OPTIMIZATION OF MULTIPROCESSORS MEMORY SYSTEM PERFORMANCE INSTRUCTION LEVEL PARALLELISM

Takialddin Al Smadi [1] , Khalid A. Al Smadi [2] & Orayb A [3]
[1,3]Department of Communications and Electronics Engineering,
College of Engineering, Jerash University, -Jordan
[2] Jordanian Sudanese Colleges for Science & Technology, Khartoum, **SUDAN**

ABSTRACT

Wide issue processors such as the superscalar processor and Very Long Instruction Word processors exploit the Instruction level parallelism to achieve high performance. ILP is an overlap between instructions to execute more than one operation at the same time. This paper performs a comprehensive study aimed at exploiting and extracting ILP beyond the basic block by handling of branches and control dependences imposed by it. In this field the speculative execution which refers to execute instructions before needed, is employed using an efficient structure called super block, it will take place after scheduling one or more linear segment of instructions that can be arranged sequentially or randomly, The most important region of the developed scheduling algorithm is to increase the performance of processors by parallel execution with in earlier scheduling of branches.

Keywords: Instruction level parallelism, Optimization, Speculative execution, High-performance.

INTRODUCTION

Development of computer technology results in a large number of processor architectures includes the use of new technologies that require compilation. For example, the architecture with explicit parallelism commands requires the presence of the compiler optimizations, designed to identify and Use such parallelism, aggressive planning teams and pipelining loops. The popularity of embedded Architectures, widely used in mobile devices of the various applications, make it necessary to develop A compiler technologies that not only the performance of programs, but also a small executable size, and low power consumption.[1] Core architectures and heterogeneous architectures with multiple accelerators, as widespread need to develop new methods of compiling, allowing the programmer to specify the desired semi-automatically distributed computing and data flow on the components of such architectures, one of the most important aspects of the computing architectures, compilers, and processors is how to utilize as much as parallelism as Possible in order to improve the performance of multiprocessors by utilising Instruction Level Parallelism, that allows the execution of Arithmetic Logic Unit and memory instructions to be executed in parallel on other mean, the independent instructions execute concurrently and the conditional branch instructions with the dependences they impose require efficient handling. ILP-processors can vary in many characteristics that are important in terms of applicability and effectiveness of these methods of optimization [2]. Developed in processors need to achieve high performance benefits, applications now running faster and easy to run realistic graphics, simulator, and interactive games. This depends on tow basics: one is the hardware improvement features, such as speed and the circuit density, the second based on modifying the processors architecture and compiler techniques by rearranging instructions and, executing the independent instructions in parallel [3].Recent studies have shown that by using conventional code optimization and scheduling methods, superscalar cant increase the speedup of more than tow for nonnumeric programs that because each block have limited number of instruction but, nonnumeric benchmarks report that about 20% to 30% of

dynamic instructions are branches, which lead to apply the branch prediction technique to extract ILP across basic blocks, on other hand misperception will cause large performance penalties.
Super block optimization and scheduling techniques in Section 2, provide a new scheduling method that provides an effective framework for full speculation of instructions. Performance improvement and limitations of ILP exploiting are also addressed.

Scheduling Instructions

Consist of linear segments which in the original programs can be arranged sequentially or randomly, Areas differ in the structure of their control flow and by the method of formation. The most famous types of areas - superblocks, the tree of the field visible - The types of areas and their main characteristics.

Based planning methods, is that the code can be reorganized so that reduce the execution time along some routes by slowing along the other. If decisions are made in favour of accelerating the most common ways is through this can be done to reduce the time of the program as a whole. This approach may be acceptable in real-time applications where there may be restrictions on the execution time along any, even the most rare execution paths. The method does not require hardware support and is based on the addition of the minimum required number of additional linear segments containing probe code to register the transfer. The probe code is organized so as to ensure their implementation at maximum parallelism. Let us consider in more detail how to generate two types of areas - superblocks and tree areas [4] [5].

