Design space exploration for optimization of the PISA architecture for mpeg2 decoder benchmark using Wattch simulator.

Submitted by:

Ananda Shankar Basu
&
Sumit Ahuja

December 10, 2003

Introduction

The goal of this project is to find an optimal configuration of the PISA architecture that will give the best performance when used for the mpeg2 benchmark. For measurements of the performance and cost function, we have used the simplescalar toolset Wattch simulator.
The strategy which we are undertaking in this work is to explore the design space in an exhaustive manner. We have a total simulation time of 3 hours, and each simulation of the mpeg2 decoder takes approximately 3 minutes, so we can run approximately 60 simulations. Hence we have to properly select the design space so that we can get a configuration giving the optimal performance.
Proposing architecture requires lot of parameters to take into design. So first of all we analyze the nature of the benchmark code and depending on it, we started from an intuitive design space to get optimal performance. In the selective design space, we have applied exhaustive search method for changing these parameters.
The results of the simulation were analyzed and based on this analysis and also considering the tradeoffs, we proposed an architecture for efficient execution of the mpeg2 decoder algorithm.

Parameter Selection

First of all we need to fix the parameters which we will measure to optimize the design. As we are considering the performance, we select simulation delay as a measure of performance. This will be minimized with respect to total energy consumption for the proposed design. Finally we need to reach a trade off that optimizes both.

Definition of Cost-Function

The mpeg2 decoding algorithm is supposed to process bulks of encoded image streams and needs good amount of processing. So measuring the energy consumption will be a good choice when we optimize the performance.
Hence we will take the Energy Delay Product (EDP) as our cost function to analyze the energy and delay tradeoffs in order to provide an analysis of the given architecture. The EDP is defined as:
EDP = Energy * Delay
Where:
– Energy is obtained from Wattch simulation result: total_power_cycle_cc3.
– Delay is obtained from Wattch simulation result: sim_cycle.

This cost function will be computed for each configuration in our design space.

The Wattch output result “total_power_cycle_cc3” gives a measurement equivalent to the total power consumed for non-ideal conditional clocking.
A Delay versus Energy plot is also created to aid the designer in making an architecture selection to fit the performance constraints.

Wattch Configuration parameter selection

In order to optimize the performance of the mpeg2 decoder we took a look on the source code and tried to find out the nature of the algorithm used and also the resources this algorithm mainly requires. We also did some initial trial simulations to determine the parameters that will significantly affect the performance as well as the cost function. These parameters will be changed to explore eh design space.
After these manual simulations we figured out what are the parameters which have significant effect on the performance of the mpeg2 decoder algorithm are as follows:

1. L1 Data Cache parameters
2. L1 Instruction Cache parameters
3. L2 Unified cache parameters
4. Register Update Unit (RUU) size
5. Number of floating point ALU

Design Space Definition

Based on our initial analysis of the algorithm, we select the following ranges for the parameters:

1. DL1:128{32,64}:4:L
2. IL1:{128,256}:32:1:L
3. UL2:{1024,2048}:{64,128}:4:L
3. RUU:{32,64}
4. FPALU:{1,4}

With the above variations we will have 64 different cases to explore.

