www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issues 7 July 2016, Page No. 17256-17259

Explicit Parallel Instruction Computing

Pranjal Mathur

Shiv Nadar University, School of Engineering and Computer Science
Greater Noida, India
Pm784@snu.edu.in

Abstract: ***VLIW (Very Long Instruction Word) machines are highly parallel ILP (Instruction Level Parallelism) based architectures that offer an alternative to scalar sequential architectures like CISC/RISC. With VLIW, the burdensome task of dependency checking by hardware logic at run time is removed and delegated to the compiler. HP and Intel introduced a new style of instruction set architecture called EPIC (Explicitly Parallel Instruction Computing), and a specific architecture called the IPF (Itanium Processor Family). EPIC is like VLIW, with extra cache prefetching instructions. EPIC combines the capabilities of both Superscalars processors and VLIW processors. Through this paper, we will try to understand the characteristics of EPIC processors which make them distinguishable.***

Keywords: VLIW (Very Long Instruction Words), RISC, CISC, ILP (Instruction Level Parallelism), Superscalars, OpCode,etc

1. Introduction

Over the past two and a half decades, the computer industry has grown accustomed to, and has come to take for granted, the spectacular rate of increase of microprocessor performance, all of this without requiring a fundamental rewriting of the program in a parallel form, without using a different algorithm or language, and often without even recompiling the program. For the time being at least, instruction-level parallel processing has established itself as the only viable approach for achieving higher performance without major changes to software. Instruction level parallelism (ILP) is the initiation and execution within a single processor of multiple machine instructions in parallel. Instruction-level parallelism (ILP) results from a set of processor and compiler techniques that speed up execution by causing individual RISC-style operations to execute in parallel. The two most important types of ILP processors, to date, differ in this respect. Superscalar processors are ILP processor implementations for sequential architectures—architectures for which the program is not expected to convey and, in fact, cannot convey any explicit information regarding parallelism.

Since the program contains no explicit information about available ILP, if this ILP is to be exploited, it must be discovered by the hardware, whereas VLIW machines are highly parallel architecture that offer an alternative to multiprocessor but have a tightly coupled, single flow control mechanism much like clean very parallel horizontal microcode. Programs for VLIWs must specify fine grained hardware control. VLIW CPUs are usually constructed of multiple RISC-like functional units that operate independently. It is the responsibility of the compiler to determine which operations can be grouped together and where they must be placed in this sequence of long instruction words. EPIC, a new generation of ILP can be considered as an evolved form of VLIW. EPIC (Explicitly Parallel Instruction Computing) is a kind of RISC. It's another level of the Instruction Level Parallelism that we will describe in this paper. Before that lets look a compiled form of tasks that are required for the processing of an instruction. They are:

- To check for the dependent instructions in ordered to find which of the instructions can be grouped together to be good for ILP
- Instruction assignment to the various hardware.
- To determine when the instruction starts executing [2].

Given below the table which describes the role of hardware or compiler in the various forms of ILP namely: Superscalars, Very Long Instruction Words and The EPIC.:

	Grouping	Assignment	Start of Execution
Superscalar	Hardware	Hardware	Hardware
VLIW	Compiler	Compiler	Compiler
EPIC	Compiler	Hardware	Hardware

At one hand, the superscalar processors are kind of conventional, non-parallel/sequenced instruction set in which the semantics of a program is based on a sequential machine model. It requires hardware support for all the stages of execution of the program namely: fetching, decoding, executing, etc. however, the necessary condition is maintains i.e. the semantics of the instruction and the program is maintained. At the other hand, VLIW has all its stages been handled by the compiler itself. Implementing a VLIW computer requires the layout and implementation of long instruction words which are required to be given with separate operation for each function unit on each cycle. The subset of the instructions which are not related, hence independents are grouped together in a single long instruction word and the distinct functions are assigned to the operational units by placing them in certain slots of a VLIW. Between VLIW and Superscalars is the EPIC architectural style, where the compiler is responsible for determining the group of unrelated set of the instructions and this information is communicated through the explicit information in the instruction set.

This is important to note that EPIC is similar to Superscalars in terms of their ability to be compatible across different implementations. However, it doesn't require specific hardware for the detection of the interdependency of instructions. Hence

EPIC provides the better halves of both the other two implementations for Instruction Level Parallelism.

2. Why EPIC?

Before getting into details of EPIC implementation, its features and others, let's see why the EPIC architecture was developed and what the motivation behind that was. HP and Intel have recently introduced this style of instruction set architecture or more specifically instruction level parallelism called EPIC (Explicitly Parallel Instruction Computing). The Very Long instruction word kind of architecture is able to get the required high level of instruction level parallelism. Along with this, it provides lesser complexity in terms of hardware requirements. VLIW being specific for operations like numerical computing, etc. along with the shortcomings with respect to branch-intensive apps brings a need to improve the architectural designs, hence the EPIC. Also, the existing VLIW architecture is unable to provide acceptable object code across the various processors. Hence, EPIC is focused to provide more general architecture for various upcoming/ existing family of processors.

However, there are certain challenges for the development and implementation of EPIC architecture and they are as follows:

1.1 Development of the architecture which is applicable to general purpose computing:
- Find substantial parallelism in "difficult to parallelize" scalar programs
- Provide compatibility across hardware generations
- Support emerging applications

1.2 Compiler must find or create sufficient Instruction Level Parallelism

1.3 Combine the best of both the attributes of VLIW & superscalar RISC.

1.4 Scale architectures for modern single-chip implementation.

The above given implementation requirements drive the main idea behind the EPIC and they are:

1.1 Compile does the scheduling, permitting the compiler to play the statistics (profiling).

1.2 Hardware supports speculation:
- Address the branch problem: predicted execution and many other techniques.
- Addressing the memory problem: cache specifiers, prefetching, and speculation on memory alias.

3. Epic instruction set architecture

Beside the architectural specifications, the design and the format of the instruction must take care of the following: description of the operation code (opcode) repertoire, data structure, types, registers and their usage [1].

However, all the ISAs must bind to the following set of points given below:
- Compiler provides the basic execution plan of the instruction while the architecture must be able to support it.
- The architecture should be able to assist the compiler in exploiting statistical ILP
- The communication of the POE is communicated to the hardware, hence it must be supported by the architecture of EPIC IS.

4. EPIC Characteristics

Following are the characteristics that defines the EPIC [2]:

4.1 Explicit Parallelism

One of the characteristics of the EPIC architecture is that, they provide explicit information on independent instructions in the program. In Intel Itanium Processor Family, implementing EPIC, a 5-bit number identifies the type of the instruction and also defines the architectural stops between the groups of instructions. Five other attributes of EPIC architectures beyond instruction grouping. The first two deal with eliminating and/or speeding up branching, the third with cache locality management, and the final two with starting load instructions as early as possible.

4.2 Predicated execution:

The instructions can be conditioned or predicated on a true/false value in a predicate register to identify if the instruction is conditional branch instruction. Only those instructions with a true predicate are allowed to write into their destination registers. Intel Itanium Processor Family provides 64 predicate registers where one register can hold one bit (true or false) and is set by compare instructions.

4.3 Compiler controlled memory hierarchy:

The latency due to instruction like LOAD can be predetermined and is told to the hardware before execution along with the probable location of load and stored data items. Intel Itanium Processor Family provides hints for LOAD, STORE and FETCH type of instructions. Along with this, it provides prefetching stride information by use of base update addressing mode.

4.4 Data speculation:

Aliasing (aliasing describes a situation in which a data location in memory can be accessed through different symbolic names in the program) limits the compilers to know the memory addresses to which the instructions refer beforehand. In the absence of exact alias analysis, most compilers must settle for safe but slower code. EPIC architectures provide speculative loads that can be used when an alias situation is unlikely but yet still possible. Intel Itanium Processor Family provides the facility to check the advanced load and advanced load instructions using Advanced Load Address Table.

Some other characteristics provided by the EPIC (IPF (Intel Itanium Processor Family)) Architecture are:
- The IPF architecture uses the predicate registers to record the results of comparisons and includes eight branch registers for use in prefetching to control the unbundled branches.
- IPF provides speculative load and speculation check.

5. The EPIC philosophy

One of the main goal of EPIC is to retain structural ideas of VLIW of statically forming the plan of execution, and at the same time to augment it with features akin to those in a superscalar ILP processor that would help it to better cope with dynamic factors, which traditionally limited VLIW-style parallelism.

The code for a superscalar processor consists of an instruction sequence that yields a correct result if executed in the stated

order. The code specifies a sequential algorithm and, except for the fact that it uses a particular instruction repertoire, has no explicit understanding of the nature of the hardware upon which it will execute or the precise temporal order in which instructions will execute.

In contrast, VLIW code provides an explicit plan for how the processor will execute the program, a plan the compiler creates statically at compile time. The code explicitly specifies when each operation will be executed, which functional units will do the work, and which registers will hold the operands. The VLIW compiler designs this plan of execution (POE) with full knowledge of the VLIW processor, so as to achieve a desired record of execution (ROE)—the sequence of events that actually transpire during execution.

To accomplish these goals, EPIC has the following characteristics [2].

5.1 Designing the plan of execution at compile time:

EPIC puts the burden of designing the plan of execution on the compiler. Otherwise, a processor's architecture and implementation can obstruct the compiler in performing this task. But EPIC processors have features that instead assist the compiler.

The essence of engineering a plan of execution at compile time is to reorder the original sequential code to best take advantage of the application's parallelism and make best use of the hardware resources to minimize the execution time. Without suitable architectural support, this reordering can violate program correctness.

Thus, because EPIC places the burden of designing the plan of execution on the compiler, it must also provide architectural features that support extensive code reordering at compile time.

5.2 Permitting the compiler to play the statistics:

Certain things that necessarily affect the flow of execution can only be known at runtime. For eg, a compiler cannot know which way a conditional branch will go, and, when scheduling code across basic blocks in a control flow graph, the compiler cannot know for sure which control-flow path is taken. In addition, it is practically impossible to make a schedule statically that jointly optimizes all the parts of the program.

An important part of the EPIC philosophy is to allow the compiler to play the odds under such circumstances the compiler constructs and optimizes a plan of execution for the likely case. However, EPIC provides architectural support such as control and data speculation to ensure program correctness even when the guess is incorrect.

When the gamble does not pay off, program execution can incur a performance penalty. The penalty is sometimes visible within the program schedule.

Or, the penalty may be incurred in stall cycles that are not visible in the program schedule.

5.3 Communicating the POE to the hardware

To do so, the instruction set architecture must be rich enough to express the compiler's decisions as to when to issue each operation and which resources to use. In particular, it should be possible to specify which operations are to issue simultaneously.

When communicating the plan of execution to the hardware, it is important to provide critical information at the appropriate time. A case in point is a branch operation, which—if it is going to be taken—requires that instructions start being fetched from the branch target well in advance of the branch being issued. Rather than providing hardware to deduce when to do so and what the target address is, the EPIC philosophy provides this information to the hardware, explicitly and at the correct time, via the code.

Cache hierarchy management and the associated decisions as to what data to promote up the hierarchy and what data to replace. Such policies are typically built into the cache hardware. EPIC extends its philosophy of having the compiler orchestrate the record of execution to having it also manage these micro architectural mechanisms.

6. Architectural features supporting EPIC

EPIC provides architectural features that permit programmatic control of mechanisms that the microarchitecture normally controls.

EPIC uses a compiler to craft statically scheduled code that allows a processor to exploit more parallelism, in the form of wide issue-width and deep pipeline-latency, with less complex hardware. EPIC permits the elimination of complex logic for issuing operations out of order by relying upon the issue order specified by the compiler. EPIC philosophy permits the elimination of runtime dependence checks among operations that the compiler has already demonstrated as independent [3].

6.1 Static Scheduling

A multi operation instruction is one that specifies many operations simultaneously. Each operation is comparable to a conventional RISC/CISC instruction.

The compiler identifies operations that can be scheduled simultaneously and packages them in a multi operation instruction. When issuing a MultiOp, the hardware does not need to perform dependence checking between its constituent operations. Furthermore, a notion of virtual time is associated with EPIC code; by definition, exactly one MultiOp instruction is issued per cycle of virtual time. Virtual time differs from actual time when runtime stalls that the compiler did not anticipate are inserted by the hardware at runtime.

Traditional sequential architectures define execution as a sequence of atomic operations; conceptually, each operation completes before a subsequent operation begins. Such architectures do not entertain the possibility of one operation's register reads and writes being interleaved in time with those of other operations.