Superblock Methods

The definition of the concept of superblock extended linear plot. Advanced linear section is a sequence of linear segments B_1 ... B_k, that, for $1 \leq i < k$ Bi - the only precursor Bi +1. Based on data profiling, the merging point in the original program are removed by creating copies of the relevant sections. At the same time seek to distinguish superblocks along the route - the most frequently executed paths on the graph control. Example of the superblock is shown in Fig. 1

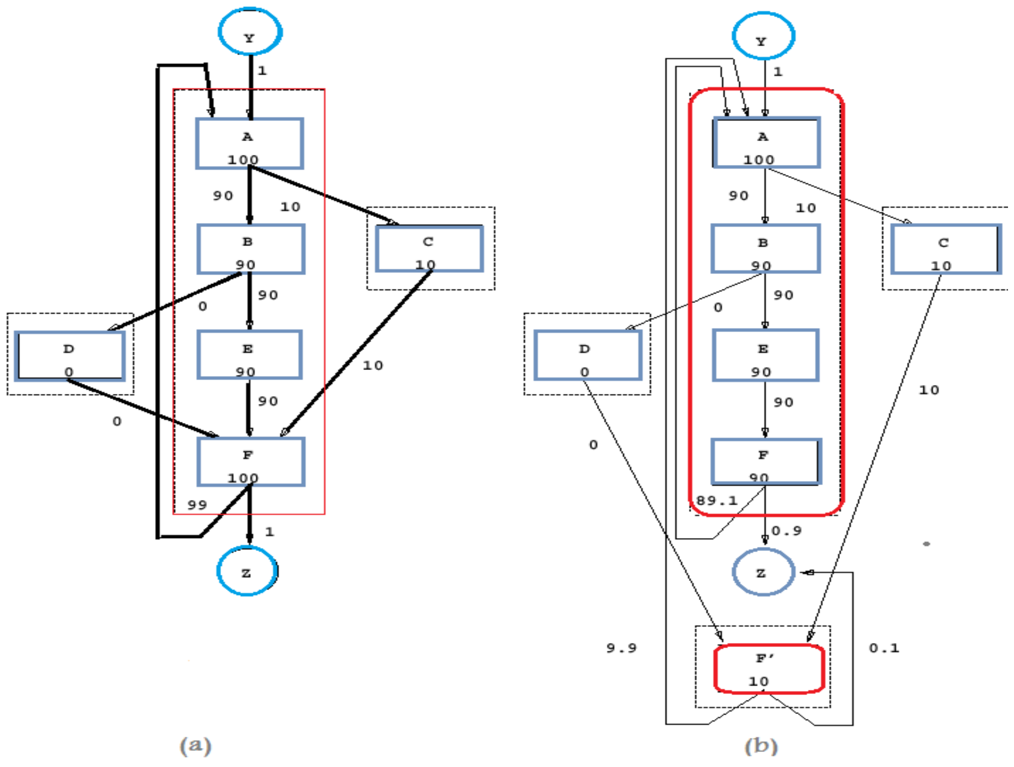


Figure.1 superblock formation procedure,
(a) After trace selection, (b) After tail duplication.

Constituting the body of the loop, indicating the frequency of execution sites and scheme shows that most execution follows: The path $A \longrightarrow B \longrightarrow E \longrightarrow F$, Therefore, the decision to form Three superblocks: $[A, B, E, F], [C], [D]$. You must eliminate the merging point of F. In Fig.1.b shows how this is achieved by adding a copy of F (F '). This technique is called "overlapping tails" tail duplication. In the end, a fragment of the original program, a 4 superblock: $[A, B, E, F], [C], [D]$. {F '}. Effectiveness of superblock optimizations for both superblock gyrations, the processor assumes to support General speculation, has no restrictions on the combination of instructions that may be issued each cycle and has the perfect caches.

