



LOVELY
PROFESSIONAL
UNIVERSITY

**EFFECT OF BRACHING & VECTORIZATION ON PROCESSING SPEED OF
VLIW PROCESSOR**

A DISSERTATION PROPOSAL

Submitted by

BHARAT SHARMA

(11307683)

TO

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

In Partial Fulfilment of the requirement for the

Award of the Degree of

Master of technology in Computer Science

Under the guidance of

Mr Deepak Kumar

(April 2015)

School of: Computer Science and Engineering

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the student : Bharat Sharma
Batch : 2013-2015
Session : 2014-2015

Registration No : 11312842
Roll No : RK2306B79
Parent Section : K2306

Details of Supervisor:
Name : Deepak Kumar
UID : 17647

Designation : Assistant Professor
Qualification : M.Tech
Research Exp. : 2 years

Specialization Area: Computer Architecture (pick from list of provided specialization areas by DAA)

Proposed Topics:-

1. An efficient approach for enhancing the speed of a processor.
2. Enhancing the parallel processing using pipeline.
3. Exploring caching technique for power efficient processors.

Deepak Kumar
13/07/14
Signature of supervisor

PAC Remarks:

Topic '1' is approved.

11/07/14
Signature
13/07/14

13/07/14
25/07/14

APPROVAL OF PAC CHAIRMAN

Signature:

Date:

*Supervision should finally encircle one topic out of three proposed topics and put up for an approval before Project Approval Committee (PAC).

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to supervisor.

Abstract

Since the microprocessor was invented from that time the major work researchers are doing is to improve performance of the microprocessor. The improved performance can be gained by so many ways includes increase in clock cycle speed, less power consumption, applying multiple processors in one CPU, implementing parallel processing techniques and new technique like adding multiple cores to the same chip either having shared memory cache or dedicated memory cache. Parallel processing is the unbeatable technique gives more processing speed without increasing cost of hardware so much. We propose an idea of implementing an array based VLIW processor having both instruction level parallelisms (ILP) and data level parallelism (DLP). Also we examine the test results with compiler and simulation framework TRIMARAN. This approach can be very useful for high performance computing (HPC) applications like video conferencing, speech processing, weather forecasting and military applications where huge amount of data is needed to be processed very fast.

CERTIFICATE

This is to certify that Bharat Sharma has completed M.Tech dissertation titled **Effect Of Branching and Vectorization on processing speed of VLIW processor** Under my guidance and supervision. To the best of my knowledge, the present work is result of his original investigation and study. No part of the dissertation is fit for the submission and the partial fulfillment of the conditions for the award of M.Tech computer science and engineering.

Date : 29/04/2015

Signature of Advisor

Name : Deepak kumar

UID:17647

Acknowledgement

I would like to express my deepest appreciation to all those who provided me support to complete this report. A special gratitude I give to my guide Mr. Deepak Kumar (assistant professor) whose contribution in stimulating suggestions and encouragement helped me to coordinate my project especially in writing this report.

Furthermore I would also like to acknowledge with much appreciation the crucial role of all faculty members and HOD Mr. Dalwinder Singh for their comments and advices. And also thank to my friends who helped me in small but important improvements in the report.

DECLARATION

I hereby declare that the dissertation entitled, **Effect of Branching and Vectorization on processing speed of VLIW Processor** submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date : 29/04/2015

Bharat Sharma

Regn. No. 11312842

Table of contents

1. Introduction
2. Literature review
3. Present work
4. Implementation
5. Performance evaluation and results
6. Conclusion and future works
7. references

List of Figures	page no.
Figure 1: Pipeline Design of Scalar Processor	3
Figure 2: Pipeline Design of Vector Processor	3
Figure 3: Addition done By Scalar Processor	4
Figure 4: Addition done By Vector Processor	5
Figure 5: VLIW Processor Internal Design	6
Figure 6: A Simple VLIW Pipeline Design.....	8
Figure 7: Trimaran Internal working Architecture	18

List of Tables

Table 1: Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of Total Execution Cycle.

Table 2: Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of IPC.

Table 3: Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of branch taken.

List of Graphs

Graph 1: : Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of Total Execution Cycle.

Graph 2: Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of IPC.

Graph 3: Comparison of Scalar, VLIW & Array Based VLIW Processor Speed in terms of branch taken.

Chapter 1

Introduction

The processor also called the microprocessor or a CPU (central processing unit) , is the brain of the personal computers. Processor is responsible for all general computing tasks and coordinating done by memory and other devices. This CPU is a very complex chip that applies directly on the motherboards of most of the PCs.

Processor design

A processor in the sequence is more or less a program processor executes a function that is a group of low-level commands, operating systems and user applications, such as programs implemented. How to efficiently and effectively execute its micro-architecture processor instruction called interior design, is up. CPU micro-architecture CPU executes instructions that determines how fast. The exterior design of the processor, especially those in its external interface external cache, main memory, chipset, and other system components can communicate with the information that determines how fast.

- floating-point unit (FPU): - The logic unit can do much faster than general-purpose non-integer calculations adapted to perform a logical unit.
- Primary cache: - also called Level 1 or L1 cache, the same pace as the primary cache of CPU that is a small amount of very fast memory.
- Bus Interfaces: - Bus interfaces that connect the processor to memory and other components are way.

Processor performance

The turnaround time taken by a computing (PC) system to execute a particular program is determined by these factors:-

Effect of branching and Vectorization on processing speed of VLIW processor

- The processor clock cycle time,
- The number of micro-processor instructions required to perform that application
In million instruction per second (MIPS).
- The average number of processor cycles required to execute an instruction (CPI).

Improved system performance or reduce one or more of these factors are of increasing. In general, the clock cycle time in a better way to implement parallelism technology and memory hierarchy can be reduced. The number of instructions the compiler design and software design, and the average number of cycles per instruction (CPI) through the processor and system architecture design is reduced through adaptation is reduced.

Vector processors

One important aspect of vector processor architecture basic data types a specific word processor word size, where the N word, conventional scalar processors .in multiple data sets (SIMD) architecture on the same instructions, and the instruction set is made up of different Words that work instructions. Now the vector architecture, it uses vector data types, and Vector N-bit vector array is a collection of length, and Cray was invented in architecture, which is a vector file is not registered, as before vector machine operates on vectors stored in the main memory.

