Major Project - I

SIMULATION RESULT VISUALIZER FOR TEJAS

Under the guidance of:
Prof. Smruti Sarangi

Shantanu Agarwal
2016MCS2661

Motivation for Simulation Result Visualizer

- Having a visual display of results of the simulation is quite a good feature for any architecture simulator.
- What Tejas was generating previously?
- Tejas was not able to show how these values changes during simulation.
- Now with current visualization modifications, user can see interactive dynamic graphs of how these values are changing with time(Number of cycles).

Timeline of Work done

Objective not met.

Create a tracing framework for Android.

- July(Begin)-August(End).

Visualizer built and working properly.

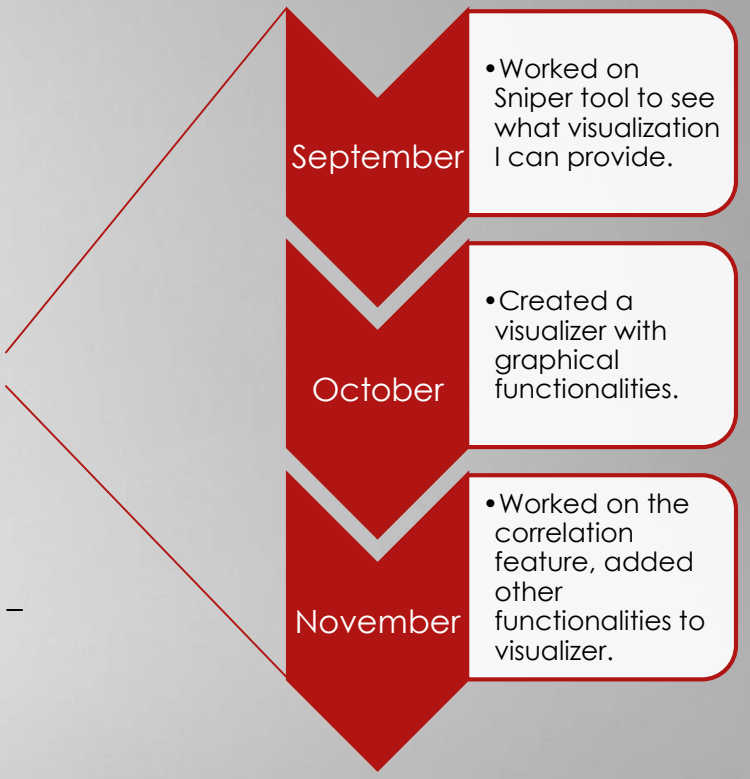
Create a visualizer tool for Tejas.

- September(Begin) – November(End).

Studied and worked on different hazards and stalls and added the data hazard metric in Tejas.

Measuring hazard penalties in pipelines using Tejas.

- December(Begin) – January(Begin).



Work Done

Following things are done for reaching the results:

- Studied the architecture, source code and code flow of Tejas.
- Modified the Tejas source code to generate a new dump file containing values of various parameters during simulation.
- Designed, and implemented the Visualizer.
- Added components for calculation of correlation, average, minimum value and maximum value for single and multiple features.
- Designed a module which displays and saves graphs of various features(single and multiple benchmarks) for comparison.
- Introduced a new metric “Data Hazard Factor(DHF)” in Tejas.

What's New?

There exist other visualization tools for the similar work. So what is new in this?

Features that are innovative in this are:

- One can easily compare multiple benchmarks and multiple features of benchmarks to see if there is some pattern followed.
- Calculates and display the correlation vector of a feature and rate of change of that feature.
- Ability to export all the calculated data to an external file.
- User can dynamically provide the epoch(gaps at which rate of change is to be calculated) value.
- Added one more feature to show the hazard factors of different potential hazards in pipeline.

List of files modified in Tejas

Filename	Modifications
DecodeLogic.java	Added a function to calculate energy spent in decode logic.
ExecutionLogic.java	Added a function to calculate energy spent in execution logic.
InstrutionWindow.java	Added a function to calculate energy spent in instruction window.
RegisterFile.java	Added a function to calculate energy spent in register file.
RenameLogic.java	Added a function to calculate energy spent in rename logic.
RenameTable.java	Added a function to calculate energy spent in rename table.
ReorderBuffer.java	Added a function to calculate energy spent in reorder buffer.
OutOfOrderExecutionEngine.java	Added 11 functions that receives the energy from different classes.
MultiIsseInorderExecutionEngine.java	Added 11 functions that receives the energy from different classes.
ExecUnitIn_MII.java	Added a function to calculate energy spent in execution unit.
WriteBackUnitIn_MII.java	Added a function to calculate energy spent in write back unit and a function to set the number of instructions for their types.