Dominant Superblock ILP Optimization on Overall Performance

Superblock formation only usually gives only modest performance, potential for duplication by broadening the scope statement. The overlap is still limited. Data dependency and failure to use a loop of the ILP while superblock ILP Optimizations, such as deployment and induction loop variable expansion, aggressively transform Superblocks increase ILP in loops and straight-line code. As a result of scheduler have many more features to reorder instructions and achieve a compact schedule [6.7]

Tree Representation Method

Generalized state of the trees with the instructions included along any of its edges is a tree view, which is used in our empirical study as an intermediate representation parallel code. Limiting the maximum number of Test sites that can be compacted into a tree and limitations Places where the instructions can be scheduled in Tree, we can evaluate a wide Range of ILP architecture That use different levels of branch ILP. For example, for many superscalar processors that parallel execution instructions from one branch of the tree support the unconditional implementation with a maximum of one testing of the full tree Nods. VLIW Representation that supports conditional execution and multilateral. Branching will be described in section [8].

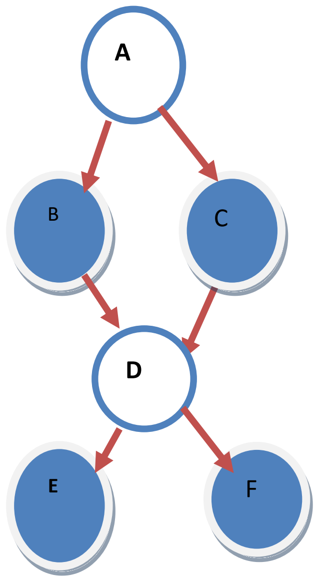


Fig.2a.Tree is consists of two prominent regions

Schedule instructions from multiple threads, Instructions from independent threads execute when function units are available, within threads, dependencies handled by scheduling and register renaming Fig2.b.

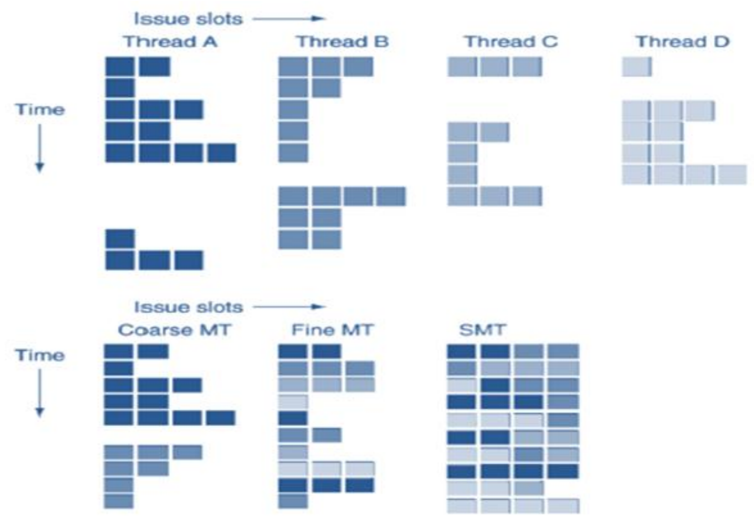


Fig.2.. duplicated registers, shared function units and caches.

Fig.2.a If the execution mainly follows the $A \longrightarrow B \longrightarrow E$, it is desirable to reorganize the code so that, the path $A \longrightarrow B \longrightarrow E$ was in the general area, and the scheduler to maximize the parallelism in this segment. In Fig. 2.b and fig.2.c shows the two stages of this transformation. First, it creates a copy of the D 'section D and emerging areas, including road $A \longrightarrow B \longrightarrow D$. It then creates a copy of the E' E, including ways $A \longrightarrow B \longrightarrow D \,\&\, A \longrightarrow C \longrightarrow D' \longrightarrow E'$ as well as the region, consisting of one section F.

Data profiling can also be used in the planning stage in the tree areas, in order to ensure the most rapid execution primarily along frequently executed paths. In order to limit the amount of the resulting program when making decisions about the "duplicate tails", in addition to profiling.
• Permitted overall expansion ratio should not exceed a predetermined value;
• The number of execution paths in each of the tree shall not exceed a specified amount;
If the number of predecessors in the graph controls, area larger than a given value, the overlapping area is performed. Similar heuristics used in the formation of other types of areas. In some cases, the scheduler is able to avoid the negative consequences of using the dominant method of dominator parallelism.If the commands in a linear plot B_{Bi} and copies B_{Bi} 'can be lifted to the dominant land B_{Bk} (which is the common ancestor of B_{Bi} and B_{Bi}'), it is possible to keep only one copy commands. An example of a situation where this [9-10]

