MODEL SIMULATION AND PERFORMANCE EVALUATION FOR A VLIW ARCHITECTURE

Cornel Popescu, cpop@csit-sun.pub.ro
Francisc Iacob iacob@apolo.cs.pub.ro
Department of Computer Science
POLITEHNICA University of Bucharest
313 Splaiul Independentei Street
Bucharest - Romania 77206

Abstract

The paper presents some of the most recent improvement techniques used in simulation and performance evaluation of a VLIW architecture. In the first part of the paper a functional description based on a typical model for a VLIW architecture is given; also, is made a comparison between VLIW execution and superscalar execution, from point of view of the processor program execution rate. Then, a Verilog HDL description program was proposed in order to make a VLIW processor simulation. Finally, a performance evaluation is made by comparing the performance of PowerPC processor with the VLIW processor performance.

The conclusions show the great importance of parallelism increasing for VLIW processing structures.

Key words: VLIW, ILP, pipelining, fill-unit, Verilog, HDL, instruction rate.

Introduction

VLIW (Very Long Instruction Word) processors are a logical extension of superscalar RISC processors. VLIW ideas came from the horizontal (i.e. parallel) microcode way back in computing's earlier days and from the fast supercomputers such as the CDC 6600 and IBM 360/91. VLIW represents an advanced way of internal parallelism increasing in microprocessor designs.

More things can be done to make a microprocessor run faster. First, we can speed up the clock frequency by inventing ever-faster (i.e. smaller) fabrication processes. Also, we make the processor perform more operations during each clock cycle; this requires adopting architectural features such as deep pipelines to keep the silicon busy. Performing more operations per cycle also implies building multiple functional units on the same chip as in the case of the superscalar processors. But, a superscalar processor slows down at about five or six instructions dispatched per cycle.

Another alternative is to assign software do all the scheduling, and that's precisely a VLIW design does; a smart compiler can examine a program, find all instructions with no dependencies, string them together in very long batches, and execute them concurrently on an equally big array of functional units [1].

In hardware terms a VLIW processor is very simple, consisting of little more than a collection of functions units (adders, multipliers, branch units, etc.) connected by a bus. The processor has register files. The instructions and data are taken from internal caches.

A VLIW processor, with multiple functional units, fetches, from the instruction cache, a very long instruction word, containing several primitive instructions, and dispatches the entire VLIW for parallel execution [8]. A smart compiler is required in order to identify the operations that can be executed in parallel. By removing complexity from the hardware, the designers create simple high performance processors. The objective of VLIW is to eliminate the complicated instruction scheduling and parallel dispatch that occur in most modern microprocessors.

A VLIW compiler packs independent operations into very long instruction words in a way that uses all the functional units efficiently during each cycle. The compiler discovers all the data dependencies, then determines how to resolve these dependencies. This process differs from a superscalar CPU, which uses special hardware to determine dependencies dynamically at run time.

To avoid parallel limitation, increasing path length, excessive code motion, pipeline stalls because of branches, and many other problems which limit the performance of a VLIW processors, hardware and software scheduling techniques were proposed. In order to take more parallelism with a lower price, it is better to use a combination among software and hardware techniques.

Thus, there are software scheduling techniques like Software Pipelining [5], Trace Scheduling [4], Enhanced Pipeline Scheduling [2] and other specific techniques [7].

Also, there are hardware scheduling techniques like Fill-Unit mechanism [3].

A typical model for a VLIW processor [8] is presented in the following figure (Fig. 1):