List of files modified in Tejas(Contd.)

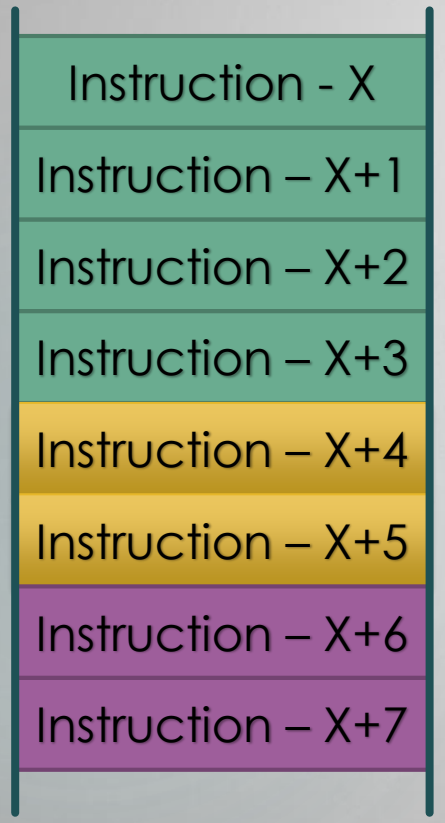
Filename	Modifications
DecodeUnit_MII.java	Added a function to calculate energy spent in decode unit.
ExecutionEngine.java	Added abstract methods to get energy from classes.
TLB.java	Added a function to calculate energy spent in TLB.
LSQ.java	Added a function to calculate energy spent in LSQ.
Cache.java	Added a function to calculate energy spent in Cache.
ArchitecturalComponent.java	Added a function to get the number of instructions executed in every category(Memory, branch, synchronization, compute). Added functions to get energy from iCache, iTLB, dCache, dTLB., and some other internal components.
OperandType.java	Changed the class(gave integer representation to operands, created enumeration).
Core.java	Added functions to get energy and count of instructions executed from internal components.
RunnableThread.java	Also added the file handling and energy calculation code to generate dump file from Tejas during run time. Created functions to interact with the JFreeChart charting library used for graphs.

Data Hazard and Stalls

A data hazard is created whenever there is a dependence between two instructions, and if the later instruction if scheduled for execution, it causes a potential race condition. Thus we need to stall the pipeline for avoiding these hazards.

So how are we calculating data hazard stalls in out-of-order pipeline?

Data Hazard and Stalls(Contd.)

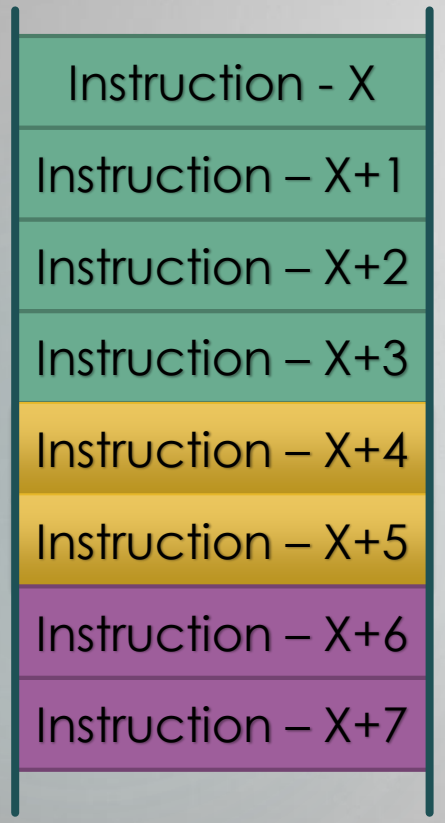


Ready instruction without any hazard → Instruction - Y

Instruction suffering from data hazard → Instruction - Y

Instruction suffering from structural hazard → Instruction - Y

Data Hazard and Stalls(Contd.)