Effect of branching and Vectorization on processing speed of VLIW processor

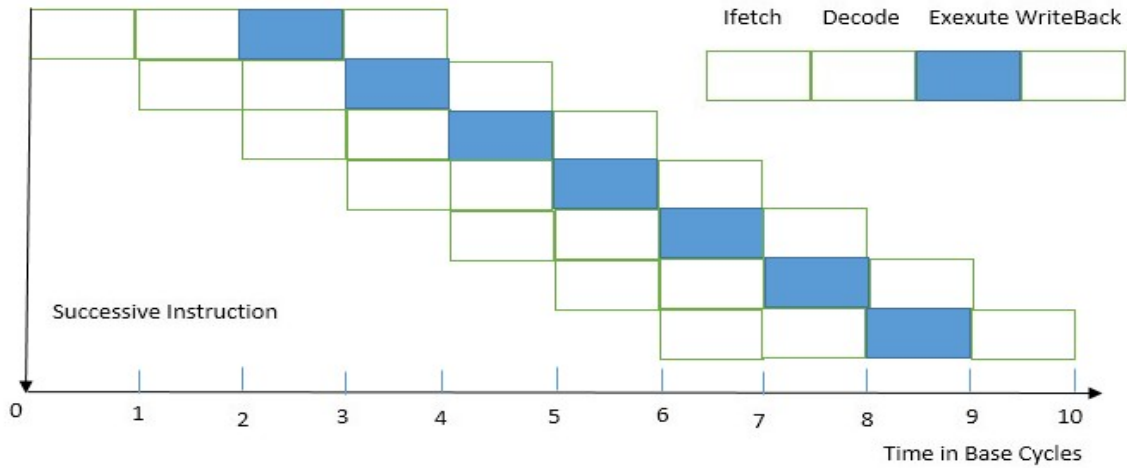


Figure 1: pipeline design of scalar processor

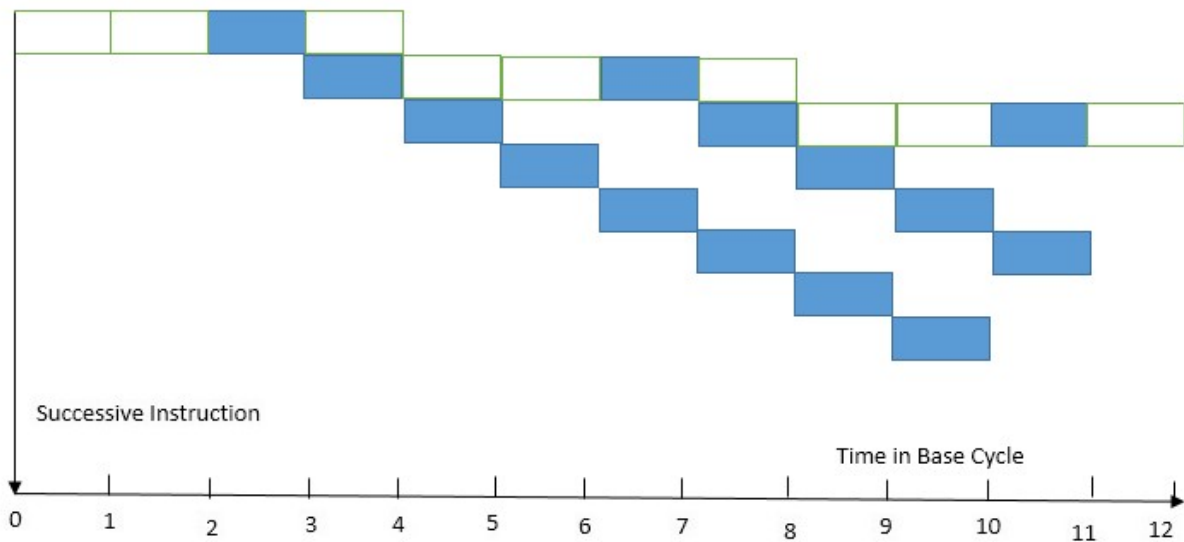


Figure 2: pipeline design of vector processor

The processor can work on just one instruction in the vector is a vector, the vector instructions to complete the operand vectors are not the only elements that means. It reduces the number of instruction are fetched as fetch and decode I reduces bandwidth. Vector processor complex scientific and data-level parallelism in today's multimedia applications shows. Based on these two parts of the instructions to register and register for memory-to-memory bring into.

Pros of vector processors

- They bring pressure on I-command at least the same amount of work required to specify the means.
 - Check all the dependences on the data set predictive branches very simple hardware means reduced the number of branches.
 - Create more effective means of instruction memory accesses vector load and store instructions use the memory as high latency is reduced using a regular pattern specifies.
 - A vector instruction stream execution, only the functional units (FU) and needed to be operated or buses, unit, means the unit starts to decode a bar •, can be locked with the power to reorder buffers.
- The program is used to reduce the size of the reduced number of instructions; It is also directed to executing a loop hides many branches.
- Require that each data item is used by the processor to use as cache memory because • Vector is no wastage.

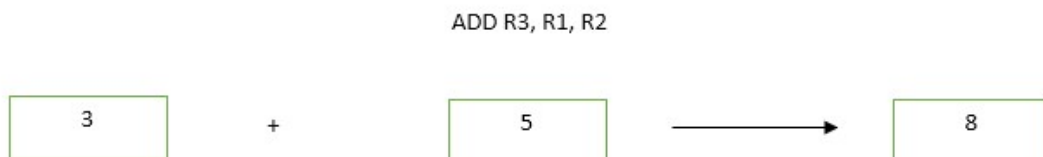


Figure 3: addition done by scalar processor

Effect of branching and Vectorization on processing speed of VLIW processor

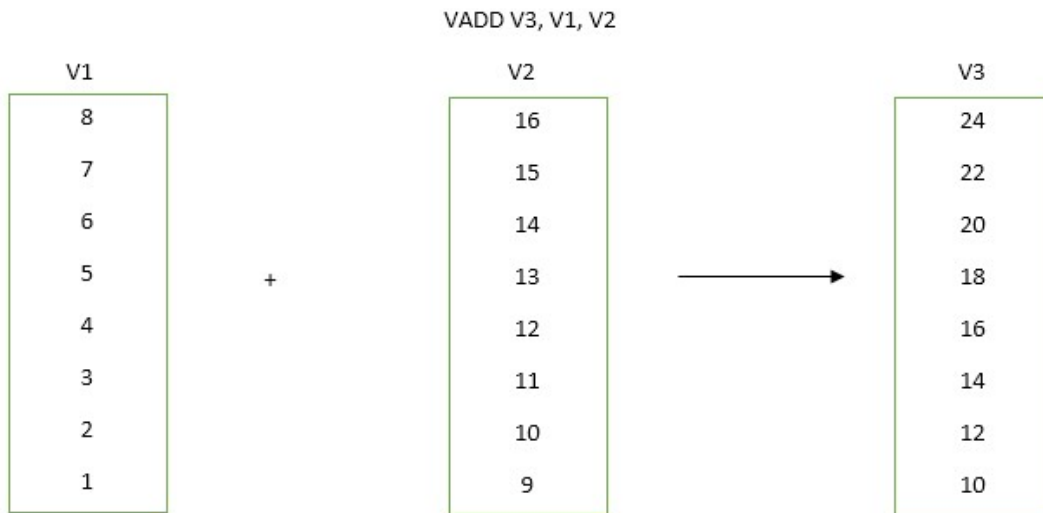


Figure 4 : Addition done by vector processor

VLIW processors

Superscalar processor composed of several operations similar to these processors by implementing a long instruction word similarity reflects the level of education. This very long instruction word too many math, logical and control operations which included multi-ops, as the saying goes. Each of these functions can be executed on perhaps simple RISC processor that person is involved in the operation. VLIW composed of several operations by executing a long instruction word of instruction-level parallelism (ILP) tries to achieve that is a type of microprocessor architecture.

Effect of branching and Vectorization on processing speed of VLIW processor

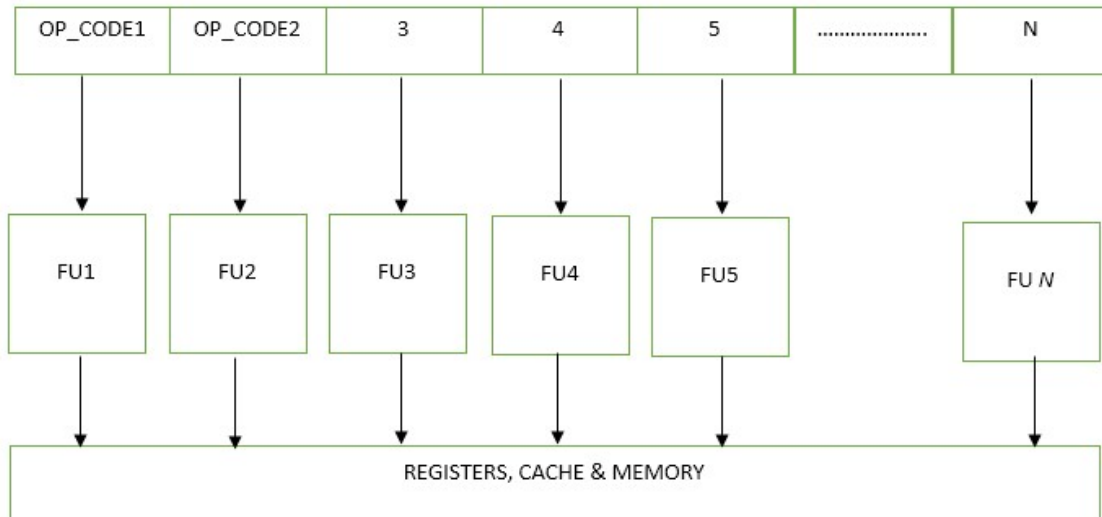


Figure 5: VLIW processor internal design

Both can apply for a program instruction level parallelism, the VLIW architecture, superscalar processor architecture compared to the very high performance at low cost offers. VLIW architecture-based processors, superscalar processors but mostly in terms of the dynamic scheduling and operations nor again do not follow orders because simple control logic that is why it is very low complexity and cost, that is. The main target hardware complexity, less design time, improved performance is to reduce the small clock cycle time.

The compiler such as some standards should be complex enough to support:

- It should detect threats and hide latency.
- The structural risk should handle.
- No two operations at the same time should be given to FU.

VLIW processor architecture is mainly generalized from two well established concepts which are: horizontal micro-coding and superscalar processing. A typical VLIW processor machine has instruction words of hundreds of bits in length, multiple functional units are used simultaneously working with use of a common large register file. Operations can be

executed by multiple functional units together which are synchronized in a VLIW instruction like 256 or 1024 bits per instruction word, implemented in multi-flow computer models.

The VLIW concept is mainly borrowed from horizontal micro-coding. Different fields of the long instruction word carries the op-codes dispatched to different functional units. Some programs written in conventional short instruction words like 32 bits must be compacted together from the VLIW instructions. That code compaction must be done by a compiler which should be able to predict branch outcomes using elaborate heuristics or run-time statics.