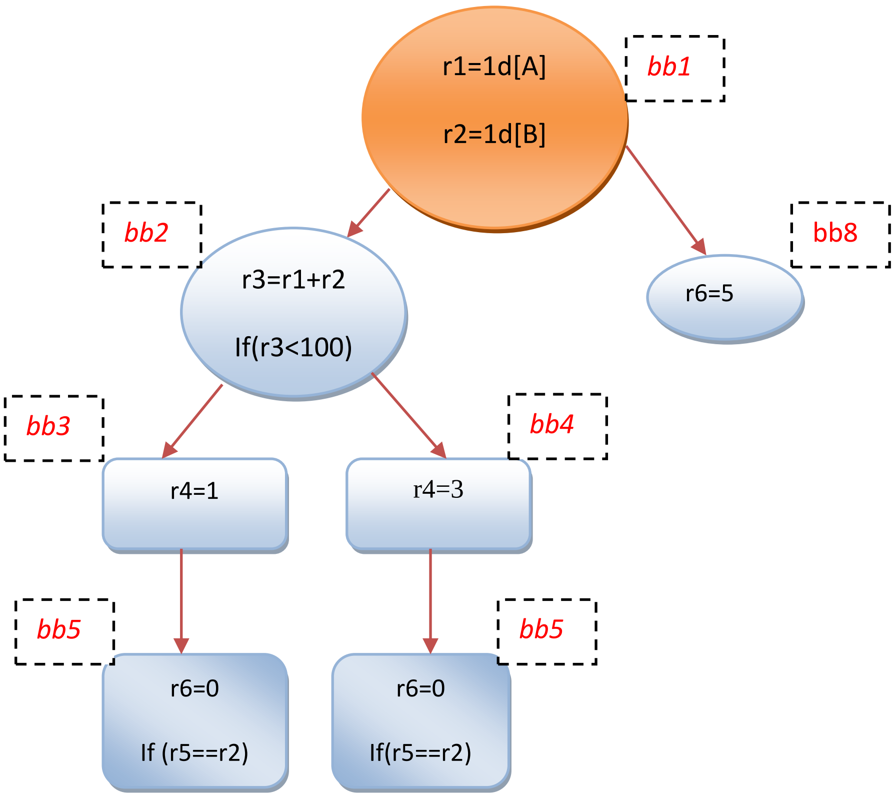


Fig. 3 Dominant parallelism
Teams r6 = 0 of the linear section of BB5 and copies of BB5 'can be raised in bb2
Or bb1 and eliminate duplication

Paralleling at the level of operations

This study analyses the program working with memory-intensive, they are a real challenge, which is dominated by calculations in loops. The Parallelization compiler architecture Elbrus provides special optimization and planning techniques that allow you to achieve maximum performance during the performance cycle on a given set of devices for a given bandwidth, processor memory [11].

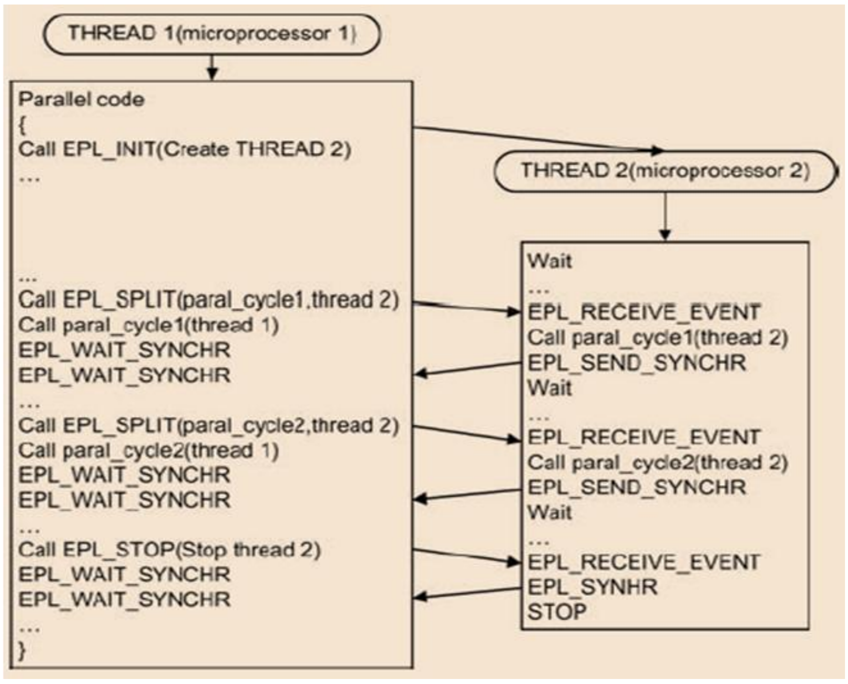
For the optimized software is applied the entire set of optimizations in the compiler's arsenal, which reduce the number of hits in memory. With the current iteration of the removed repeated treatment on the same addresses. Identifies and removes the same locations that appear on the next iteration. These optimizations can more than halve the number of hits in memory for cycles.