Design Space Exploration

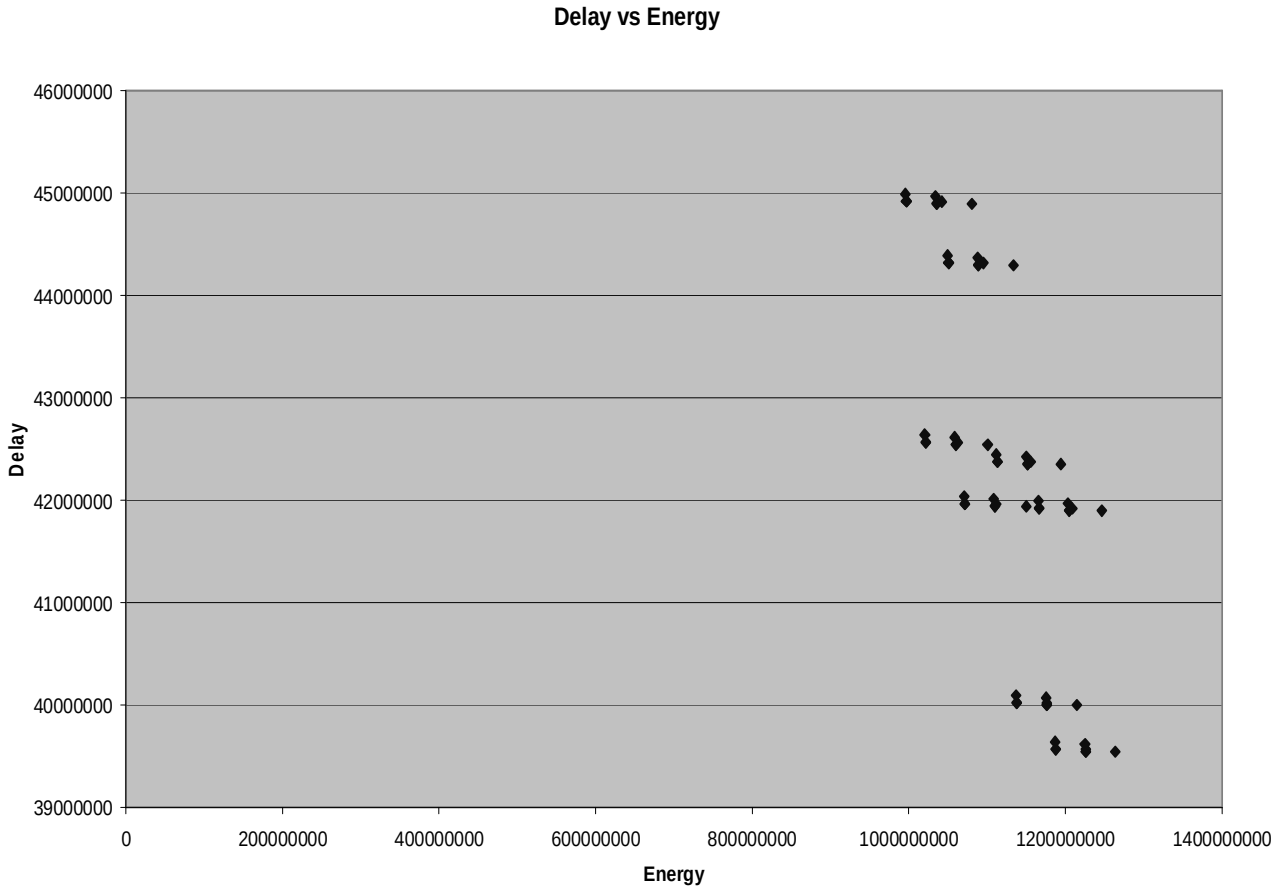
We simulated all the combinations of the parameters(64 different cases) and obtained the values for the delay and energy consumption. The whole process was automated through a script.
We are running an exhaustive search algorithm to cover the different options.

Conclusion

The result of simulation is analyzed through the following graphs.

Delay versus Energy graph:

Here we plot the simulation delay with respect to the total energy consumed, given by the Wattch outputs “sim_cycle” and “total_power_cycle_cc3”