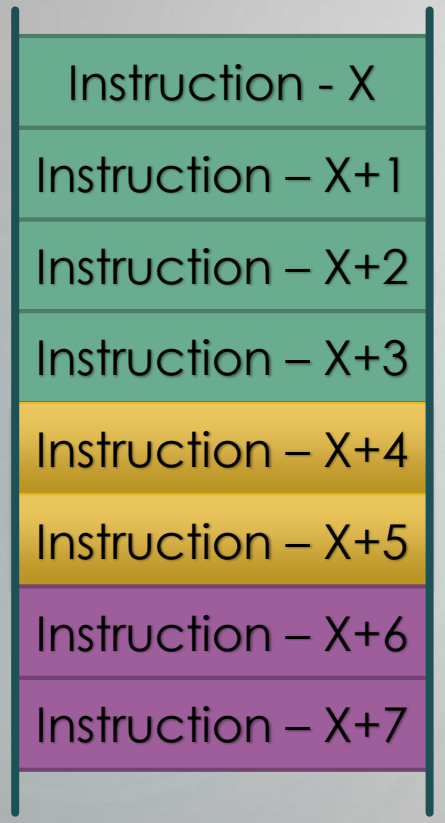
Issue Width = 4

Ready instruction without any hazard → Instruction - Y

Instruction suffering from data hazard → Instruction - Y

Instruction suffering from structural hazard → Instruction - Y

Data Hazard and Stalls(Contd.)



Issue Width = 4

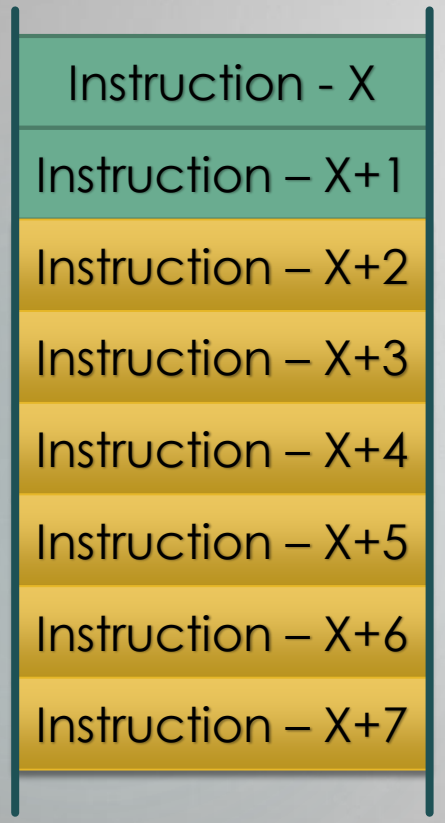
Data Stalls = 0

Ready instruction without any hazard → Instruction - Y

Instruction suffering from data hazard → Instruction - Y

Instruction suffering from structural hazard → Instruction - Y

Data Hazard and Stalls(Contd.)



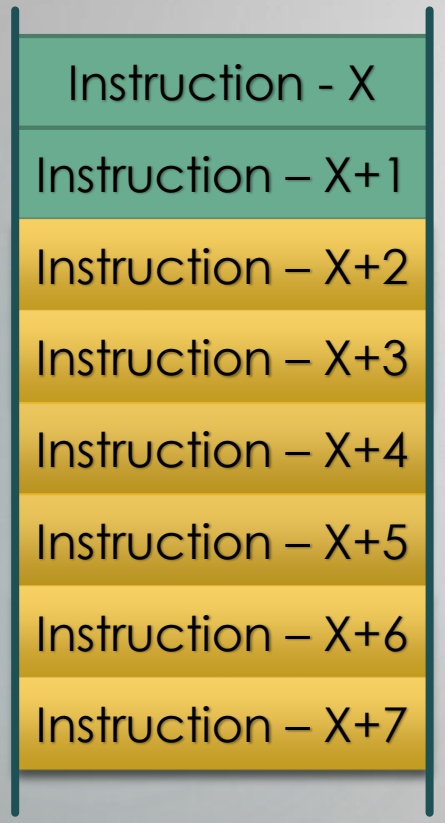
Issue Width = 4

Ready instruction without any hazard → Instruction - Y

Instruction suffering from data hazard → Instruction - Y

Instruction suffering from structural hazard → Instruction - Y

Data Hazard and Stalls(Contd.)



Issue Width = 4

Data Stalls = 2

Ready instruction without any hazard → Instruction - Y

Instruction suffering from data hazard → Instruction - Y

Instruction suffering from structural hazard → Instruction - Y

Data Hazard and Stalls

If the number of instructions issued are less than issue width and instructions without data hazard are more than issue width then no data hazard is there and if instructions without data hazard are less than issue width then some amount of data hazard is there.

Data Hazard Factor

- We define **Data Hazard Factor(DHF)** as a metric which implies how often we need to stall the pipeline to avoid data hazards.
- DHF lies in range [0,1).
- Higher DHF implies lower IPC.

Data hazard factor for out of order pipeline is defined as:

$$DHF = \frac{Total\ data\ stalls}{IssueWidth \times Number\ of\ Cycles}$$

Where,

DHF is data hazard factor,

Total data stalls are the stalls that are discovered during the benchmark execution,

IssueWidth is the maximum number of instructions that can be issued for execution in single cycle,

Number of cycles are the total cycles taken to complete the execution.

Why this expression?

Data hazard factor for out of order pipeline is defined as:

$$DHF = \frac{Total\ data\ stalls}{IssueWidth \times Number\ of\ Cycles}$$

Using hit and trial method some other expressions were used as well like:

$$DHF = \frac{Total\ data\ stalls}{Number\ of\ Instructions},$$

$$DHF = \frac{Total\ data\ stalls}{IssueWidth \times Number\ of\ Instructions}.$$

We ran the object file of a benchmark instead of the executable with the following results:

Data Stall	:	1254920
Total cycles taken	:	574087
Number of instructions	:	215297

So there exist a set of instructions where the above two expressions will fail to give satisfactory results.

Experiments

As an experiment to verify the results for data stall factor, some common benchmarks with varying instruction issue width of pipeline are used to test the resulting data stall factor.

Benchmark Name	Benchmark Description
hello.asm	X86 assembly program to print Hello, World! on screen.
countlakh.asm	A counter program in assembly x86 to count from 1 to 1,00,000.
print1000.asm	An assembly x86 program to print first 1000 natural numbers.
countmillion.asm	A counter program in assembly x86 to count from 1 to 1 million.
dependent.asm	A x86 assembly program where every instruction depends on the previous instruction.

Results

The results of running the benchmarks mentioned previously are:

Benchmark Name	Issue Width	Data Stalls	Number of Cycles	Data Hazard Factor
hello.asm	1 2 3 4	0 0 0 0	0 0 0 0	Invalid Invalid Invalid Invalid
countlakh.asm	1 2 3 4	2 5 11 19	399069 200156 200145 200052	$5*10^{-6}$ $1.2*10^{-5}$ $1.8*10^{-5}$ $2.3*10^{-5}$
print1000.asm	1 2 3 4	115 207 6549 26936	120827 70167 57123 57177	$9.5*10^{-4}$ $1.4*10^{-3}$ $3.8*10^{-2}$ 0.11
countmillion.asm	1 2 3 4	2 5 11 19	3998959 2000085 2000043 2000037	$5*10^{-7}$ $1.2*10^{-6}$ $1.8*10^{-6}$ $2.3*10^{-6}$
Dependent.asm	4	129	694	0.046

Conclusion

- We can’t predict whether outliers exist for the presented expression.
- Experimental results conform with the presented expression.
- The Data Hazard Factor metric is able to show the severity of data hazards in the input benchmark.

Future Work

- Moving the Visualizer online so that anyone can access it from anywhere.
- Adding structural and control hazard factors in out of order pipeline in Tejas.
- Working on the hazard factors to get more accurate metric.

Resources and links

- Used JFreeChart charting library for generating graphs.
- A complete modified source code of Tejas is available at:
  www.Shantanu.pro/mtp/tejas.tar.gz
- A manual for how to do these changes from scratch on tejas is available at:
  www.Shantanu.pro/mtp/TejasChanges.odt
- A video for single run on simulation a basic hello_world.c program is available at:
  www.Shantanu.pro/mtp/tejasVideo.ogv

Thank you!