Pros of VLIW processors

- Superscalar processor architecture compared to the very high performance at low cost.
- Because no dynamic scheduling and operations is very low complexity and cost of a re-ordering.
- The direction has been the creation of a separate machine operation
- These instructions can start in parallel
- Several parallel functional units and large register set

Pipelining in VLIW processor

The execution of the instructions by an ideal VLIW processor was shown in figure 6. Each instruction specifies multiple operations, the effective CPI will be according to the degree of the VLIW processor like if degree is 3 the CPI will be 0.33. VLIW machines behave much like superscalar processor machines with three differences which are:

First, the decoding part of VLIW instructions is easier than of superscalar processor instructions. Second, the code density of the superscalar processor machine is better when the available instruction level parallelism is less than that is exploitable by VLIW processor machines. This is because the fixed VLIW format includes bits for non-executable operations, while the superscalar processor issues executable instructions only. Third, a

superscalar machines can be object code compatible with a larger family of non – parallel machines, but the VLIW machines exploit different amount of parallelism would require different instruction sets.

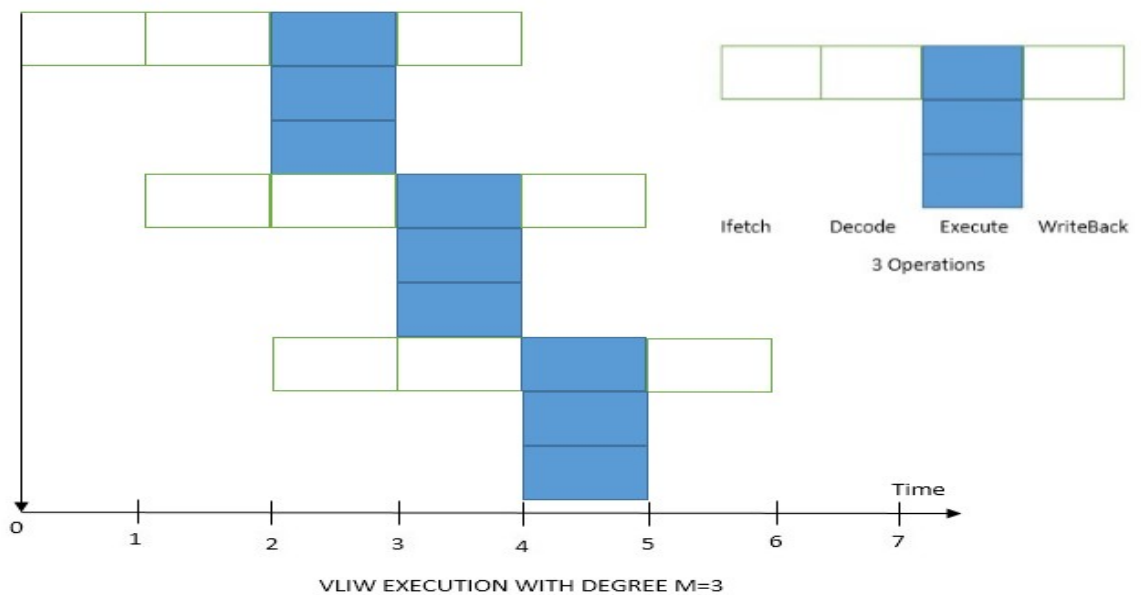


Figure 6: A simple VLIW pipeline design

Motivation

This is a good thing that instruction level parallelism (ILP) and data level parallelism (DLP) are both can be used to make fast processor that rely on the high performance computing (HPC). Two main strategies to make the processor speed faster are first improve the semiconductor technology and another is use of parallel processing techniques in an efficient way. the VLIW and the superscalar are two basic architecture designs those uses the parallel processing. The VLIW processor architecture does multiple issues of instructions statically by compiler where in

superscalar processor architecture does multiple issues of instructions but in dynamic way by extra hardware. That's why the hardware complexity and the cost is much more in superscalar processor architecture as compared to VLIW processor architecture and performance is more in VLIW than superscalar due to this less hardware complexity. Very long instruction word or VLIW is a one kind of processor design that tries to achieve high level of parallelism by executing long instruction word composed by multiple operations.

Now a day's computer system for DSP and multimedia applications are major workloads. 3D graphics, video conferencing, military applications, weather forecast, speech processing, speech recognition and mobile multimedia applications such as VLIW or superscalar processors as processing speed than conventional multi-thread processor demands. Some of the rich applications of DSP vector processing needs. Instruction-level parallelism as well as data-level parallelism, both to provide an architecture that can be made, then that type of architecture for such application can provide sufficient computing speed. Single instruction and multiple data stream display always I (SIMD) support as well as education levels, which promotes the concept of array-based VLIW processors have given .in my thesis is a major issue in processor design. Array-based VLIW long instruction word (Multi-OP) are all operand vector, is made up of many operations that have killed, is just an extension of the VLIW processor design. This research trimaran tool I use some of the standard array-based VLIW processor performance evaluated.

Instruction Level Parallelism (ILP)

ILP is concerned with equality between independent instruction. The primary goal of parallel processing to reduce the execution time of a program in order to improve the performance of the microprocessor independent hardware units is to execute multiple operations simultaneously. ILP parallel processing is slightly different. Individual machine operation (MUL, add load), although some of the different processors and each task is to get all together and are executed in parallel with the processing concern is the question of overlap with the ILP. Instruction-level parallel processing sequential instructions to identify and independent with a stream of instructions executed by a comprehensive program to reduce the execution time is a combination of software and hardware technologies. Even through instruction-level parallelism connected

through functional units (3-integer unit, 2-load / store unit, 2-arm unit, etc.) may require more number of machines for example with ILP demands more hardware complexity, improve system performance, single / cluster register file.

High level sentence as a sample:

$p = q * r - s/t + u * v - w \% x;$

The operator% in C modulus operator here the remainder of the division of X by W yields. * - And + operators respectively, subtract and addition operators are multiplied. The right hand side of the tree can be represented by the following expression is expression. The tree-top level four operations provided several math units can be carried out in parallel to the inspection. The results of these actions can be done in parallel again that actions are investing. However, here are two of a number of independent operations. Present day processors to support this kind of equality in fact many math units. Equality of this kind exists at the level of instruction, because the instruction level parallelism (ILP) is called. Hardware compiler technology available in several functional units for parallel execution instructions and schedule to perceive the similarity is quite advanced.

Dependency among Instructions

A program that can be exploited by the amount of similarity can be determined in a cycle that depends on the number of independent directions. So ILP is limited due to the dependencies between instructions.

Dependence with 4 types of programs may exist between instructions.

1.Name dependence

One. Output dependency: instruction i and j write the same register or memory location. To order or to leave the correct value should be protected.

Like: $i = \text{add } r7, r4, r3$

$J = r7, r2, r8$

B. Anti-dependency: that education instruction reads a register or memory location j i writes.

I= add r6,r5,r4

J= sub r5,r8,r11

2.Data dependence

I hold both the instruction following an instruction j is dependent on the data:

Instruction by instruction I j I or II may be used that produces a result. Jammu and Kashmir instructions instruction is data dependent, and instruction K instruction i is dependent data

3. Control dependence

I correct program instructions can be executed in order, so that a control with respect to the dependence of a branch instruction, an instruction I order restricted

Ex: If P1 {

S 1;

};

If P2 {

If S2;

};

Here is dependent on S1 P1 and S2 P2 and S2 P1 can not be earlier than scheduled, can not be scheduled before the S1 is up to P2.

Dependence directive imposes control two barriers

Depending on the instructions of a branch instruction control, then I, then education can not be executed before the branch instruction.

II. Depending on the instructions of a branch instruction does not control, the instruction after the branch instruction can not be executed.

4.Resource dependence

It is still used by a previously issued instructions being who requires a hardware resource is dependent on an instruction resource is on a previously issued instructions.

Ex: -

div R1, R2, R3

div R4, R2, R5

ILP Architecture

ILP's ultimate target is required to make a decision .before the hardware to the compiler about the program by independent operation or a series of instructions to perform independently. ILP-based methods for equality hardware architecture of the information transmitted by the compiler can be three types of

1. sequential architecture

This is clearly directed between hardware and instructions are determined it is not indicative of dependence between .dependencies the compiler architecture. Although the compiler to exploit hardware parallelism within the program instruction to facilitate re-organize. Dependency and the issue of instructions is dynamically by both finding hardware.

Ex: - superscalar processor architecture exhibition like this

2. Dependence Architecture

This type of architecture ILP compiler identifies dependencies between instructions and hardware to express it. An instruction will be free to determine at what time and in order to

release them accordingly. The main objective of this type of architecture operand and resources (functional unit) are available when a directive as soon as possible to schedule.

Ex: - Dataflow processor architecture are such.

3. Freedom Architecture

Statically compiler to exploit parallelism in the program is responsible. Hardware on the issue of multiple independent instruction does not need to take a decision. Compiler, the cycle needs to be issued special instructions, which may specify the functional unit

Ex: - VLIW processor number of independent operation by the compiler are bundled together into a single instruction, the ILP is based on this type of architecture.

Data Level Parallelism (DLP):-

Data convergence of computing across multiple processors in parallel computing environment is a form of parallelization. Data parallelism is focused on distributing the data across different parallel computing nodes. It is another form of equality as sameness to act contrasts.

Data-level parallelism (D) instruction-level parallelism is more than a special case. DLP is working together to take action on several units. DLP is a classic example of each pixel processing (such as brightening) is independent of the people around it, which is performing an operation on an image. Vector data processing level is an example of equality.

Chapter 2

Review of Literature

The main focus on the possibility of Vectorization on the VLIW processor architecture to gain the advantage of both instruction level parallelism (ILP) and data level parallelism (DLP) in an application program. This effort is done already to some extent but a lot of areas are remaining untouched.

The efforts which are already been made are as follows in these research papers and articles. In research paper [11] it has shown that instruction level parallelism and data level parallelism both can be merged together into a single architecture to execute the vectorizable code with more efficient way. That architecture exploits high performance equivalent to 10 issue superscalar processors means the execution is that much faster according to speedup of the machine with comparison to ILP only architecture machine. The highest instruction per cycle (IPC) rate can be achieved is 10 by this research work.

Research paper [18] says that DLP has some drawbacks like it's not truly general purpose method that can be used in any application, and also says about the future challenges in this ILP and DLP combination machine like memory access time and wire delays. Research paper [12] says about the Micro-architecture of a superscalar processor, having more capability of executing more than one instruction in one clock cycle by converting an sequential program into a parallel principle and constraints. the conclusion and future work says that now the time of extension of super scalar processors is almost over, so the next comes VLIW processor architectures to implement ILP.

Research paper [6] shows a new efficient architecture clustered organization for decoupled execution (CODE) to remove or overcome some existing limitations in conventional vector processors up to some extends. These limitations are, first is the complexity and size of vector register file limits the number of functional units(FU), second is that the precise executions for

vector instructions(VI) are difficult to implement, third is vector processors requires an expensive on-chip memory system of high capacity storage that support high bandwidth at low access latency. This CODE architecture is 26% faster than a vector processor with a centralized register file, with a disadvantage of 5% performance degrade.

Research paper[5] says, vector processors can be made scalable for embedded systems unlike superscalar and VLIW processors which are not fit for data level parallelism applications. A new compiler VIRAM (with help of IRAM research group) is introduced to evaluate the multimedia benchmarks.

Second are the basic and other related works in this field which are essential for students to take a small knowledge about this field of microprocessor working.

Research paper [22] says Most of the wire pipelines are edge triggered flip-flops based. The study on these wire pipelines is necessary because of increasing wire delay, shrinking clock period and growth in chip size. The main thing is that the use of level sensitive latches has advantages of latency and area, cost and comparison in flip-flop based and register based software pipelines can be done easily.

Research paper [21] gives a new architecture and compiler approach that will help in I –fetch that is a instruction register file (IRF) ,frequently occurring instructions can be placed in this register file. Multiple entries in this register file can be referenced by a simply single packed instruction in ROM or L1 instruction cache .The experimental results shows improvements in space and power when using 32 entries. It is difficult to enhance one parameter by not negating the other connected parameter. Like if we increase clock frequency to enhance the performance then it also enhances the power consumption,code compression techniques will enhance the complexity of the code means density of code will be more .so the main focus is on I-fetch operations which takes 36% power of processor of the total program in embedded systems. It is inefficient in information content of instruction encoding.

Research paper [1] gives a new proposed method for defining instruction level energy estimation frame-work for mainly VLIW processorsto define an effectiveness energy-aware Software optimization for instruction level parallelism (ILP) is the key issuethis approach claims to reduce the

Effect of branching and Vectorization on processing speed of VLIW processor

complexity of characterized problem of VLIW processor for exponent values ,with respect to no. of parallel instructions and operations.

The given approach says that it results with an average error of 19 % with using standard dev. Of 5.8%, so it provides the average energy saving 12%.

Conclusion: the basic and specific study which is required to get knowledge for research work is presented here in form of the literature review.

Chapter 3

Present work: objectives and methodology

Objectives

- The main Objective of this research is to design and measure the performance of a statically multi issue processor by taking the advantages of vector processing .The design of new processor can be called as a “array based VLIW processor”.
- Another objective is to compare the performance of “array based VLIW processor” with Vector and VLIW processor architecture. One major aim is to exploit both ILP and DLP from application program in order to enhance the performance.
- This “array based VLIW processor” can be one of the promising architecture in High Performance Computing (HPC) application. It can be widely used in dynamic Multimedia application where a huge amount of data needs to be processed in very short period of time.
- The effect of branching on the processing speed of a processor, how the program execution is done with and without branching.

Methodology

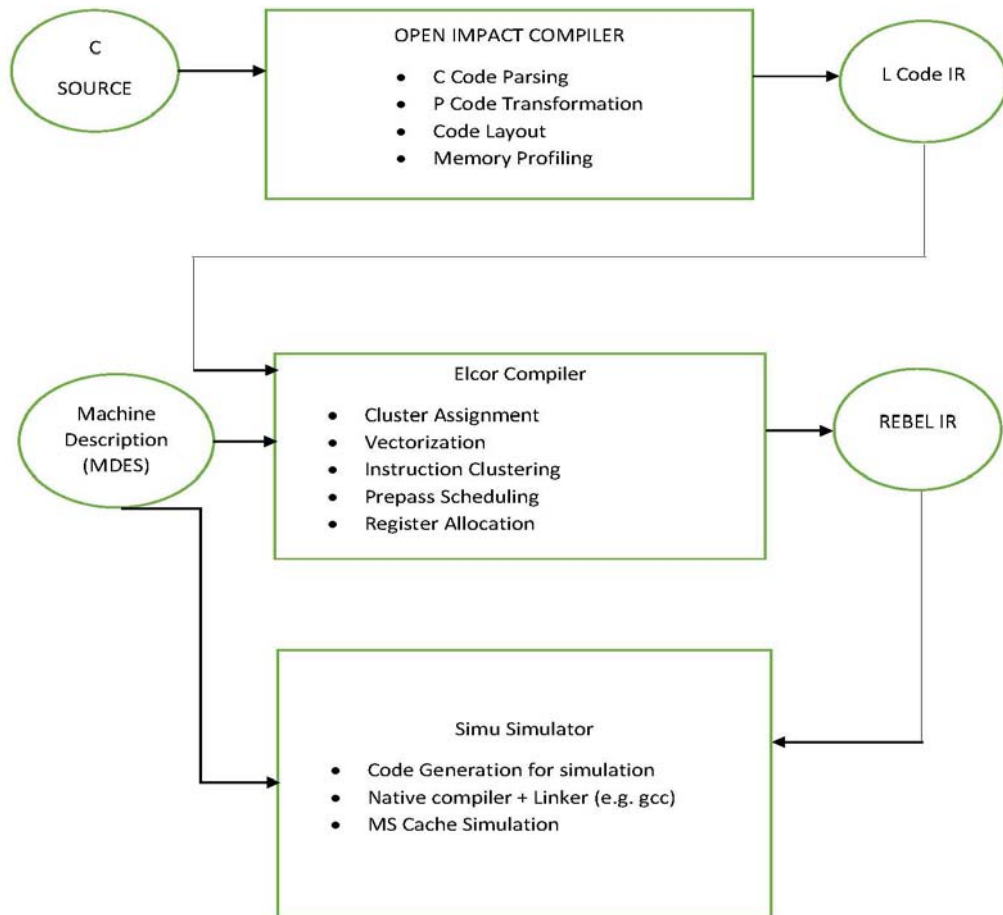
The base of the proposal of combining two techniques those are already very much efficient to perform a specific kind of task is that if two best approaches are mixed two use it in a specific manner they will form a better approach to perform those specific tasks. Simply says “two heads are better than one “. The framework to be used to test results and compare the total execution time taken by a processor to execute a single program having multiple different type of instructions is TRIMARAN version 4.0 a compiler and simulator infrastructure for research in embedded system and EPIC architecture. Because the TRIMARAN framework is best suitable for this kind of research as it has inherently built in tool for compiling and simulating and also it can target variety of ILP processor.

The main components of the framework are as follows:

1. Open-IMPACT compiler
2. Elcor compiler
3. Simu simulator
4. SUIF (vectorizer)
5. Mdes (Machine description)
6. Example benchmarks

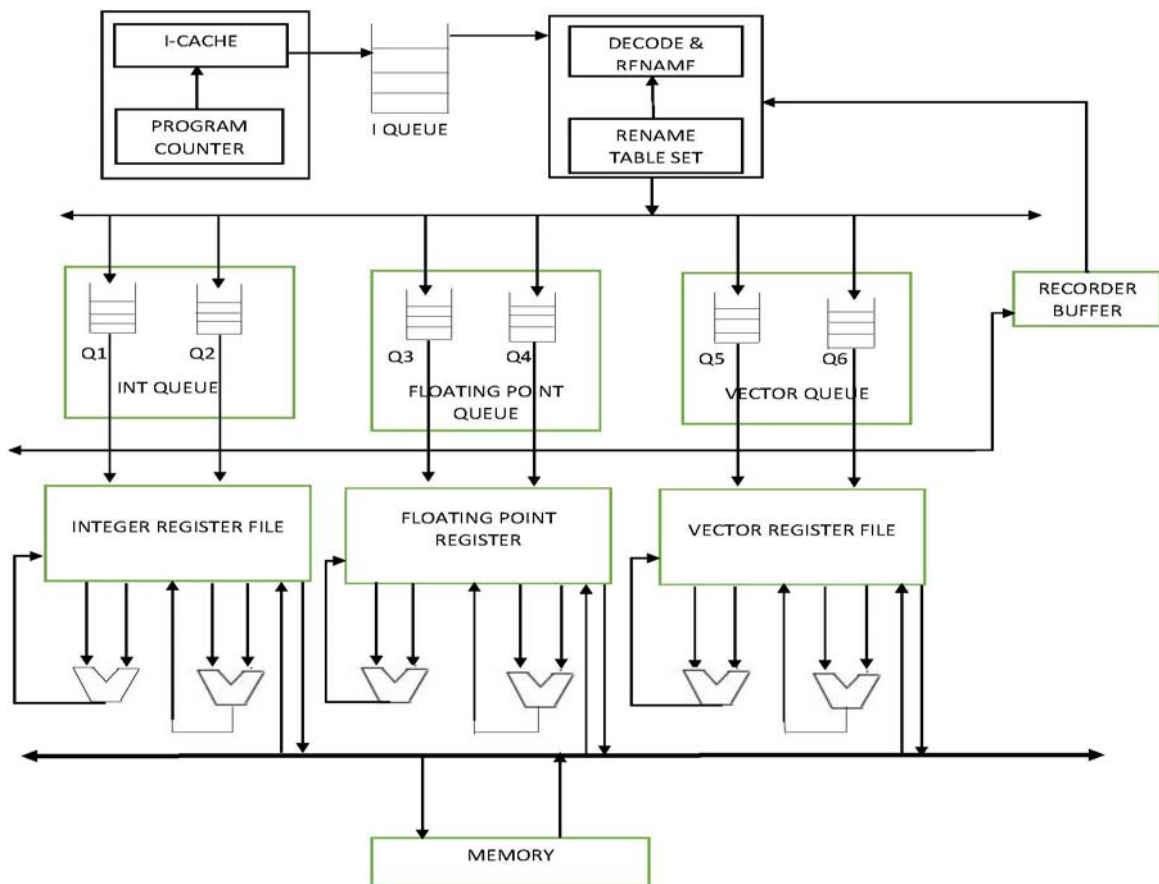
Effect of branching and Vectorization on processing speed of VLIW processor

TRIMARAN SYSTEM ORGANIZATION
OVERVIEW OF COMPILATION STEPS IN TRANSMISSION



Proposed array based architecture: introduction

Array based VLIW processor can be mainly made to exploit ILP and DLP both. This thing can be implemented by following architecture.



Effect of branching and Vectorization on processing speed of VLIW processor

Some main points of this architecture are as follows:

1. multiple functional units are attached with a single global register file.
2. compiler itself dynamically issues the long instruction word.
3. each instruction consists of multiple parallel operations which are independent.
4. each operation requires statically known number of cycles to execute.

Architecture shows the execution and flow of the data and instructions for all four operations: I-fetch, Decode and rename, execution and write-back.

Chapter 4

Implementation

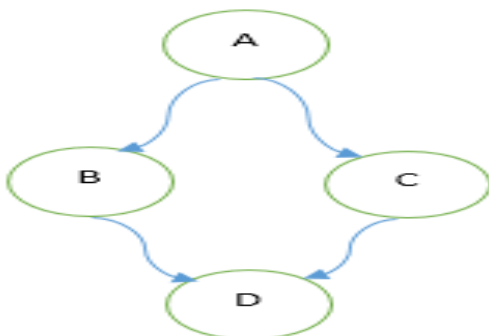
Introduction

The performance is evaluated by TRIMARAN compiler and simulation framework. By use of the simulation framework it can generate the control flow graph (CFG) and data flow graph (DFG). For different benchmarks there will be different dependencies arises and corresponding to that data flow and control flow is shown in the diagrams in TRIMARAN.

CFG and DFG

Instruction-level parallelism maximum but it can be minimized to some standard data flow and control flow dependencies between instructions should be. Program flow systems: conventional computer program execution order can be clearly stated in the user programs are based on the mechanism by which the flow of control. Data flow computer data operand data to be driven by the availability, which allows the execution of an instruction to operate the system are based. Data Computer fine grain level of education emphasizes a high degree of similarity flow. Reduction of demand for computers other computations based on its results of operation, which are based on a demand-driven system.

Control flow graphs (CFG): control flow graphs in computer science and technology are representation of all paths that might be traversed through a program during its execution using graph notation. In a control flow graph all nodes in the graph represented by a basic block and directed edges used for jumps from one node to another. A straight line code has no jump and execution will be smoother. Like $a \rightarrow b \rightarrow c \rightarrow d$ is for straight line. Or may be of this type.



Analysis of CFG and DFG:

CFG program to almost mechanical view is a very simple graph. CFG in each block within an instruction block to execute all instructions if such instructions executed corresponds to the linear block. The last instruction in the block will continue to block a jump instruction. The basic block is a unique entry code and zero or more exit edges between nodes represent the flow of control represent. Being so simple CFG excellent for both the control flow graph based optimization (unreachable code elimination and cross-linking) and whose structure is the target code to create exact duplicates of CFG's. Although other program counter CFG does not give any details on the progress of the computation compared.

CFG trimaran multiple compiler optimization and static analysis tool to control flow graph and dataflow graph Elcor component analysis is required for the various modules.

Flow control modules provide the following further analysis of CFG.

Vectorization

Vectorization displays and SIMD architecture, with support for the efficient execution of a program which is a technique to exploit the data-level parallelism. Trimaran as the technique is called selective vectorization. This is in order to improve resource utilization and performance scalar and vector processing units by distributing computing between education creates highly efficient schedule. This type of technology is mostly applied to multimedia applications. Vectorization by default, one at a time on a single thread application program, a series of actions that are modified to act, which is a special case of parallelization. Vectorization a vector processing instruction in both traditional computers and modern supercomputer is a key feature where a process conducted over several operands. Vectorization a vectorized compiler without human assistance programs that allow a process to convert scalar functions. Exploiting data-level parallelism from the application program.

Multi-thread processor vectorization for high performance computing applications are yet to find out. We are array-based VLIW processor architecture VLIW infrastructure in order to realize the many standard vectorize. In this type of architecture known as vector directions of each instruction operates on the array of numbers where a single cycle are issued. Array-based VLIW

processor vectorization s2lc used for the purpose we combine the advantages of VLIW processor and vectorization process. Open space of the impact of automatic vectorization stage S2lc. vectorization function of a vector instruction set processor that utilizes a parallel version is to convert a sequential loop. The shift operation is a major rearrangement.

HPL-PD and machine details

HPL-PD, especially in relation to the amount of equity offering, the separate structure and scale machines believe that is a parametric architecture. BE the compiler can be controlled so that HPL-PD architecture visible compiler executable resources and made hibernating. Code generators in HPL-PD (scheduler and register allocator) target epic architecture requires the following information.

Registration file structure. The number of register files and for each one, in that the number of registers and bit width is also included.

- Operation repertoire. An action consists of an opcode and a register per each tuple-an operand register. Instead of addressing mode operand opcode as part as part of the machine is supported by the target. An operand is a literal or a target machine does not have a register. Several instances of an operation that can perform operations that are consistent with the presence of several functional units machine. Operation repertoire each opcode, it can use the register set of files, opcode and specifies the repertoire. Implicitly, the more directly useful to a compiler that is in a way, it specifies the functional connectivity between units and register files.
- Clearly scheduled resources. The compiler two operations at the same time, do not attempt to use the same resources, it must ensure that the management of resources is set. Etc. Such functional units, buses, instruction areas, as
- Each operation of the resource use behavior. Compiler use the same resource is such that once the issue of any of the two operations, the use of resources relative to one another that they will not end up together to ensure that this information should be used. This operation may be issued with a set of the processor which determines.

- latency descriptors. Every operation, each source operand is read and written to each destination operand, the operation is issued that is relative to the time that specifies a latency descriptors. HPL-PD architecture within the space or files, log files to register a set of functional units, and a hierarchical memory system consist of a set. A special machine control register file, except for one or more of each type of register file can have. Each machine has a control register file no. Similarly, the operation of a particular machine HPL-PD in the repertoire of each operation can be one or more instances. Many examples of the many examples that the operation of an operation, ie, the machine can perform this operation in several functional units, which can be issued in parallel means. Vector processing as many similar functions are executed simultaneously. Parameters above all in order to target a specific architecture of Trimaran MDES may be specified in the file. The goal of our project architecture is array-based VLIW processor architecture.

Array-based VLIW architecture for VLIW processors, we have to specify all the parameters and all operand vector liner array of numbers that will be called as such action has been taken in vectorized mode. MDES vector potential target architecture are specified in the file. All vector register file or the file must be where the vector must be able to host each register within a register file. VLIW architecture based on our target array by changing some or all of the above mentioned parameters describe the high-level machine can be specified in the file. It may be that the low level of a script in the file conversion for the court using the compilation (*. Lmdes) would be understood by the compiler.

Machine description file structure are given below. All parameters of the target architecture is hmdes2 extended four files (*.hmdes2) use are described. One of these four files in the file it P.hmdes2.where P Our goal is to be considered as the processor architecture. P.hmdes2 file other 3 files (structure_pristine.hmdes2, hpl_pd_pristine.hmdes2 and hpl_pd_elcor.hmdes) are included. Elcor compiler MDES P.hmdes2 file present in the module corresponds to the main file which is hpl_pd_std.hmdes2. File P.hmdes2 (hpl_pd_std.hmdes2).

Machine description for array based VLIW processor

```
$def !num_clusters          4

// Register File sizes

$def !gpr_static_size      64
$def !gpr_rotating_size   64

$def !fpr_static_size      64
$def !fpr_rotating_size   64

$def !pr_static_size       64
$def !pr_rotating_size    64

$def !cr_static_size       64
$def !cr_rotating_size    64

$def !btr_static_size      64

// SLARSEN: Vector register files
$def !vir_static_size      64
$def !vir_rotating_size   64

$def !vfr_static_size      64
$def !vfr_rotating_size   64

$def !vmr_static_size      64
$def !vmr_rotating_size   64

// HPL-PD 2.0 Extn
// Literal Register File sizes

$def !short_lit_size       7
Vector length and functional units
$def !vec_length           4
$def !vec_integer_units   4
//$def !vec_float_units    4
$def !vec_integer_perm_units 4
//$def !vec_float_perm_units 4
$def !vec_integer_xfr_units 2
//$def !vec_float_xfr_units 2

// RMR: issue slots
$def !issue_slots          6

$def !memory_lit_size      10
$def !branch_lit_size     10
$def !long_lit_size        16
```

Effect of branching and Vectorization on processing speed of VLIW processor

```
$def !unrestricted_lit_size 32

// Per-cluster Functional Units

$def !integer_units      1
$def !float_units       1
$def !memory_units      1
$def !branch_units      1
Vector length and functional units
$def !vec_length        4
$def !vec_integer_units 1
//$def !vec_float_units  1
$def !vec_integer_perm_units 1
//$def !vec_float_perm_units 1
$def !vec_integer_xfr_units 2
//$def !vec_float_xfr_units 2
```

List of bench marks taken to check the performance

1. If then : to apply if then conditional statement
2. Fact2: to find factorial of a number
3. Fib: to find the Fibonacci series
4. Nested : to apply the nested loops
5. Sqrt: to find the square root of a number
6. Strcpy : to copy a string a no. of times
7. Eight : to run a code having eight variables

Statistics measured for a bench mark fact2:

1. When taken the scalar processor machine to run the code.

```
Function _main {
total_cycles.....63
compute_cycles.....63          (100.00)
  stall_cycles.....0           ( 0.00)
  total_dynamic_operations.....31
  total_static_operations.....79
  total_committed_operations.....31      (100.00)
  total_speculated_operations.....0      ( 0.00)
  total_notrapping_operations.....0      ( 0.00)
  total_exceptions_encountered.....0     ( 0.00)

  average_issued_ops/total_cycles.....1.17
```

Effect of branching and Vectorization on processing speed of VLIW processor

```

average_issued_ops/compute_cycles.....1.17

ialu:      9 ( 29.03)      dynamic    26 ( 32.91).....static
falu:      0(  0.00)..... dynamic    0 (  0.00).....static
load:      0 (  0.00).....dynamic    14 ( 17.72).....static
store:     0 ( 38.71).....dynamic    14 ( 17.72).....static
cmp:       0 (  0.00).....dynamic    1 (  1.27).....static
pbr:       3 (  9.68).....dynamic    6 (  7.59).....static
branch:    1 (  3.23).....dynamic    6 (  7.59).....static
icm:       6 ( 19.35).....dynamic    12 ( 15.19).....static
vialu:     0 (  0.00).....dynamic    0 (  0.00).....static
vvalu:     0 (  0.00).....dynamic    0 (  0.00).....static
vload:     0 (  0.00).....dynamic    0 (  0.00).....static
vstore:    0 (  0.00).....dynamic    0 (  0.00).....static
vxfr:      0 (  0.00).....dynamic    0 (  0.00).....static
vperm:     0 (  0.00).....dynamic    0 (  0.00).....static

spills_restores: 24 ( 77.42).....dynamic    52 ( 65.82).....static
caller_save: 21 ( 87.50).....dynamic    10 ( 19.23).....static
callee_save: 3 ( 12.50).....dynamic    42 ( 80.77).....static
}

```

2. When taken the VLIW processor machine to run the code.

```

Function _main {

total_cycles.....55
compute_cycles.....55          (100.00)
  stall_cycles.....0          (  0.00)
  total_dynamic_operations.....31
  total_static_operations.....79
  total_committed_operations.....31          (100.00)
  total_speculated_operations.....0          (  0.00)
  total_notrapping_operations.....0          (  0.00)
  total_exceptions_encountered.....0          (  0.00)

average_issued_ops/total_cycles.....1.64
average_issued_ops/compute_cycles.....1.64

ialu:      9 ( 29.03)      dynamic    26 ( 32.91).....static
falu:      0(  0.00)..... dynamic    0 (  0.00).....static
load:      0 (  0.00).....dynamic    14 ( 17.72).....static
store:     0 ( 38.71).....dynamic    14 ( 17.72).....static
cmp:       0 (  0.00).....dynamic    1 (  1.27).....static
pbr:       3 (  9.68).....dynamic    6 (  7.59).....static
branch:    1 (  3.23).....dynamic    6 (  7.59).....static
icm:       6 ( 19.35).....dynamic    12 ( 15.19).....static
vialu:     0 (  0.00).....dynamic    0 (  0.00).....static
vvalu:     0 (  0.00).....dynamic    0 (  0.00).....static
vload:     0 (  0.00).....dynamic    0 (  0.00).....static

```

Effect of branching and Vectorization on processing speed of VLIW processor

```

vstore:      0 ( 0.00).....dynamic      0 ( 0.00).....static
vxfr:        0 ( 0.00).....dynamic      0 ( 0.00).....static
vperm:       0 ( 0.00).....dynamic      0 ( 0.00).....static

spills_restores: 24 ( 77.42).....dynamic      52 ( 65.82).....static
caller_save:  21 ( 87.50).....dynamic      10 ( 19.23).....static
callee_save:  3 ( 12.50).....dynamic      42 ( 80.77).....static
}

```

3. When taken the array based VLIW processor machine to run the code.

```

Function _main {

total_cycles.....10
compute_cycles.....10          (100.00)
  stall_cycles.....0          ( 0.00)
  total_dynamic_operations.....31
  total_static_operations.....79
  total_committed_operations.....31          (100.00)
  total_speculated_operations.....0          ( 0.00)
  total_notrapping_operations.....0          ( 0.00)
  total_exceptions_encountered.....0          ( 0.00)

  average_issued_ops/total_cycles.....3.10
  average_issued_ops/compute_cycles.....3.10

ialu:        9 ( 29.03)      dynamic      26 ( 32.91).....static
falu:        0 ( 0.00)..... dynamic      0 ( 0.00).....static
load:        0 ( 0.00).....dynamic      14 ( 17.72).....static
store:       0 ( 38.71).....dynamic      14 ( 17.72).....static
cmp:         0 ( 0.00).....dynamic      1 ( 1.27).....static
pbr:         3 ( 9.68).....dynamic      6 ( 7.59).....static
branch:      1 ( 3.23).....dynamic      6 ( 7.59).....static
icm:         6 ( 19.35).....dynamic      12 ( 15.19).....static
vialu:       0 ( 0.00).....dynamic      0 ( 0.00).....static
vfaluc:      0 ( 0.00).....dynamic      0 ( 0.00).....static
vload:       0 ( 0.00).....dynamic      0 ( 0.00).....static
vstore:      0 ( 0.00).....dynamic      0 ( 0.00).....static
vxfr:        0 ( 0.00).....dynamic      0 ( 0.00).....static
vperm:       0 ( 0.00).....dynamic      0 ( 0.00).....static

spills_restores: 24 ( 77.42).....dynamic      52 ( 65.82).....static
caller_save:  21 ( 87.50).....dynamic      10 ( 19.23).....static
callee_save:  3 ( 12.50).....dynamic      42 ( 80.77).....static
}

```

Conclusion:

This is clear that the TRIMARAN framework is very much suitable for this kind of research so these parameters are to be passes into the framework and the results are shown in next phase.

Chapter 5

Performance evaluation and results

Parameters for different architectures are passed to the framework and the results coming in the following way.

Performance matrices

The improved performance can be gained by so many ways like increase in clock cycle speed, less power consumption, applying multiple processors in one CPU, implementing parallelism techniques and new technique like adding multiple cores to the same chip either having shared memory cache or dedicated memory cache. Parallel processing is the unbeatable technique without increasing cost of hardware so much.

1. IPC : instructions per cycle

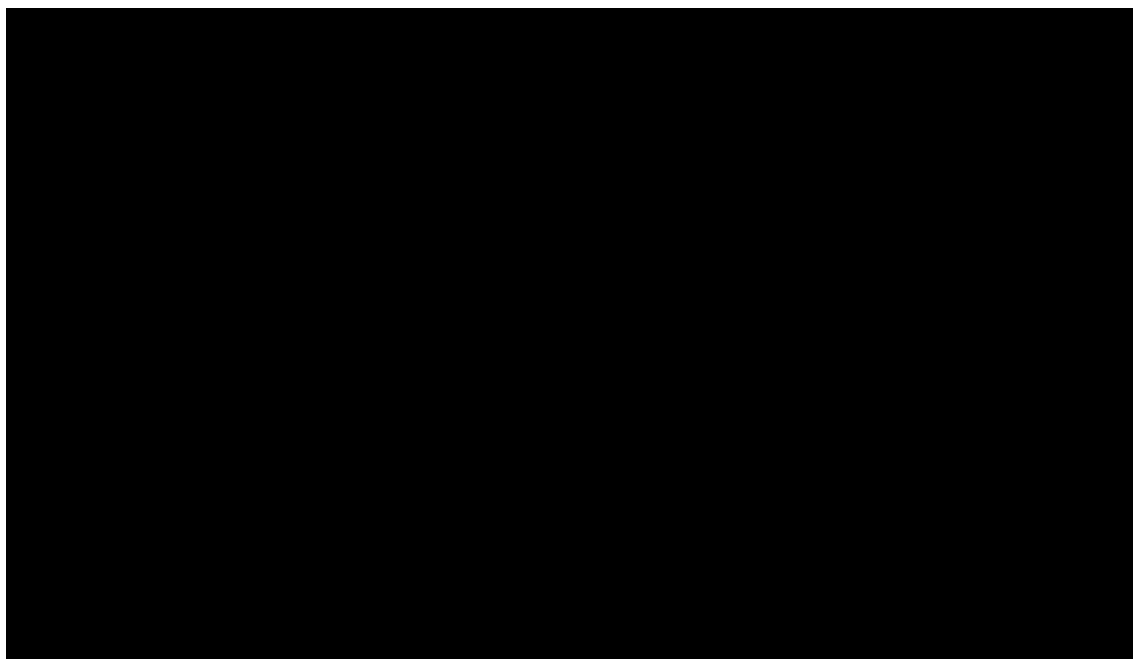
Meaning of IPC is that the number of executed instructions by a processor within single cycles. It is generally calculated in terms of million instructions per cycle(MIPS rate).

2. TCC: total compute cycles

Meaning of the TCC is that the number of cycles taken by a processor to execute one single program.

Performance measurement and results

For some benchmarks here shown the graph and the table for TCC.



Effect of branching and Vectorization on processing speed of VLIW processor

S.NO.	BENCHMARK	SCALAR	VLIW	ARRAY BASED VLIW	PERFORMANCE INCREASES WITH COMPARE TO	
					SCALAR (%)	VECTOR (%)
1.	Ifthen	4112	3216	3103	24.53	3.51
2.	Fact2	63	55	10	88.33	81.81
3.	Fib	46	46	12	73.91	73.91
4.	Nested	22020	16041	3884	82.36	75.57
5.	Sqrt	3171	2950	3189	0.056	0.81
6.	Strcpy	21468	17461	15434	28.10	11.60
7.	eight	4344	4336	4013	7.61	7.44

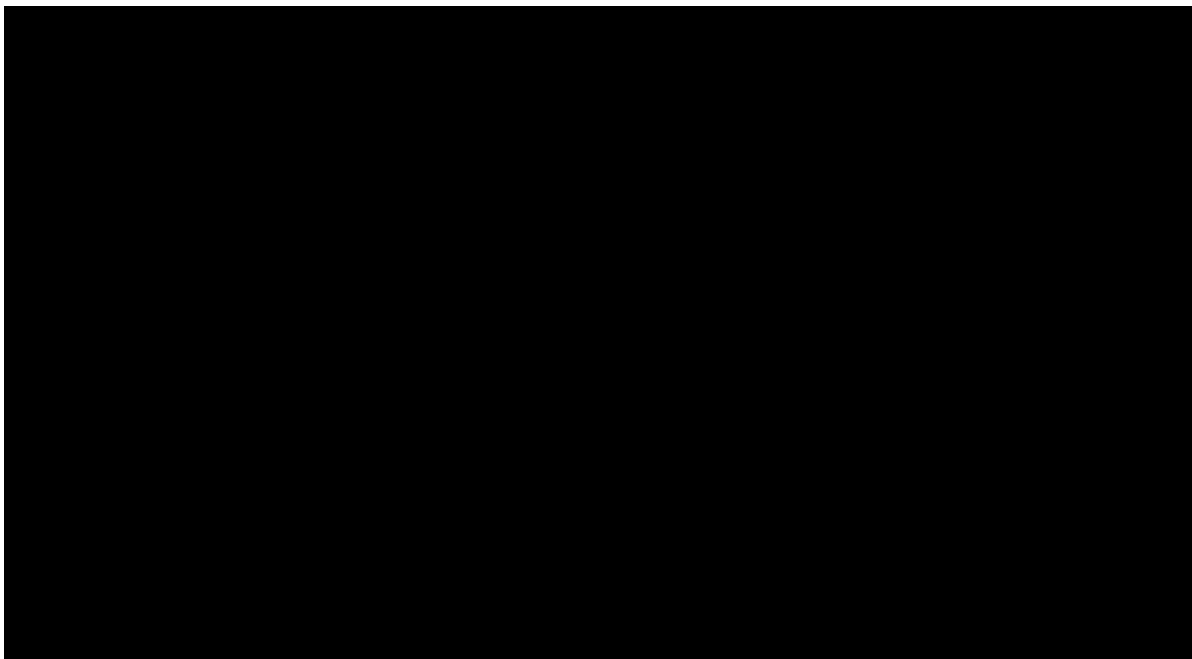
From the above table it is shown that the total compute cycles in three different processors are calculated by the TRIMARAN framework. It is clear that the performance is increased for almost all benchmarks, array based VLIW Processor with comparison to scalar and VLIW processor.

IPC Measurements

Now taking in consideration of IPC measures. The table below is the instruction per cycle count of the benchmarks. And the graph shows the variations of the IPC values for different benchmarks.

Effect of branching and Vectorization on processing speed of VLIW processor

S.NO.	BECNHMARK	SCALAR	VECTOR	ARRAY BASED VLIW
1.	Ifthen	1.03	1.63	0.87
2.	Fact2	1.17	1.64	3.10
3.	Fib	2.50	2.50	2.90
4.	Nested	2.04	1.57	1.52
5.	sqrt	2.60	2.77	3.00



BRANCHING Effect:

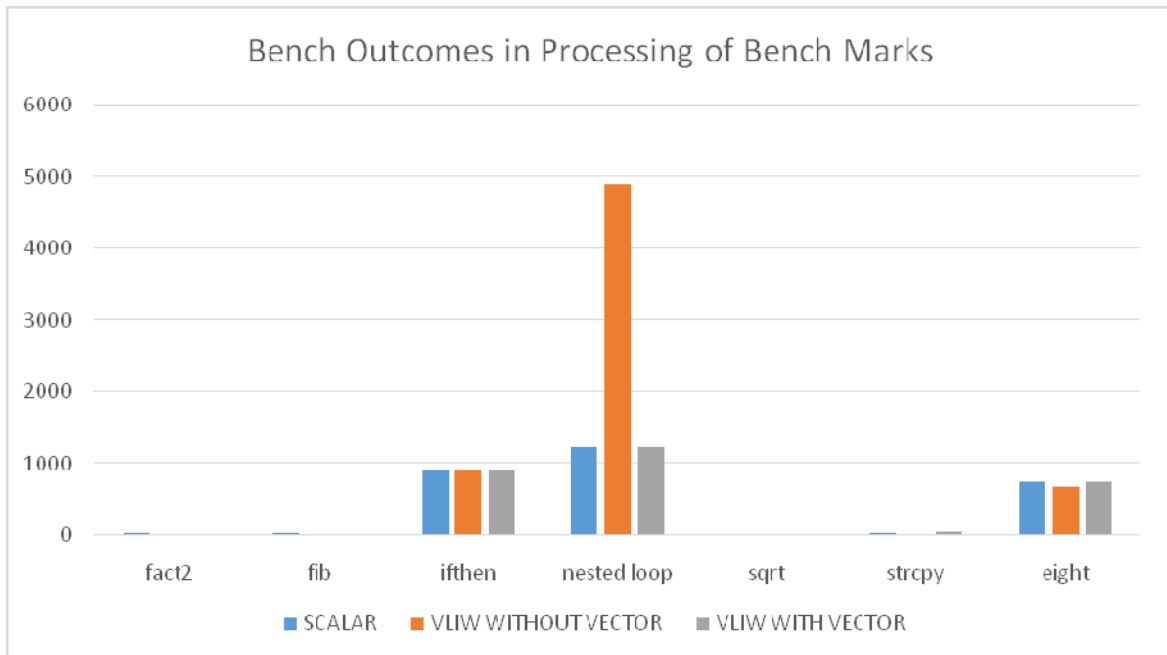
Branching is the problem in program execution, can delay the execution of a program.

It is beneficial to eliminate branches which are hard to predict. Branch prediction is a strategy in computer architecture design to eliminate the branches.

BRANCH OUTCOMES IN PROCESSING BENCH MARKS

S.NO.	BENCHMARK	SCALAR	VLIW WITHOUT VECTOR	VLIW WITH VECTOR
1.	Fact2	15	15	1
2	Fib	16	16	1
3.	Ifthen	901	902	901
4.	Nested loop	1214	4906	1214
5.	Sqrt	1	18	3
6.	Strcpy	23	19	29
7.	Eight	735	668	735

Effect of branching and Vectorization on processing speed of VLIW processor



The above Table shows the branch taken values in the program execution of some benchmarks. Branch taken causes the delay in program execution. Jump statements are the kind of branches that moves the address pointer from one location to another location. If one branch is taken then the instructions which are followed by that branch will be made useless and they will wait that when the pointer will come back to that instruction.

Chapter 6

Conclusion and future works

conclusion

Array based VLIW processor is feasible to be made and should be more efficient to execute the programs in less time by using two types of parallelism ILP and DLP. Array based VLIW processor have which takes advantages of both VLIW and Vector processor architecture.

However, the compiler is critical for achieving maximum benefit because complete benefit of this processor cannot be realized unless the compiler can schedule the instructions in parallel efficiently. So Compiler needs to more complex and should able to schedule the multiple instructions in an effective way. Branching problem is still remained as it is, not resolved like VLIW processor branch prediction mechanism.

Future works

This array based VLIW processor has a lot of possibilities of improvements in many areas like efficient branch prediction and less access time on-chip memory. It can be a very good and efficient architecture for multimedia programs.

1. Reference book:

[1] Kai Hwang (2001) “Advanced Computer Architecture”, edition first, Tata McGraw Hill publications Ltd. New Delhi.

[2] Kai Hwang and Naresh Jotwani (2010) “Advanced Computer Architecture”, edition second, Tata McGraw Hill publications Ltd. New Delhi.

2. Research Papers and Articles:

1. A. Bona, M.Samit, D.Sciutot, C.silvanog (2006), “energy estimation and optimization of embedded VLIW processor based on instruction clustering”, ACM, page: 886-891.
2. Mohammad Suaib, Abel Palaty, Kumar Sambhav Pandey(2011), “Architecture of SIMD Type Vector Processor”, International journal of computer applications.
3. Andrea Lodi, Mario Toma, Fabio Campi, Andrea Cappelli, Roberto Canegallo, and Roberto Guerrieri (2003), “A VLIW Processor With Reconfigurable Instruction Set for Embedded Applications”, IEEE.
4. Anju S. Pillai, Isha T.B.(2013), “Factors Causing Power Consumption in an Embedded Processor - A Study”, IJAIEM.
5. Christoforos E., Kozyr David A. pattersonakis (2003), “scalable vector processors for embedded systems”, IEEE.
6. Christos Kozyrakis, David Patterson “Overcoming the limitations of conventional vector processors”, Department of Electrical Engineering Stanford University.
7. Christoforos Kozyrakis, David Patterson (2002), “Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks”, IEEE.
8. Carlos Carvalho (2012), “The Gap between Processor and Memory Speeds”, ICCA.
9. Erkan Diken, Roel Jordans, Lech J´o´zwiak and Henk Corporaal (2014), “Construction and Exploitation of VLIW ASIPs with Multiple Vector-Widths”, 3rd Mediterranean Conference on Embedded Computing MECO.
10. Eric Spangle, Doug Carmean (2002), “Increasing Processor Performance By Implementing Deeper Pipelines”, IEEE.
11. Francisca Quintana, Roger Espasat, Mateo Valero (1998), “A Case for Merging the ILP and DLP Paradigms”, IEEE.
12. James E. Smith, Guindar singh sohi (1995), “Micro-architecture of superscalar processors”, IEEE, page: 1609-1623.
13. Kouros Gharachorloo, Anoop Gupta, and John Hennessy, “Two Techniques to Enhance the Performance of Memory Consistency Models”.

14. Khoi-Nguyen Le-Huu, Diem Ho, Anh-Vu Dinh-Duc, Thanh T. Vu (2014), "Towards a RISC Instruction Set Architecture for the 32-bit VLIW DSP Processor Core", University of Information Technology – VNUHCM Ho Chi Minh City, Vietnam.
15. Mikhail Smelyansky, Gary S.Tyson and Edward S.Davidson (2000)," Register Queues: A new hardware/software approach to efficient software pipelining", IEEE, page: 3-12.
16. Mostafa I. Soliman, Al-Madinah Al-Munawwarah 2898, Saudi Arabia (2013), "A VLIW Architecture for Executing Multi-Scalar/Vector Instructions on Unified Datapath", IEEE.
17. M.priyanka, K.Niranjan Reddy (2014), "VLIW-Based Processor for Executing Multi-Scalar/Vector Instructions", IJIREC.
18. Roger Espasa, Mateo Valero(1997), "Simultaneous Multithreaded Vector Architecture: Merging ILP and DLP for High Performance", IEEE.
19. Rekha Halkatti, Veeresh Pujari(2014), "FPGA BASED 128-BIT CUSTOMISED VLIW PROCESSOR FOR EXECUTING DUAL SCALAR/VECTOR INSTRUCTIONS", IJRET.
20. Deepak Kumar, Ranjan Kumar Behera, K. S. Pandey (2013), "Concept of a Supervector Processor: A Vector Approach to Superscalar Processor, Design and Performance Analysis", IJER.
21. Stephen Hines, Joshua Green, Gary Tyson and David Whalley (2005) "Improving program efficiency by packing instructions into registers" IEEE.
22. Vikram Seth, Min Zhao, Jiang Hu (2004), "Exploiting level sensitive latches in wire pipelining" IEEE, page: 283-290.
23. Venkata Ganapathi Puppala (2014), "A VLIW-Vector Co-processor Design for Accelerating Basic Linear Algebraic Operations in Open-CV", IEEE.
24. Yuan Xie, Wayne Wolf Lekatsas (2001), "A code decompression architecture for VLIW processors", IEEE, page: 66-75.
25. Yangzhao Yang, Naijie Gu, Kaixin Ren, Bingqing Hu (2014), "An Approach to Enhance Loop Performance for Multicluster VLIW DSP Processor", ARCS.
26. Yonghua Hu, Shuming Chen, Jie Huang (2012), "Preprocessing scheme of intelligent assembly for a high performance VLIW DSP", IEEE.

3. WebPages and Websites:

1. <http://trimaran.org/documentation.shtml>
2. <http://www.cs.nmsu.edu/~rvinyard/itanium/branching.htm>
3. <http://www.technologyreview.com/view/421186/why-cpus-arent-getting-any-faster/>
4. <http://cacm.acm.org/magazines/2011/5/107702-the-future-of-microprocessors/fulltext>