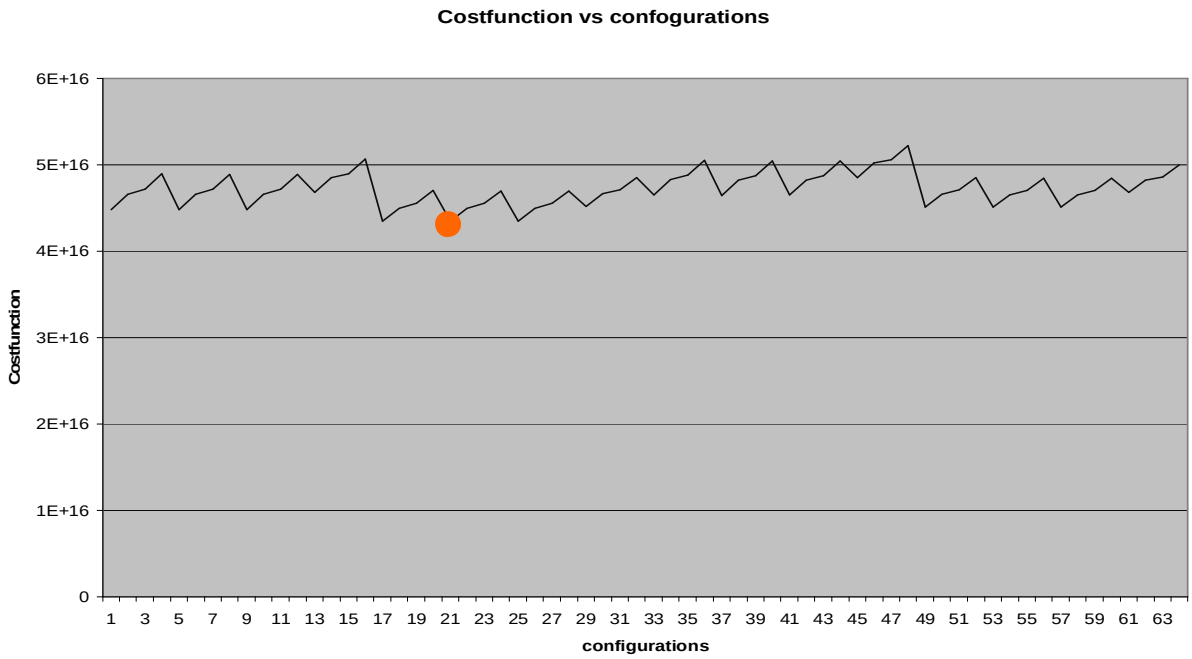
From the plot, we see four cluster of points, with the middle two clusters almost overlapping. The behavior is describes as follows.
When we increase the number of FP-ALU’s , we come from the first row to the second row, i.e. delay decreases, but energy consumption increases.
When the RUU size is increased, the performance point jumps to the middle rows indicating a good increase in performance. Again, keeping the RUU size constant, if number of FP-ALU is increased, the points change between the rows in the middle cluster.

The operating point jumps to the lower cluster due to increase of LI data cache size. This signifies a huge improvement in performance, but accompanied with a good amount of power increment also. Here again the operating point shifts between the middle rows due to variation of FP-ALU number, and between the middle and lower roes due to RUU size variation.

The above behavior of the operating points are quite clear and justified as per the change of parameters. So we can expect the optimal point somewhere in the left half of the middle cluster of points. That will be evident once we analyze the cost function curve.

Cost Function Curve:

The cost-function is the product of delay and corresponding energy for the configuration under simulation.
The cost-function plot is shown as below. It shows the variation of the cost-function with the configuration number.



We find that the minimum value of the cost-function corresponds to the configuration number 21. However, configurations 17 and 25 also give a very close cost function value.

Optimal configuration

From the analysis, we get that the configuration that gives the optimal result, i.e. optimal tradeoff between performance and total energy consumption is as follows :

dl1:128:32:4:l
il1:256:32:1:l
ul2:1024:128:4:l
ruu size 32
fp-alu 1

Observations

We found that splitting the L2 cache into separate data and instruction cache resulted in no significant improvement.
Also we changed the replacement policy from LRU to FIFO, but that again produced no satisfactory outputs.
Also we tried varying the number of integer ALU's but of no significant result.
Hence we did not vary these parameters in our design space exploration.

Analysis

We see that the number of cache sets in LI data cache is not varied, because increasing this value gives an increase in energy consumption without remarkable improvement in performance. Also increasing the LI cache size implies increasing cost.
But we did vary the block size in this case. We found that increasing the block size gave improvement in the performance. Since the mpeg2 computes on streams of data, there are lot of looping operations over the data, hence we can infer that if more amount of related data is present in the same block, performance enhances, thus increasing the block size gave significant performance improvement.
Also by changing the LI instruction cache number of sets, we get good results.

Also the performance is dependent on the RUU (register update unit)size.
We find that increasing the RUU size improved the performance significantly. This is quite true as increasing RUU implies increasing the number of ROB's, increasing the number of issues and increasing the degree of out of order execution. And in the mpeg2 decoder algorithm, the main complex part does a lots of looping and there is very less dependency of the data, i.e. very less dependency on the values of the intermediate data, so a data once computed is seldom used in a future instruction. Hence increasing the out of order simulation increased the performance significantly.

But another important observation is that the increase of floating point ALU doesn't enhances the performance significantly, and this is quite clear as there is not a good amount of floating point operations in the mpeg2 decoder.

Comments

We feel that some more optimizations could be done from the style of writing the code.
For example, multiplication by powers of 2 can be replaced by shift operations, and some symbol tables can be used for reading complex constant values.
The compiler can also do some optimization (although we used optimization level 2 for the cross compiler).

Appendix:

Code of Scripts used for simulations.

runReg.sh

```
#!/bin/tcsh -f

set count = 1
set WATTCHBIN  = /opt/wattch_alari/bin/wattch
set WATTCHARGS = "-config ./temp.config"
set TARGETBIN  = "../bin/mpeg2decode"
set TARGETOPT  = "-b ./encoded.m2v -r -f -o0 ./tmp%d"

#if (-e ./report.txt) then
#  mv ./report.txt ./report.txt.last
#endif

#echo "       Energy Versus Delay report" > ./report.txt
#echo "       **************************" >> ./report.txt
#echo " "

foreach dl1_block_size (32 64)
  foreach il1_no_of_sets (128 256)
    foreach ul2_no_of_sets (1024 2048)
      foreach ul2_block_size (64 128)
        foreach ruu_size (32 64)
          foreach fp_alu_no (1 4)
            echo "Running configuration no $count....."
            # Generate the simulator options
            set l1_data_cache_opt  = "-cache:dl1 dl1:128:"$dl1_block_size":4:l"
            set l1_instr_cache_opt = "-cache:il1 il1:"$il1_no_of_sets":32:1:l"
           set l2_unif_cache_opt  = "-cache:dl2 ul2:"$ul2_no_of_sets":"$ul2_block_size":4:l
cache:il2 dl2"
            set ruu_size_opt = "-ruu:size "$ruu_size
            set fp_alu_opt  = "-res:fpalu "$fp_alu_no

            echo "-redir:sim ./wattch.log" > temp.config
            echo $l1_data_cache_opt  >> temp.config
            echo $l1_instr_cache_opt >> temp.config
            echo $l2_unif_cache_opt  >> temp.config
            echo $ruu_size_opt       >> temp.config
            echo $fp_alu_opt         >> temp.config
            endif
            @ count = $count + 1;
            # Run wattch with current option
```

```
        $WATTCHBIN $WATTCHARGS $TARGETBIN $TARGETOPT  >& /dev/null
        # Extract result parameters
        ./makeReport.sh ./wattch.log ./report.txt
        # Remove config file
        \rm temp.config tmp*.U tmp*.V tmp*.Y
      end
    end
  end
 end
end
echo $count
```

makeReport.sh

```
#!/bin/sh -f
if [ ! $# == 2 ]; then
   echo " Usage <report> input output"
   exit
fi
#usage <report> input output

#grep -w "sim_CPI" $1 >> $2
grep -w "total_power" $1 >> $2
grep -w "total_power_cycle_cc3" $1 >> $2
grep -w "sim_cycle" $1 >> $2
echo "" >> $2
```