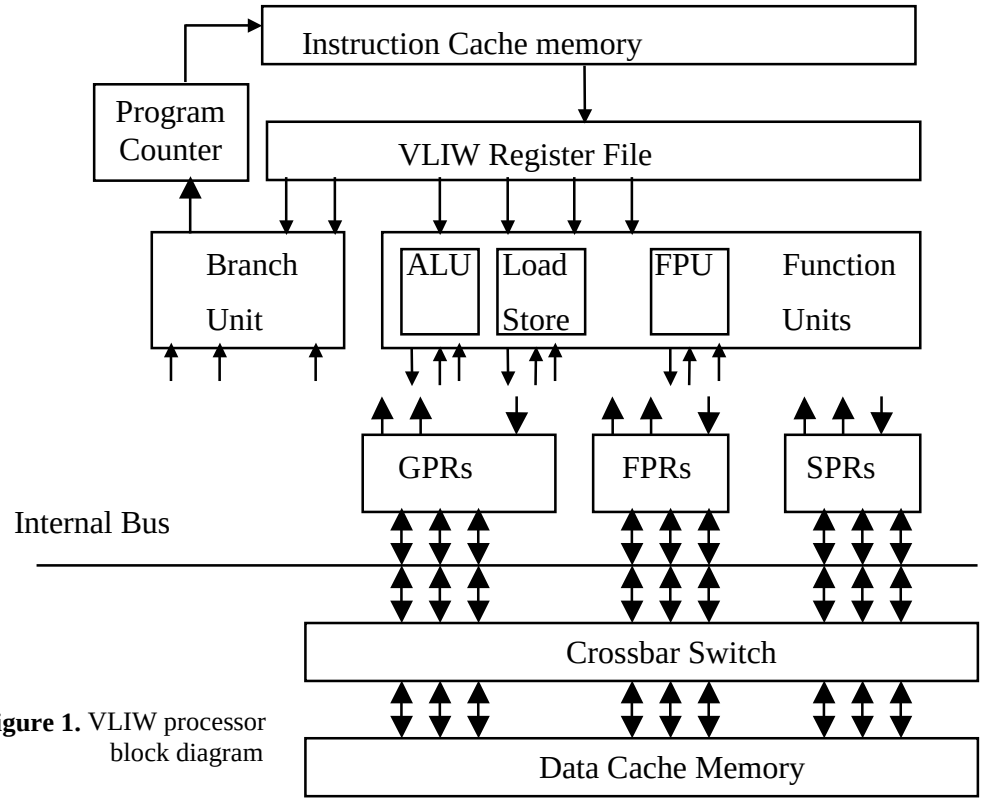

Figure 1. VLIW processor block diagram

VLIW architecture is appropriate for sequential programs execution [7], as well as for vectorized algorithms. In order to execute a program on a VLIW architecture, VLIW scheduling techniques extract the parallelism from the program code and they schedule the independent parallel operations, packed into the very long instruction words, for execution on multilpe functional units.

Processor architectures can be simulated using high-level hardware description languages (Verilog, VHDL) [8;9]. The paper presents a VLIW architecture described in Verilog HDL; the Verilog simulation program is given by the program structure.

The results of a recent simulations, like the I.B.M. T.J. Watson Research Center is, show that a VLIW processor executes 4-5 times faster than an advanced superscalar processor. The same result is obtained by applying performance evaluation calculations [6], like those which are proposed in the paper.

VLIW Architectural Model Simulation

Through an hierarchical approach, a complex hardware system can be described, on more levels of abstracting, in all respects: behavioural, structural, and physical. This is made by using powerfully formalized hardware description languages, like is Verilog HDL [9], and which are used into an automata design environment.

Further on it is presented the structure simulation program in Verilog HDL for the VLIW processor which on it was made the execution of the sort program). As a next research-design it is to implement the VLIW processor using Xilinx technology. Also, it is to write for this processor an optimized compiler appropriate for packing the programs in VLIW code.

The structure of the Verilog program:

1. **Statements**
   - data paths;
   - memories, registers;
   - flags, control signals; …
2. **Definings**
   - VLIW instruction fields;
   - operation codes, operation functions;
   - condition codes; …
3. **Functions and tasks**
   - condition flag calculation;
   - operation type decoding and dispatching;
   - operation type execution;
   - instruction address translation;
   - data address translation; …
4. **Tasks**
   - **reads** VILW instructions from the instruction cache; there are set work semaphores;
   - **decodes** parallely the operations packed into VLIW instruction **and dispatches** parallely these operations towards the execution units (for each operation, functional tasks for decoding and dispatching operation types are called: ALU

operations, FPU operations, load/store operations, branch and jump operations; there are set work semaphores):
- **if** the operations i:j are not "nop", "halt", or "wrong code"
  dispatch the operations to functional units (ALU, FPU, LSU, BPU);
  otherwise executes "nop", "halt", or "fault code";

