, the dedicated units come to help a lot. The architecture proposed here can also be used for any other application not necessarily, Image processing, since all it looks for is the independence of consecutive instructions. The architecture can improve the performance with a speed up factor of maximum 2. For almost all programs, it easily exceeds 1.5.

As mentioned earlier, this study is easily extensible to any specific field. For example, if it is going to be used for a data base specific applications, the same factors of speed up could be gained easily. Since the data base applications will hardly be using the DCT and DFT units, they can be replaced with the functions these applications often use, like sorting and searching. The units can also be replaced by dynamically reconfigurable units, which can do specific hardware functions, giving speed and also be reconFigd, thus giving programmability. Thus reconfigurable hardware units give us a very good hardware-software balance.

REFERENCES

Arvind, R. and S. Nikhil, 1986. Executing a program on the MIT Tagged Dataflow Architecture. Memo 302, MIT Lab, Comput. Sci., Cambridge MA.

Arvind, R. and R.A. Iannuci, 1986. Two fundamentals Issues in Multiprocessing". Memo 226-6, Mit Lab, Computer Sci., Cambridge, MA, pp: 1-21.

Buehrer, R. and K. Ekanadham, 1987. Incorporating data flow ideas into Von Neumann processor for Parallel execution: IEEE Transaction on Computers, 36: 1515-1521.

Espasa, R. and M. Valero, 1997. Exploiting Instruction and Data level parallelism. IEEE Micro, Vol. 17.

Hwang, K., 1993. Advanced Computer Architecture. McGraw-Hill.

IEEE, 1998. Signal Processing Magazine, Special Issue on Multimedia Processors, Vol.15, No. 2.

IEEE, 1998. Symposium on FPGAs for Custom Computing Machines, Napa Valley, California.

Lilja, B.J., 1994. Exploiting the parallelism available in loops. IEEE Computer, 27: 13-26.

Patterson D.A. and J.I. Hennessy, 1990. Computer Architecture A Quantative Approach. Morgan Kaufmann Publishers, Inc San Mateo, California.

Pratt, W., 1991. Digital Image Processing. Springer-Verlag.

Schlansker, M. and T.M. Conte *et al.*, 1997. Compilers for instruction level parallelism. IEEE Computer.

Sohi, G.S., 1990. Instruction issue logic to high performance, interruptible, multiple functional unit, pipelined computers. IEEE Transactions on Computers, 39: 349-358.