With MultiOp, operations no longer are atomic. When executing operations within a single MultiOp, multiple operations may read their inputs before any operation writes its output. Thus, the non-atomicity and the latencies of operations are both architecturally exposed.

6.2 Addressing the branch problem

Most applications of normal use are very branch intensive. Latency of branches are measured in cycles. Branch operations have a hardware latency that extends from when the branch begins execution to when the instruction at the branch target begins execution. During this latency, several actions occur:

- The hardware computes a branch condition
- forms a target address
- fetches instructions from either the fall-through or taken path
- And then decodes and issues the next instruction.

Although conventional ISAs specify a branch as a single operation, its actions are actually performed at different times, which span the branch's latency.

EPIC allows static schedules to achieve better overlap between branch processing and other computation by providing architectural features that facilitate three capabilities:

- Separate branch component operations, which specify when each of the branch actions is to take place;
- Support for eliminating branches; and
- Improved support for static motion of operations across multiple branches.

6.2.1 **Unbundled branches**: EPIC branches unbundle into three components:

- A prepare-to-branch, which computes a branch's target address;
- A compare, which computes the branch condition
- And an actual branch, which specifies when control is transferred.

6.2.2 **Control speculation**: Branches present barriers to the static reordering of operations needed for efficient schedules. In addition to predicated execution, EPIC provides another feature that increases operation mobility across branches: control speculation.

6.3 Addressing the memory problem

EPIC provides architectural mechanisms that allow compilers to explicitly control the motion of data through the cache hierarchy. These mechanisms can selectively override the default hardware policies.

6.3.1 Cache specifiers: Unlike other operations, a load can take on a number of different latencies, depending on the cache level at which the referenced datum is found. EPIC provides load operations with a source cache specifier that the compiler uses to inform the hardware of where within the cache hierarchy it can expect to find the referenced datum and, implicitly, the assumed latency. In order to generate a high quality schedule, the compiler must do a good job of predicting what the latency of each load will be (and then communicate this to the hardware using the source cache specifier). This it can do by using various analytical or cache miss profiling techniques.

6.3.2 Data speculation: Another impediment to creating a good plan of execution results from low-probability dependencies among memory references. Often, a compiler cannot statically prove that memory references are to distinct locations and must conservatively assume that the alias, even if in reality this is generally not so. Data speculation allows the compiler to generate program schedules that assume that a store and a subsequent load do not alias even though there is some small chance that they do.

Conclusion

The new era of faster computing is supported by various instruction set architectures. EPIC is one of them, meant for instruction level parallelism. It combines traditional and new set of ideas to bring out the features required for faster, uninterrupted execution of program. The EPIC philosophy—in conjunction with the architectural features that support it provides the means to define ILP architectures and processors that can achieve higher levels of ILP at a reduced level of complexity across diverse application domains. EPIC architectures can claim to combine the best attributes of superscalar processors (compatibility across implementations) and VLIW processors (efficiency since less control logic). EPIC exposes various mechanisms at the architectural level so that the compiler can control these dynamic mechanisms, using them selectively where appropriate, hence optimum for the faster generation of results.

References

[1] http://www.cse.unsw.edu.au/~cs9244/06/seminars/02-nfd.pdf

[2] Understanding EPIC Architectures and Implementations by: Mark Smotherman, Dept. of Computer Science, Clemson University

[3] http://www.hpl.hp.com/techreports/1999/HPL-1999-111.pdf

[4] http://cse.yeditepe.edu.tr/~gkucuk/courses/cse533/vliw-epic.pdf

[5] EPIC: An Architecture for Instruction-Level Parallel Processors by Michael S. Schlansker, B. Ramakrishna Rau, Compiler and Architecture Research, HP Laboratories Palo Alto

Acknowledgements

Special thanks to **Dr. Rajeev Kumar Singh**, Assistant Professor, Department of Computer Science and Engineering, Shiv Nadar University.

Author Profile

Pranjal Mathur is a current student of esteemed Shiv Nadar University, pursuing his undergraduate studies in Computer Science and Engineering. He is an active member of IEEE Student Chapter, Shiv Nadar University. Awarded by HRD Ministry of India for exceptional academic achievements, he holds certification of Merit for AISSE 2014 Examinations. He has worked for various organizations including Indian Railways, Geeksforgeeks, and various startups and has an active record of research publications in the field of Computer Science and Education.