About cache level 2 processor has the Organization in the form of four independent banks. The data are placed in such a way that their simultaneous processing banks delivered the most evenly. Another optimization is to optimize the class that creates subsidiary hastening the process significantly arrays of list structures [12]. These optimizations can reduce significantly the number of locks because of conflicts when applying for the data in memory.

The technique is developed automatic parallelization compiler, which allows to parallelize cycles in sequential tasks that are implemented in C/c++ a multiple threads of control. The technology cycles suitable for parallelization compiler machined into separate procedures. In these procedures is a modifier that identifies the executable line and the rest of the procedure parameters transmitted through the stack. Use the technique of cut-out loops in separate procedures allows you to opt out of the use of the supplementary instructions to the compiler internally, which is indicated by a parallel region a similar approach is used in the Intel compiler [13].

Automatic Parallelization on Threads of Control

Developed automatic parallelization support library the main purpose of these interfaces is to establish deletion threads and synchronization of activities between them. For example, the EPL_INIT interface creates a second thread that waits for a pointer to the created code and permission from the main stream (EPL_SPLIT) on his performance (EPL_RECEIVE_EVENT interface). In turn, the first thread after execution of parallelized loop waits for the execution of the second part of the cycle in the second thread (EPL_WAIT_SYNCHR), all cycles for basic parallelized inductance [14], which defines the number of loop iterations. In Figure 5. Those show a scheme of a vector zed program


THREAD 1(microprocessor 1)
Parallel code
{
Call EPL_INIT(Create THREAD 2)
...
...
Call EPL_SPLIT(paral_cycle1,thread 2)
Call paral_cycle1(thread 1)
EPL_WAIT_SYNCHR
EPL_WAIT_SYNCHR
...
Call EPL_SPLIT(paral_cycle2,thread 2)
Call paral_cycle2(thread 1)
EPL_WAIT_SYNCHR
EPL_WAIT_SYNCHR
...
Call EPL_STOP(Stop thread 2)
EPL_WAIT_SYNCHR
EPL_WAIT_SYNCHR
...
}
THREAD 2(microprocessor 2)
Wait
...
EPL_RECEIVE_EVENT
Call paral_cycle1(thread 2)
EPL_SEND_SYNCHR
Wait
...
EPL_RECEIVE_EVENT
Call paral_cycle2(thread 2)
EPL_SEND_SYNCHR
Wait
...
EPL_RECEIVE_EVENT
EPL_SYNHR
STOP
Fig.4

Automatic parallelization can be conducted at any number of threads. In particular, for complex parallelization is performed on two threads. In Figure 1, an example of a parallelized loop nests on base of inductance covering cycle. On the left is a nest of cycles, consisting of three nested loops to each other, the variables i, j, k are base these inductances cycles. Basic inductance must be presented in the following form:

induct_var = oper (induct_var,const),

Where: induct_var is the basic variable inductance, const constant, and oper-addition or subtraction. As a result, the first half of the cycle, Sung in the first stream, and the other half on the second cycle. This technique allows the automatic parallelization of a separate parallelize a loop and a whole nest of cycles. Paralleling the outer loop is the most effective because it reduces the overhead of parallelization show fig.5 an example of parallelized loop nests [14].

