Karlsruher Institut für Technologie

Fakultät für Informatik
Institut für Technische Informatik
Chair for Embedded Systems

Lab: Low Power Design and Embedded Systems

Software for Low Power Consumption

Jeferson González-Goméz
jeferson.gonzalez@kit.edu
Jorge Castro-Godínez
jorge.castro-godinez@kit.edu

1 Goal

To experiment and analyze the effect of loop optimization techniques in the power consumption and performance.

2 Description

In this part of the Lab, you will test and analyze the effect of loop optimization techniques in the power consumption and performance of software using the SimpleScalar/Wattch microarchitectural simulator. You will analyze the effects in a two-dimensions array and in three applications: matrix multiplication, Fast Fourier Transform (FFT) and Adaptive Differential Pulse Code Modulation (ADPCM).

SimpleScalar is a simulator used to perform design-space explorations of superscalar architectures. Among the parameters that can be explored are the cache memory configuration and the configuration of branch predictor. Wattch is a platform based in SimpleScalar that allows the exploration and analysis of power consumption for a particular software.

3 Architecture

The default architecture configuration of SimpleScalar will be used. In Table 1 the configuration for the cache memory is presented. This can be configured in the Wattch simulator as:

```
sim-outorder -cache:dl1 dl1:256:32:1:l -cache:il1 il1:256:32:1:l \
             -cache:il2 dl2 -cache:dl2 dl2:4096:32:1:l <binary>
```

Table 1: Memory configuration.

	L1 Instr.	L1 Data	L2 Unified
Size (KB)	8	8	128
Number of sets	256	256	4096
Associativity	1	1	1
Line size (B)	32	32	32

For compiling the source code, you will use `sslittle-na-sstrix-gcc`, a modified gcc-based compiler.

4 2D Arrays

You will implement a small C program that fills a two-dimensions array through two cycles, using a cycle per dimension, i.e. there is an outer-loop and a inner-loop. The program must fill the array with data associated to the loop index. For instance, if `i` is the counter for the outer-loop and `j` for the inner-loop, the data to assign in the array is the addition of both indexes: `array[i][j] = i+j`.

The analysis of this simple program will be made from two points of view: a) the effect of loop transformations, and b) the effect of the compiler optimization, both in the performance and the power consumption.

4.1 Loop transformations

It is required to perform the compilation and analysis for 4 different array dimensions: 16x16, 32x32, 64x64 and 128x128 Bytes. The reference code is:

```
1 for (i=0; i<N; i++)
2   for (j=0; j<N; j++)
3     array[i][j] = i+j;
```

where `N` takes the values 16, 32, 64 and 128.

Loop transformations: The following is a list of possible transformations to be introduced. These transformations can be implemented and analyzed independently or combined for each array size.

- Loop permutation.
- Loop unrolling.
- Loop tilling.
- Loop fision.

Measurements and analysis: For each transformation tested and each array size, the following data from the simulator should be measured and analyzed:

- Power consumed by the software.
- Number of instructions executed and execution time.
- Cache performance.

4.2 Compiler optimization levels

Different compiler optimization levels can be used when compiling a program. Use a 32x32 array dimension and the compiler options O0, O1, O2 and O3, for the reference code. For each optimization level, you should measure and analyze the previously mentioned points.

5 Applications

You will modify three applications, looking for make them more efficient. For this purpose, use the loop transformations and compiler optimization already tested. You are free to determine where to apply transformations in the code, but take in count that the loop transformations should not alter the results produced by the application. The code for these applications, matrix multiplication [1], FFT [1] and ADPCM [2], is provided.

You have to present and analyze the final implementation with the best performance achieved. However, you can present and analyze intermediate implementations and their effects. Consider as part of your analysis:

- Power consumed by the software.
- Amount of instructions executed and execution time.
- Cache performance.

References

[1] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. *LegUp: high-level synthesis for FPGA-based processor/accelerator systems*. In Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '11). ACM, New York, NY, USA, 33-36.

[2] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda and Hiroaki Takada. 2009. *Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis*. Journal of Information Processing, Vol. 17, pp.242-254.