1. (40 points)

**Department of Computer Science National Tsing Hua University EECS403000 Computer Architecture** Spring 2024, Homework 1

Due date: 2024/03/21

Install and use AndeSight™ for RISC-V program development.

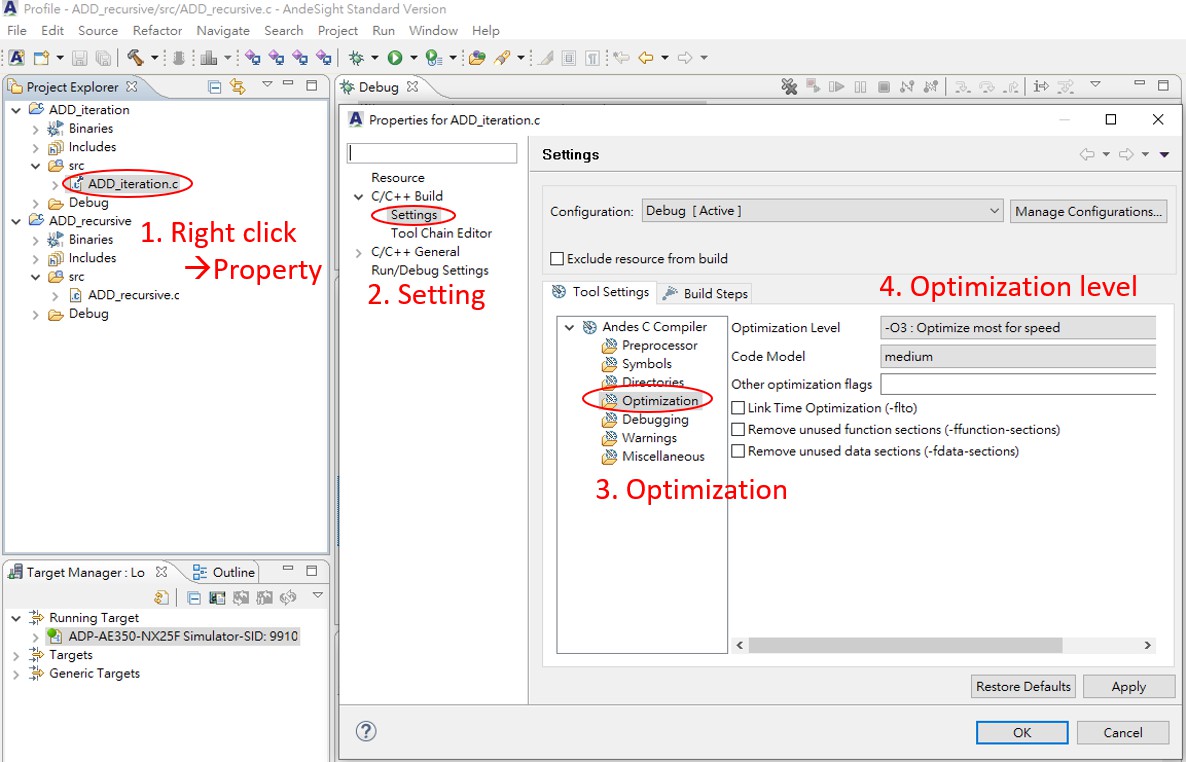
* 1. See AndeSight\_STD\_v5.3\_Installation\_Guide.pdf for the installation guide.
  2. Create a new Andes C Project
     1. Click on File  New  Project  C/C++, and select “Andes C Project”.
     2. Create a project with the project name “fast\_power\_recur”.
     3. From Chip Profile, select chip profile “AE350”  “ADP-AE350-NX25F”.
     4. From Project Type, select project type “Andes Executable”  “Hello World ANSI C Project”.
     5. From Toolchains, select the “nds64le-elf-mculib-v5d”.
     6. Other configurations are left as default.
  3. Replace fast\_power\_recur.c in the project with the one we provided.
  4. To build the project, click on the expanding arrow (a small triangle ) beside “Build”  in the toolbar

 “1 Debug”  for project “fast\_power\_recur” in the toolbar.

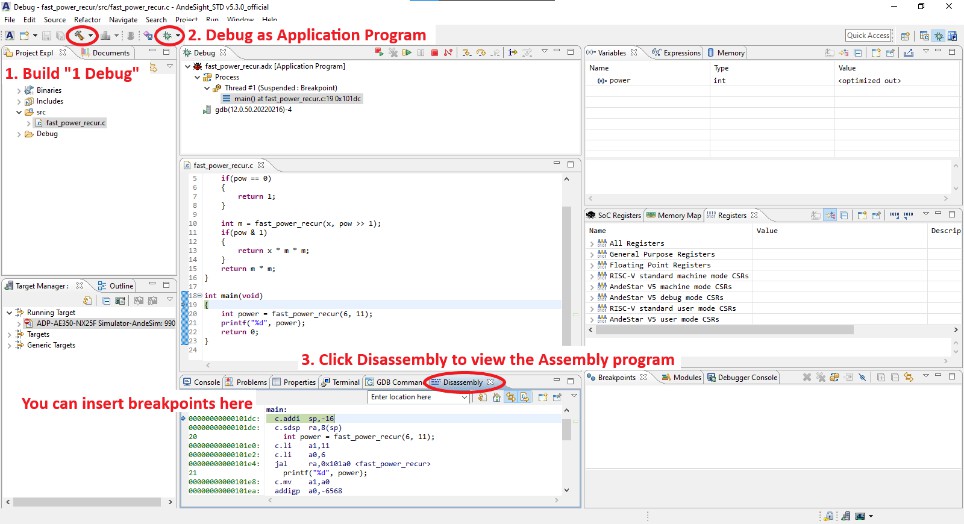
* 1. To execute the program, press “Debug”  in the toolbar  “1 Application Program” and press “Resume”  in the debug window.

You can follow the same steps for other program codes.

To select (or check) the optimization setting, follow the figure below.



To inspect the assembly code of the program, follow the figure below.



In this exercise, we will experiment with the naïve and fast power computation in two different implementations (iterative and recursive) with AndeSight™. There are four source codes, namely naive\_power\_iter.c, naive\_power\_recur.c, fast\_power\_iter.c, and fast\_power\_recur.c. The optimization level -Og will be used by default in the following questions unless stated otherwise.

1. (10 points) ***Effects of algorithms on performance***

Press “Profile”  in the toolbar and select Profile as “Application Program”. Press the button “Resume”  in the debug window, record the CycC and InsC for the four functions listed in the table below and complete the table. Based on the characteristics of the programs, briefly compare and explain the differences between the naïve and fast power algorithms in their profiles.

naïve algorithm 是一項一項慢慢乘起來的，fast power algorithm 有點像是一組一組乘起來。

以8次方為例：

naïve algorithm: (((((((x \* x) \* x) \* x) \* x) \* x) \* x) \* x) => 共做7次乘法

fast power algorithm: ((x2)2)2 => 共做3次乘法

=> fast power algorithm的乘法運算次數約為naïve algorithm乘法運算次數取log2

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Source | CycC | InsC |
| naive\_power\_iter() | naive\_power\_iter.c | 89 | 50 |
| naive\_power\_recur() | naive\_power\_recur.c | 218 | 135 |
| fast\_power\_iter() | fast\_power\_iter.c | 42 | 27 |
| fast\_power\_recur() | fast\_power\_recur.c | 118 | 73 |

1. (10 points) ***Effects of programming on performance***

From the table above and based on the characteristics of the programs, briefly compare and explain the differences between the iterative and recursive implementations of the fast power algorithm in their profiles. Suppose that they are executed in a processor with a clock rate of 3 GHz, what are the average CPI and CPU execution time for the fast\_power\_iter() and fast\_power\_recur() functions?

|  |  |  |
| --- | --- | --- |
| Function | Average CPI | Average Execution Time |
| fast\_power\_iter() | 1.56 | 5.39 ps |
| fast\_power\_recur() | 1.62 | 5.18 ps |

1. (10 points) ***Effects of compilers on performance***

Compile fast\_power\_iter.c and fast\_power\_recur.c with two optimizations levels, -O0 and -O1. Record the CycC and InsC and compute their corresponding CPI for the two different optimization levels. Furthermore, briefly compare and explain the differences in their profiles.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function | Optimization level -O0 | | | Optimization level -O1 | | |
| CycC | InsC | CPI | CycC | InsC | CPI |
| fast\_power\_iter() | 191 | 87 | 2.20 | 42 | 27 | 1.56 |
| fast\_power\_recur() | 267 | 167 | 1.60 | 118 | 73 | 1.62 |

1. (10 points) ***Compilers versus hardware implementations***

If we want to run the -O0 codes compiled in (c) on a faster processor to achieve the same speedup as running the -O1 codes on the original processor in fast\_power\_iter() and fast\_power\_recur(), what will the clock rates of the faster processor be for fast\_power\_iter.c and fast\_power\_recur.c respectively?

Denote clock rates as R

fast\_power\_iter.c: Rfaster processor = Roriginal processor \* (191 / 42)

fast\_power\_recur.c: Rfaster processor = Roriginal processor \* (267 / 118)

1. (25 points) ***Benchmarking***

Below is a comparison between three mobile phones and their processors.

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Samsung Galaxy S24 Ultra | Apple iPhone 15 Pro Max | Google Pixel 8 Pro |
| SoC | Snapdragon 8 Gen 3 | Apple A17 Pro | Google Tensor G3 |
| Cores | 8 (1+3+2+2) | 6 (2+4) | 9 (1+4+4) |
| PDF renderer | 227.4 Mpixels/sec | 178.5 Mpixels/sec | 153 Mpixels/sec |
| HDR | 238.2 Mpixels/sec | 232.4 Mpixels/sec | 136.9 Mpixels/sec |
| Background blur | 26.7 images/sec | 27.9 images/sec | 15 images/sec |
| Photo processing | 64.9 images/sec | 79.1 images/sec | 47 images/sec |
| Ray tracing | 7.38 Mpixels/sec | 7.58 Mpixels/sec | 4.52 Mpixels/sec |

The information above is provided by <https://nanoreview.net/en/soc-list/rating>.

1. (5 points) Follow the link <https://nanoreview.net/en/soc-list/rating> and fill in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Core name | Peak frequency of the most performant block of cores (MHz) | | | | | |
| Snapdragon 8  Gen 3 | One core | Three cores | | Two cores | | Two cores |
| