Exploratory Study of Techniques for Exploiting Instruction-Level Parallelism

Misra Sanjay
Covenant University,
Ota, Nigeria.
ssopam@gmail.com

Abraham Ayegba Alfa, Mikail Olayemi Olaniyi
Federal University of Technology,
Minna, Nigeria.
abrahamsalfa@gmail.com, mikail.olaniyi@futminna,edu.ng

Sunday Olamide Adewale
Federal University of Technology,
Akure, Nigeria.
adewale@futa.edu.ng

***Abstract*— The performance of memory system depends majorly on types of instruction constructs, speedup of executions, capacity of processing elements and scheduling techniques. Most scheduling techniques are faced with several challenges such as multiple issues, exploiting more parallelism in programs instructions, speedup rate of executions and support for conditional instructions constructs. Recent innovations in memory system and scheduling techniques required support for instruction-level parallelism (ILP) algorithm, which is overlapping of instructions sets for parallel processing and execution. To achieve these, a survey of the widely used techniques for exploiting of instruction-level parallelism (ILP) is carried out to identify their strengths and their weaknesses by reviewing several related works. This paper finds out the limitations of the various techniques for exploiting ILP and used these reviews to propose a new technique to overcome these limitations.**

Index Terms—Parallelism, algorithm, instructions execution, instructions constructs, basic blocks (BB), instruction-level parallelism (ILP), loops architectures, two-way loop technique

I. INTRODUCTION

In computer architecture, it is possible to influence the logical execution of instructions from the programs of a user because of some attributes of a system are readily obvious to the user. This potentially makes it possible to improve the performance of the system by altering these attributes [1], [22]. In the past decades CPU frequencies steadily increased at a continuous scale until recently when this trend stopped due to physical limitations in the technology of the ICs (Integrated Circuits; that is space between components). The opportunities to speed up program execution through instruction-level parallelism exploitation became feasible because segments of the program can be executed at the same time as compared with sequential execution speeds [2], [22], [23].

Parallel execution at the instruction level can be achieved through techniques such as the superscalar execution, VLIW and Single Instruction Multiple Data (SIMD), while parallelism intrinsic in high level algorithms can be better put to practical use through multi-core architectures [3].

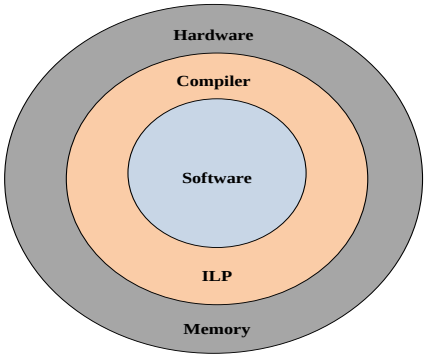

Fig. 1. A model of computer system depicting ILP and its relationship to software and hardware components [23]

ILP is ability to overlap executions of instructions sets, which can be attained dynamically (hardware related) or statically (software related such as compiler and system software) [7], [22], [23]. A model of computer system depicting ILP and its relationship to other components such as software and hardware components are illustrated in Fig. 1. [23]

The remainder of the paper is organized as follows. The next section discusses various techniques of instruction-level parallelism to determine their unique features and strengths. In section III, challenges of instruction-level parallelism techniques are identified. Following that, the outcome of the exploratory study are conversed in section IV and, finally, conclusions are drawn in section V.

II. TECHNIQUES OF INSTRUCTION-LEVEL PARALLELISM

Below are some of the widely deployed techniques for exploiting instruction-level parallelism in compilers/memory system.

A. Loop Unrolling Technique

Present day high-performance processors comprise hardware resources that support overlapped execution, independent instructions execution, parallel instruction pipelines, multiple functional components and multiple data

paths. This characteristic of overlapping instruction execution is what is called instruction-level parallelism (ILP). The use of hardware resources to take advantage of available ILP may have become commonplace but, even more interesting is the use of software resources to exploit ILP present in programs [4].

In general, loop unrolling technique is a paradigm code optimization mechanism that is combined with register renaming to boost ILP. These optimization codes replicate the original loop body iteratively, adjust the loop exit code and remove inessential branch instructions (or transforms conditional instructions to straight instructions) [4]. The use of larger block output for instructions increases the chances that the instruction scheduler can rearrange instructions to utilize ILP. That notwithstanding, the efficacy of scheduler is bound by simulated dependencies between instructions. While, register renaming eliminates the simulated dependencies, the successive loop has more ILP presence than the primitive (rolled loop). Generally, compiler is responsible for discovering dependencies in loop bodies using process analysis [2], [4].

In applying loop unrolling, it is possible to adapt hardware caching approach to loop bodies by categorizing its hardware components such as a comparator (use to detect conditional instructions), a negative branch displacement and a stack-based approach (called the loop stack) [5]. During commit time, when loop entry is discovered, the loop linked information is written on to the loop stack. This dynamic information takes account of the per-iteration in-loop branch log and the number of successful iterations per visit [4], [6]. The outcome of each conditional instruction executed inside loop iteration is registered in a prediction table for the loop.

Upon the expiration of a loop visit, the loop predication table keeps the complete branch log for all successful iterations during the prior loop visit. In reality, majority of loop tend to go over a range of paths in an event of a visit and these patterns carry on to successive visits to the loop [4]. However, updating the records for these set of paths dynamically makes it possible to predictably initiate the entire loop visits by apportioning dynamic loop traces in the cache. The dynamic loop behavior can be distinctly recognized by monitoring information for each loop pair (loop head, loop tail); these help to monitor the behavior of hot loops in a small or fast loop cache and lookup table [4], [6].

In practice, loops are accountable for the greater part of the execution time in several categories of the applications. Almost all scientific applications demonstrate approximately close to 90% of the time of execution in one or more loops [2]. Compile-time loop unrolling is widely utilized in scientific programs to attain higher speedups. In general-purpose applications (SPECint programs) several loops are available, though many of these loops have inherent qualities that limit their capacity to unroll at time of compilation [2]. The features of different workloads determine the different degrees of ILP revealed. Another advantage is the ability to estimate the actual number of successful loop iterations over during multiple visits to the same loop. The interrelationship of the pattern of conditional branch results provide guide to the loop can become a reliable predicting tool for loop entry point. [4], [5].

Loop unrolling technique is a hardware-based approach, which captures information about past loop behavior and instructions present in the loop body. It gathers this information for an entire execution that can be suggestively utilized to load into instruction window. Again, it can provide useful information for constructing instruction window containing thousands of instructions, as a consequence higher levels of ILP is revealed [4], [5], and [23].

B. Superscalar Technique

Superscalar machines have ability to issue multiple autonomous instructions per clock cycle when these instructions are suitably scheduled by the compiler and runtime scheduler. Simply put, multiple issues combined with dynamic scheduling are referred to as superscalar [9].
Superscalar architecture for processor is designed for common instructions sets such as integer and floating-point arithmetic, loads, stores and conditional branches, which can be initiated simultaneously and executed individually [1], [2].

The superscalar machine is invented to improve the performance of the execution of scalar instructions in most applications (the bulk of these operations are on scalar quantities). Essentially, the superscalar approach provides the potentials to execute instructions independently and concurrently in different pipelines [1], [2].

The exploitation of this concept can be expanded to allow instructions to be executed in an order different from the program order. In general, the superscalar model of two integers, two floating-point and one memory (either load or store) operations can be executed at the same time as depicted in Fig. 2. There are multiple functional units that are implemented as a pipeline to support parallel execution of diverse instructions [1].

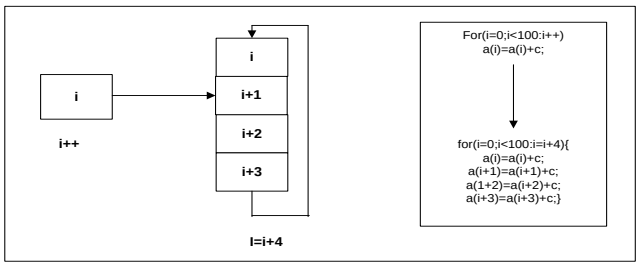

Fig. 2: Loop unrolling architecture and code. [8]

C. Super Pipelining Technique

Super pipelining technique attempts to achieve greater performance by taking advantage of the fact that many pipeline stages carry out tasks that require less than half a clock cycle. Consequent upon this, a doubled internal clock speed allows the performance of two tasks in one external clock cycle. The pipeline has four stages: instruction fetch, operation decode, operation execution and result write-back. Figure 3 shows the super pipelining architecture in that the

base pipeline issues one instruction per clock cycle and can perform one pipeline stage per clock cycle [1], [2].

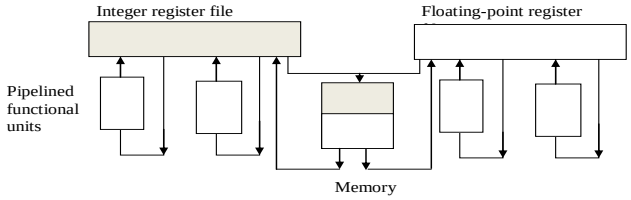

Fig. 2. Superscalar architecture. [1]

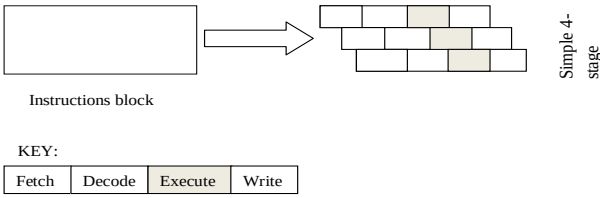

Fig. 3. Supers pipelining architecture. [1]

D. Single Basic Block Technique

The execution of a single basic block of instructions on machine is partitioned into a series of independent operations referred to as execution cycle of instruction. The instruction pointer (or program counter) maintains the address of the successive instruction. The instruction queue keeps details about instruction to be executed. There are five basic steps for executing a machine instruction if instruction makes use a memory operand; *fetch operand and store output operand* [10], [23].

E. Very Long Instruction Word Technique

Very Long Instruction Word (VLIW) is machine in which the compiler is responsible for initiating package of instructions for concurrent multiple issues without interference of the hardware [1]. The compiler is concern dynamically with reaching decision on multiple-issue or multiple-issue side by side static scheduling. Compiler takes the role for scheduling instructions, simplifying hardware and software complexity [9]. Decoupled architectures combines the best features of static scheduling of register-to-register instructions and dynamic scheduling of memory operations (buffers) [9].

F. Trace Scheduling Technique

Trace scheduling is the formerly deployed for VLIW architecture. Trace describes sequence of straight line instructions executed within specific data or collection of operations that make up the potential path based on branches predicated. Trace scheduling determines most likely order of instructions and allocates the instructions in this path. Tools are designed for loops (such as unrolling) while that of branches is static branch predication [9].

G. Pipeline Parallelism Technique

Flynn, Weiss and Smith introduced pipelining in the architecture of the CPUs to permits instructions execution in minimum time possible [11], [12]. This is done by partitioning each instruction set into several segments that need different hardware for possibility of taking a cycle. A CPI (cycle per instruction) equal to one can be attained given any scenario to be true [11].

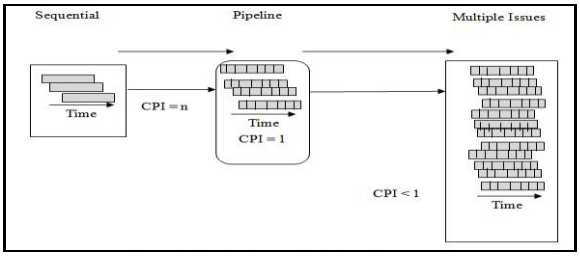

Fig. 4. Architecture of pipelining of instructions. [12]

Pipelined machines increases hardware parallelism, as a result of increases in the measure of parallelism that a complier generates keeps all the components of hardware units active. However, these various segments vary for different processors and instructions type. These instructions include: fetch and decode, generate address and fetch data for memory instructions, execute and write back. The architecture is realized for such *n*-segments can be determined in an instruction (*n* = pipeline depth).

H. Multiple Issue of Instructions Technique

Multiple issue of instructions technique takes advantage of parallelism inherent in programs of application. This process include: sequential stream of codes, compare and control data dependencies found in instruction, identifying sets of independent instructions to be issued concurrently without altering the correctness of program. VLIW (Very Long Instruction Word) separates and checks for dependent/independent instructions [2], [8]. The compiler completes the entire process of execution and produces a parallelized machine code format of instructions from previous of sequential code by grouping independent operations to generate one VLIW that does not rely on hardware for its final scheduling. Such instructions are not a single assembler-like operation but bundle of several of such operations issued simultaneously [1], [8].

This case is different for superscalar machines because, they reckon on hardware for scheduling instructions. In this architecture, a specifically large portion of the silicon area of processor is reserved for the analysis of potential dependencies within instructions at run-time to identify more candidates for concurrent issue. The locality of codes for calling block and the called block could pose a major challenge for simultaneous instructions issue logic in superscalar scheduling. Both techniques depend on hardware for control logic [2], [8].

Software pipeline rebuilds loops iterations comprising individual iteration from several other instructions loops iterations. The size of code diminishes for software pipelining

as compared to unrolling. It provides special software feature to rotate register banks, multiple issues and instructions predicate. Another benefit of software pipeline is to render high performance loop and exploit more ILP [9]. Modulo Scheduling is simple and widely deplored software pipelining technique that is very efficient approach suitable for single basic-block loops, but it does not address properly the software pipelining of the loops containing conditional constructs [14].

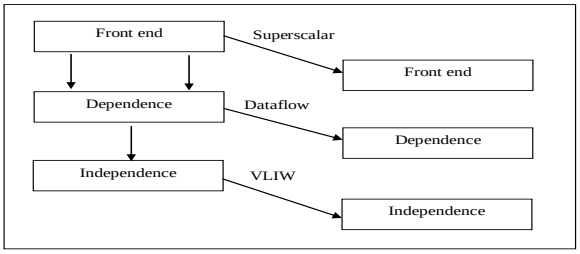

Fig. 5. Types of ILP architectures showing superscalar, dataflow and VLIW. [13]

High ILP can be exploited from loop because inherent quality of iterating on sequence of instructions available within its body giving rise to high special and locality present for these structures. Compilers recognize, replicate copies of loop bodies, perform software pipelining, provide improved instruction schedules and expose higher levels of ILP [4].

The branches elimination and control dependences are achieved by applying method of transformation algorithm [15], [16], [17].

III. CHALLENGES OF INSTRUCTION-LEVEL PARALLELISM TECHNIQUES

A. Branch Instruction Prediction

Present day microprocessors need extremely correct prediction of branch to attain better performance. New predictors types proposed over time produces very high accuracy requiring complex hardware are unable to deliver single cycle predictions. These techniques generally make use of complex computations and several table lookups which account for latency of several cycles per prediction as well as the bid to maintain clock frequencies [18].

Gshare predictor is a branch instruction prediction mechanism that takes advantage of a fixed length history of the several recent branch results for every successful prediction carried out. While, Neural predictors is another branch instruction predictor that allows a predictable branch to make use of some segment(s) of the global branch history practically connected to it assigning weights to select the most suitable segments. Again, spotlight branch predictor is a simple-design branch predictor that is capable of achieving very strong prediction accuracy using profiled information for a particular section of the global history register to arrive at decision spotlight. Branch instruction predictors can provide very low latency because of one or two table lookups merged with simple combinational logic mechanism [19].

The major penalty arises if significant quantity of time is expended when the system fetches along the wrong path (such as pipeline refilling time) and delays in computing dependencies in case of branch/conditional instructions. Although, the penalty for misprediction varies in degrees from one branch to another, a predictor makes effort to predict high-penalty branches that may improve performance though the total numbers of mispredictions remain unchanged. Most of branch predictors' mechanism focus only in minimizing the misprediction rate but ignores the major issue misprediction penalty [20], [21].

B. Multiple Branch Prediction

Delivering instructions for multiple issue systems remains a bottleneck as a result of the need to predict (or determine) the execution path that decode branches. Most basic blocks are usually smaller than width of issue for near future systems exploiting multiple branch prediction. The benefit of the rate instruction supply for wide-issue superscalar processors has long been discovered. Early studies found out that execution rates of well over 20 instructions per cycle (IPC) were possible though feasible rates would be much lower [21].

Recognizing that both of these execution rates and the decode width of near-future processors will exceed the basic block size of many programs. That is the motivation for the development of instruction supply mechanisms that could fetch more than one basic block per cycle by investigators [21], [22].

Another innovation in this area is the use of multiple branch predications such as a multiple branch predictor is that of Yeh, Marr and Patt often referred to as Y-MBP [21]. In PCs, index branch address cache (BAC) supplies details on the tree of basic blocks inside a specific number (say 3) basic block for the program counter (PC); a path through the block is chosen in one variation with a global history branch predictor. These schemes looked at capacity to supply instructions to a perfect execution engine, assessing an effective fetch rate (correct-path instructions fetched per cycle). But, the actual performance would be lower because of limited ILP available and load latencies [21].

C. Loop Predictor

1) *Penalty predictor:* It is used to determine a normal or high-penalty allotment to a mispredicted branch. The penalty predictor utilizes a PC-indexed penalty table that holds an 8-bit penalty counter and a state bit each entry. Penalty counters increments by 8 for a high-penalty branch and otherwise decrements by 1. A branch is considered as a high-penalty if the time taken for the branch to flow from the fetch stage of the pipeline (carries out a prediction) to the retire stage (for the branch to be resolved) surpasses a starting point and 120 cycles configuration for the competition [20], [23].

2) *Two-Class TAGE predictor:* It produces maximum prediction accuracy for branches with high-penalty prediction instead of other branches that utilize multiple tables to predict for the same branch simultaneously.

   a) *Loop predictor:* Is made up of a prediction process that is based on conditions being true such as WITHLOOP in program, loop pattern detected and

loop branch has been synchronized during fetch stage. It detects unutilized branch of this loop.

b) Update: At retire phase, the actual outcome of the present branch and its two predictions created by the loop which is utilized to renew the loop predictors [20], [23].

IV. OUTCOME OF THE EXPLORATORY STUDY

The study revealed the features that make unrolling loop, software pipelining and VLIW widely deployed techniques, and their specific limitations for exploiting ILP as summarized in Tables 1 and 2 respectively.

TABLE 1. FEATURES OF UNROLLING, SOFTWARE PIPELINING AND VLIW

Techniques	***Features***
Unrolling loop	-Provides x/y times overhead if x iteration and y unrolling -Large code size -Predictable execution -ILP exploitation -Replicates loop
Software Pipelining	-Provides two times, one at prologue and one at epilogue -Storage constraint -Optimal runtime -Reduce code -ILP exploitation
VLIW	-Memory port deficiency -Serious memory stall -Basic block may be too small as much as global code motion is difficult -Exploits ILP

TABLE 2. LIMITATIONS OF THE ILP TECHNIQUES

Techniques	***Limitations***
Loop unrolling	-No support for branch/dependencies in instructions constructs. -No overlap between sub-loops of original loop body. -The control of execution order is solely by compiler.
Superscalar	-No support for branch instruction constructs. -No overlap of executions
Super pipelining	-No support for branch instruction constructs. -Predefined order executions/phases.
Single Basic Block	-Basic blocks are often small to accommodate large size of instructions window. -No support for interrupts because, there is only one entry and exit point available in block. -No overlap of execution. -Predefined order of execution of instructions.
VLIW	-It is machine dependent. -No support for complexity but decouples architecture to their simplest forms.
Trace Scheduling	-No support for branch instruction constructs. -Time is wasted in predication cycle for branch instructions constructs. -No consideration for controller costs for most scheduling algorithms. -Controller style of scheduling determines the cost.
Pipeline parallelism	-It depends on hardware resources such as CPU. -No support for branch instruction constructs.
Multiple Issue of Instructions	-Supports for instructions construct. -No support for branch instruction constructs. -Resource constrained algorithm is required to achieve better interaction between scheduling and floor planning.
Realism	-Scheduling realistic design contains several special language constructs. -More Realistic libraries and cost functions. -Scheduling algorithms must be expanded to incorporate different target architectures. -No support for multiple instructions constructs.

A. Two-Way Loop Algorithm

Two-Way Loop algorithm supports multiple issues/concurrent instructions executions of straight and branch paths of loops. It modifies unrolling of loop technique by severally enlarging basic block for parallelism exploitation.

1) Identify conditional branch instructions //across several loop unrolling
2) Transform instructions in Step I into predicate defining instructions // instructions that set a specific value known as a predicate
3) Instructions belonging to straight and branch constructs are then modified into predicate instructions // both of them execute according to the value of the predicate
4) Fetch and execute predicated instructions irrespective of the value of their predicate// across several loops unrolling
5) Instructions retirement phase
6) If predicate value = TRUE // continue to the next and last pipeline stage
7) If predicate value = FALSE // nullified: results produced do not need to be written back and hence lost

B. Parameters for Evaluating Performance of Instruction-Level Parallelism Techniques

[23] proposed parameters for evaluating ILP techniques. They include: Performance index, the speedup of execution of instructions, number of multiple instructions paths, frequency of executions of instruction, number of loops processes available to complier, utilization – capacity of compiler to support more parallel processes.

C. Evaluation of Instruction-Level Parallelism

Time of Execution: The impact of ILP can be measured by the speedup in execution time (that is speedup of ILP) is defined by Equation 1

$$ILP\ Speedup = \frac{T0}{T1}\ . \quad (1)$$

where,
T0 = execution time of pipelining technique
T1 = execution time of TWL technique

Performance: According to Flynn Benchmark, Execution time equals to total time required to run program (that is wall-clock time for product development and testing) [22].

$$Performance = \frac{1}{(execution\ time)} \leq 1\ , \quad (2)$$

Utilization: Is number of instructions issued/number completed per second. The mean time that a request speeds in the system exposes more ILP. Cantrell [22] develops a benchmark and formula to compute number of instructions executed (μ) if the mean time of execution is T seconds, is given by: $\mu = \frac{1}{T}$

$$Utilization: \rho = \lambda T = \frac{\lambda}{u} = 1 . \quad (3)$$

The mean waiting time (i.e. no parallelism is present in program), Tw = ∞, $\lambda = rate\ of\ issue\ of\ instructions.$

V. CONCLUSION

The exploratory study reveals that the various techniques for scheduling and executing instructions largely depend on straight forward loops processing/executions as well as predication processes in the case of conditional (or branch) instruction constructs.

This paper recommends a Two-way loop algorithm approach that supports exploitation ILP and improves memory in computer system. Two-way loop technique has support for multiple processing of both straight and conditional instruction constructs, improve memory system performance by removing overhead incurred for prediction processes of the deployed techniques.

The execution speedup rate of computer system is better because of presence of many more parallel paths available to compilers. Two-way loop algorithm provides improved frequency of executions of computer system.

REFERENCES

[1]. S. William, Computer Organization and Architecture Designing for performance. 8th Edition, *Prentice Hall, Pearson Education Inc., Upper Saddle River, New Jersey, USA, 2006. pp. 3-881.*

[2]. J. Hennessy, and D. A. Patterson, Computer Architecture. Fourth Edition, *Morgan Kaufmann Publishers Elsevier, San Francisco, CA, USA, 2007, pp. 2-104.*

[3]. W. Pepijn, Simdization Transformation Strategies - Polyhedral Transformations and Cost Estimation. An *M.Sc Thesis, Department of Computer/Electrical Engineering, Delft University of Technology*, Delft, Netherlands, 2012, pp. 1-77.

[4]. D. Kaeli, and R. A. Rosano, Exposing Instruction Level Parallelism in Presence of Loops. *A Journal of Computational and Systems*, Mexico, CIC-IPN, Vol. 8, No. 1, 2004, pp. 74-85.

[5]. S. P. Vijay, and A. Sarita, "Code Transformations to Improve Memory Parallelism". In *proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, IEEE Computer Society, Haifa, Israel, 1999, pp. 147-155.

[6]. D. F. Bacon, S. L. Graham, and O. J. Sharp, Complier Transformations for High Performance Computing. *In Journal of ACM Computing Surveys*, New York, USA, 1994, pp. 345-420.

[7]. R. D. A. Marcos, and R. K. David, "Runtime Predictability of Loops". In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*, I.C., Ed., Austin, Texas, USA, 2001, pp. 91-98.

[8]. L. Pozzi, Compilation Techniques for Exploiting Instruction Level Parallelism, A *Survey. Department of Electrical and Information*, University of Milan, Milan, Italy Technical Report 20133, 2010, pp. 1-31.

[9]. E. Garcia, and G. Gao, Instruction Level Parallelism. In *Publications of CAPSL on Architecture and Parallel Systems Laboratory*, Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA, 2012, pp. 1-101. DOI: http://www.capsl.udel.edu

[10]. K. A. Parthasarathy, Performance Measures of Superscalar Processor. *International Journal of Engineering and Technology, IJET Publications*, UK, vol. 1, No. 3, 2011, pp. 164-168.

[11]. M. J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design. 1st Edition, *Jones and Bartlett Publishers Inc*., USA, ISBN: 0867202041, 1995, pp. 34-55.

[12]. J. E. Smith, and J. Weiss, PowerPC601 and Alpha 21064: A tale of two RISCs. In *Journal of Computer, IEEE*, vol. 27, Issue 6, 1994, pp. 46-58.

[13]. B. R. Rau, and J. A. Fisher, Instruction-Level Parallel Processing: History Overview and Perspective. *The Journal of Supercomputing, Massachusetts, Boston, USA, Vol. 7, Issue No 7, 1993, pp. 9-50.*

[14]. M. Lam, Software Pipelining: An Effective Scheduling Technique for VLIW Machines. In *Proceedings of ACM SIGPLAN Notices – Best of Conference on Programming Language Design and Implementation*, New York, NY, USA, vol. 39, No. 4, 2004, pp. 244-256.

[15]. S. A. Mahlke, R. E. Hank, J. E. McCormick, D. I. August, and W. W. Hwu, "A Comparison of Full and Partial Predicated Execution Support for ILP Processors". *IEEE Proceedings 22nd International Symposium on Computer Architecture*, ACM, New York, NY, USA, Vol. 23, No. 2, 1995, pp. 138-150.

[16]. M. S. Schlansker, and J. H. Park, On Predicated Execution. In *Software and System Laboratories*, Technical Report HPL-91-58, Hewlett-Packard Research Laboratories, Palo Alto, USA, 1991, pp. 1-7.

[17]. R. Johnson, and M. Schlansker, *Analysis of Predicated Code*. Technical Report HPL-96-119, Hewlett-Packard Research Laboratories, 1996, pp. 23-38.

[18]. S. Verma, B. Maderazo, and M. D. Koppelman, "Spot Light – A low Complexity Highly Accurate Profile-Based Branch Predictor". In *Proceedings of the 28th IEEE International Performance Computing and Communication Conference*, Phoenix, Arizona, USA, 2009, pp. 239-247.

[19]. S. Verma, and M. D. Koppelman, "Efficient Prefetching with hybrid Schemes and Use of Program Feedback to adjust Prefetcher Aggressiveness". *The IEEE International Parallel and Distributed Processing Symposium Ph.D. Forum in Conjunction with Journal of Instruction-Level Parallelism, Atlanta, GA, USA, 13(2011), pp. 1-4.*

[20]. A. Seznec, The L-TAGE Branch Predictor. *Journal of Instruction-Level Parallelism, Rennes, France, Vol. 9, 2007, pp. 1-13*, URL: www.jip.org/vol9/v9paper.pdf.

[21]. D. M. Koppelman, The Benefit of Multiple Branch Prediction on Dynamically Scheduled Systems. In *Workshop on duplicating, deconstructing and debunking Held in conjunction with the 29th International Symposium on Computer Architecture*, Anchorage, AK, USA, 2002, pp. 42-51.

[22]. M. Sanjay, A. A. Alfa, S. O. Adewale, A. M. Akogbe, and M. O. Olaniyi, "A Two-way Loop Algorithm for Exploiting Instruction-Level Parallelism". *In Proceedings of 14th Int'l Conference on Computational Science and Its Application (ICCSA 2014), published in Lectures Notes on Computer Science (LNCS), Journal of Science, Springer, University of Minho, Guimaraes, Portugal*, in press.

[23]. A. A. Alfa, Development of a Two-way Loop Algorithm for Improving Memory System Performance. A Master's Thesis, Department of Computer Engineering, Federal University of Technology, Minna, Nigeria, pp. 1-75, unpublished.