- **executes** parallely the unpacked operations on the execution units (for each execution unit, functional tasks for execution operation types are called); there are set work semaphores);
- **writes** parallely the results of the executed operations, to registers ot to the memory;
- **sets** the pointers of the instruction and result queues;
- **displays** general purpose register (GPR) content, floating point register (FPR) content, and data cache mamory;
- **displays** waves;
- **resets** the system;

5. **Initial functional blocks**
   - **"setup"** and **"program load";**
   - **"reset"** system;
   - **begins** clock system;
6. **Functional blocks with continue execution** for the pipeline stages
   - **executes** on the decrease edge clock:
     - **while** the instruction queue is not full, **executes readings** of VLIW instructions;
     - **while** the instruction queue is not empty, **executes parallel decode and dispatch** of the operations from VLIW instruction;
     - **while** there is at least one decoded and dispatched VLIW instruction, **execute in parallel** the VLIW instruction operations on the execution units;
     - **while** there are results of the execution of the operations from at lest one VLIW instruction, **writes in parallel** the results to GPRs, FPRs , and data cache memory;
   - **verifyes** on the decrease edge clock:
     - **if** misspredict branch
       flush instruction queue **and resume reading** of VLIW instructions from the correct branch address;
       otherwise sets queue pointers;
     - **if** there is a "halt" operation
       stop the system;

End.

VLIW Performance Evaluation

The various configurations, especially the parallelism, they are very important to improve performance. The using VLIW techniques makes performance improvement by parallel execution (in pipeline technique) of a multiple instructions per clock cycle. Also, the performance increases by using hardwired control, Harvard architecture (cache memory separate for instructions and data), branch prediction etc.

They will be calculate the Instruction Execution Rate (I_R) in MIPS for a VLIW processor, using the Bus Bandwidth (B). In order to do this we propose some formulas [6] to calculate the performance taking account on the VLIW processor structure. We will make a comparison from point of view of the performance obtained for a VLIW processor and that which is obtained on an actual superscalar processor (such should be PowerPC).

Bus Bandwidth in MBytes/sec is defined by the amount of work transferred to and from memory in the unit time. We can calculate *B* as:

$$B = u_b \times \frac{n_b}{n_c \times t_c} \qquad (1)$$

where: u_b is bus utization, n_b is number of bytes transferred per memory access, n_c is a number of clock cycles per transfer with the memory (M), and t_c is clock cycle time; n_c is formed by a number of clock cycles for SB and a number of clock cycles to access off-chip memory.

Now, I_R can be calculated as:

$$I_R = \frac{B}{n_t} = \frac{B}{n_i + n_d} \qquad (2)$$

where: n_t is total average number of bytes per instruction executed, n_i is number of instruction bytes, and n_d is number of data bytes.

We will define n_c , n_i , and n_d in case of a system with on-chip instruction cache and data cache memories in conjunction with a fast off-chip instruction/data cache.

$$n_c^* = n_{c1i} + (n_{c2} + n_{ca} \times \text{miss ratio off-chip cache}) \times \text{miss ratio on-chip icache} \qquad (3)$$

$$n_i^* = \text{miss ratio} \times \frac{n_i \times \text{no. of instruction bytes per miss}}{n_b} \qquad (4)$$

$$n_d^* = \text{miss ratio} \times \frac{n_d \times \text{number of data bytes per miss}}{n_b} \qquad (5)$$

We will consider two processors, a PowerPC processor and the VLIW processor used in the paper, with the same main features. The different consists of that

PowerPC executes 4 instructions and VLIW executes 13 instructions per clock cycle.

The common features are:

Data/instructions off-chip cache = 1 MB

- miss ratio = 0.1%; data line width: 64 bytes

Instruction on-chip cache = 32 KB

- miss ratio = 1.7%; data line width = 32 bytes

Data on-chip cache = 32 KB

- miss ratio = 2.5%; data line width = 32 bytes

Internal data paths = 64 bytes

Internal instruction path = 256 bytes for PowerPC and 432 bytes for VLIW processor.