The Results of Automatic Parallelization

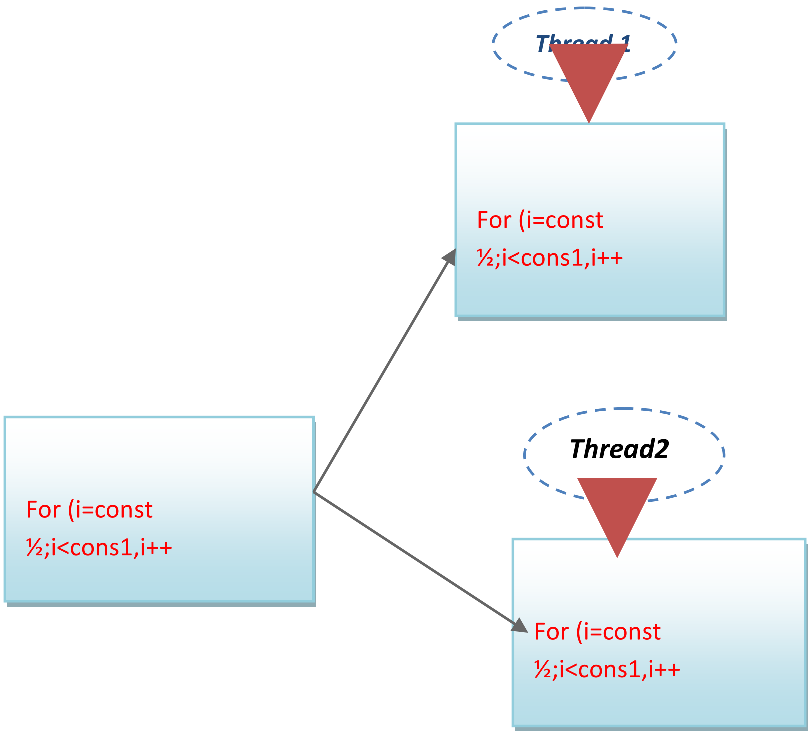


Fig.5 an example of a parallelized loop nests

That with full Threading tasks and there are conflicts when memory parallel version will be twice faster than the sequential version. But in reality the odds of accelerating parallel tasks proved to be far less two (Table 1, column REAL – for automatic parallelization and column MEM_IMP is to run two identical tasks). This is because the tasks to parallelize the query stream in memory increases, which lead to increased conflicts within the memory subsystem, and therefore slow down a vector zed version of the task. With conflicts lower system performance not only parallelizes a single task, but while performing the same tasks execution statistics package SPEC95 and SPEC2000

By paralleling the five discussed the tasks is an average of 1.42. The effectiveness of automatic parallelization can be estimated on the basis of the coefficients of the EXEC, as namely this indicator allows correlating the number of large teams, performed in sequential version and in a vector zed version. This means that the tasks were almost completely parallelization. In the case of tasks tomcat and hydro2d failed to achieve that result, because a significant portion of the time spent on task data execution of in/output functions that cannot be rasparalleleny by the compiler. In the case of art tasks this coefficient was obtained due to the fact that not everyone was able to parallelize loops.

Thus, the EXEC box determines the maximum possible acceleration of a vector zed tasks compared to sequential option. Speed up data failed to achieve in practice because of the influence of different types of locks when dealing with memory, as evidenced by the values of fields L1 MISS, and APB MISS. The true nature of these locks, most likely due to the influence of the memory subsystem, which is the subject of future study.

Handling arrays in loops. Even while performing one task improving memory configurations leads to significant performance gains (on average 19% from 21% to the chipset and the supplemental increase memory banks). Improvement from increasing the number of memory banks not as significantly as from improved chipset, since losing to switch modes of address switch and memory controller are reduced in proportion to the increase in clock frequency of chipset (16%).

CONCLUSION

The most difficult, apparently, is the problem of generating efficient code for the DSP. For VLIW-processor with a regular organization of the control word the main obstacle to parallelization of software teams is a lack of parallelism, so the efforts of the developers of compilers for them to focus mainly on improving his by expanding the areas of planning and removal of data dependencies and management. If successful, this task fast heuristic scheduling algorithms are planning to give acceptable results.

Establishment of an effective compiler for ILP-processor in each case remains a challenge. The complexity of the problem is related in part to the fact that the effectiveness of virtually all methods depends strongly on the hardware features of the processor (the degree of parallelism, the number of registers, support for conditional and speculative execution) and the nature of the compiled programs planning in the advanced fields of the most effective programs for the control character, in the nature of a computer program their effect is less significant.Method of reducing conflicts with exchanges with memory is the use of multiprocessor architecture with unequal access to shared memory (NUMA). This architecture allows you to significantly increase performance while executing tasks, but requires additional efforts related to privatization and localisation data, when threading one task. This area also is the subject of further research.

ACKNOWLEDGEMENTS

This Work Was Supported by the Deanship of Scientific Research, Jerash University

REFERENCE

[1] L. Pozzi., Compilation Techniques for Exploiting Instruction Level Parallelism, a Survey, Milano, ITALY, March 2013.
[2] Al Smadi, Takialddin. "Design 3-D ultrasonic spatial mouse using microprocessor system." Journal of Information and Optimization Sciences 31.5 (2010): 955-971.
[3] Al-Maitah, Mohammed, Takialddin A. Al Smadi, and Hani Qasem Rashrash Al-Zoubi. "Scalable User Interface." Research Journal of Applied Sciences, Engineering and Technology 7.16 (2014): 3273-3279.
[4] S.Alan Mahlke, Exploiting instruction level parallelism in the presence of conditional branch, Champaign,
[5] M. Gschwind, The cell Broadband Engine:Exploiting Multiple Levels Of Parallelsim in Chip Multiprocessor, International journal of Parallel Programming, 2007.

[6] Rajagopalan S, Vachharajani M,. Malik S. Handling Irregular ILP Within Conventional VLIW Schedulers using Artificial Resource Constraints. - CASES'00, November 17-19, 2000, San Jose, California.
[7] Rao S, Stallman R. Electrical and Computer Engineering, June 2000, USA
[8] Zalamea J, Losa J, Ayguade E, ValeroM. Software andhardware techniques to optimize utilization in VLIW architectures. Int J Parallel Program 32(6):447–474, (2004)
[9] Ravindran R, Senger R, Marsman E, Dasika G, Guthaus M, Mahlke S, Brown R .Partitioning variables across register windows to reduce spill code in a low-power processor. IEEE Trans Comput 54(8):998–1012, (2005)
[10] Abderazek BA, Kawata S, Sowa M. Design QueueCore. and architecture for an embedded 32-bit, 2(2):191–205, (2006).
[11] Tayson G, Smelyanskiy M, Davidson E. Evaluating the use of register queues in software pipelined loops. IEEE Trans Comput 50(8):769–783, (2001).
[12] Al Smadi, Takialddin A. "Computer application using low cost smart sensor." International Journal of Computer Aided Engineering and Technology 4.6 (2012): 567-579.
[13] Hennessy J, Patterson D .Computer architecture: a quantitative approach. Morgan Kaufman, SanMateo, (1990).
[14] Allen R, Kennedy K .Optimizing compilers for modern architectures. Morgan Kaufman, San Mateo, (2002).
[15] Dehnert, J., Hsu, P. and Bratt, J. (1989) Overlapped loop support in the Cydra 5. In Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS- 3), pp. 26–38.
[16] Mahlke, S., Lin, D., Chen, W., Hank, R. and Bringmann, R. (1992) Effective compiler support for predicated execution using the hyperblock. In Proceedings of the 25th Annual International Symposium on Microarchitecture (Micro-25), December, pp. 45–54.
[17] Ravindran R, Senger R, Marsman E, Dasika G, Guthaus M, Mahlke S, Brown R .Partitioning variables across register windows to reduce spill code in a low-power ocessor. IEEE Trans Comput , 54(8):998–1012, (2005).
[18] vijay S,sarita A,kieth.D,willy,.kenneth E. exploiting instruction level parallelism on memory system performance Houston texas,2002.