External data path = 128 bytes

Clock frequency = 100 MHz

Bus utilization u_b = 75%

The number of bytes n_b = 128 / 8 = 16 bytes

n_{c1i} = 3 cycles; n_{c2} = 3 cycles; n_{ca} = 2 cycles

The specific parameters are:

- For PowerPC:

  n_i = 256 / (8 *4 instructions) = 8 bytes

  n_d = 64 / 8 = 8 bytes

- For VLIW:

  n_i = 432 / (8 *13 instructions) = 4.154 bytes

  n_d = 64 / 8 * 12 operations = 0.667 bytes

By applying formulas (1) up to (5) they obtaines:

$I_{RpowerPC}$ = ~585 MIPS

I_{RVLIW} = ~2247.4 MIPS

Because I_{RVLIW} / $I_{RpowerPC}$ = ~3.84, it obtaines a theoretic decrease of execution rate of 3.84 times better for VLIW processor versus PowerPC processor.

Conclusions

VLIW processors will certainly revolutionize the present processors because they can execute a great number of operations in parallel with a very good performance/cost rapport. Also, they can use technique designs from multiprocessors and vector machines.

All of these and clever compilers make them very suitable for a large classe of applications.

Program execution on VLIW machines is very simple and easy.. However, this requires the elaboration and utilization of program parallelization techniques. Thus, the exctraction of ILP from the scalar code is essential for VLIW processors because this code is present in all module of a sequencial program. A distinct attention must pay for the loops with jumps and/or branches because their parallelization is vital for parallel execution of more operations per one clok cycle.

The performance evaluation of any new system versus the existent systems, it is very useful to determine the viability of new systems. The superscalar techniques are used by the all actual processors, including VLIW, because of the rapidly increasing of performance by using of multiple functional units and of an efficient scheduling of the resources.

The increasing of ILP is another important factor for performance improvement; this increase is greater for VLIW processors versus any other actual processor implementation.

Because of the locality of the information and because of the faster rate of the caches, using cache memories, together with inconsistency cache memory avoiding schemas, become a necessity to increase system performance of actual superscalar processors, generally, of VLIW processors, specially.

The performance obtained on the VLIW proposed model shows that a VLIW processor can process 4-5 times more information than an actual RISC superscalar processor.

References

[1] M.Breternitz Jr., *"VLIW Compilation and Architecture Synthesis"*, Carnegie Mellon, 1991.

[2] K.Ebcioglu, *"A Compilation Technique for Software Pipelining of Loops with Conditional Jumps"*, in Proc. Of 20th Annual Workshop on Microprogramming, pages 69-79, ACM Press, 1987.

[3] M. Franklin, and M. Smotherman, *"A Fill-Unit Approach to Multiple Instruction Issue"*, Proceedings 27th Ann. International Symposium on Microarchitecture, San Jose, California, Dec., 1994, pp.162-171.

[4] J.A.Fisher, *"Trace Scheduling: A Technique for Global Microcode Compaction"*, IEEE Transaction on Computers, vol. C-30, pages 478-490, July 1981.

[5] M.S.Lam, *"Software Pipelining: An Effective Scheduling Technique for VLIW Machines"*, in Proc. of the SIGPLAN'88 Conference on PLDI, pages 318-328, Atlanta, GA, June 1988.

[6] C.Popescu, *"Performance Evaluation for the Superscalar Processor Systems"*, Development and Application Systems International Conferences, Suceava, RO, 1996, pages 143-152.

[7] C.Popescu, *"Compilation Techniques for VLIW Architecture"*, in International Conference on Development and Application Systems, No. 10, Suceava, Romania, 21-23 May 1998, pages 83-88.

[8] C.Popescu, F.Iacob, C.Morarescu, and A.Boicea, *"VLIW Processor – Architecture and Operating"*, Automation & Quality Control International Conference, Vol. Q2, Cluj, RO, 1998, pages 191-196.

[9] *** *"Verilog Language Reference Manual"*, Open Verilog International, Inc., 1993.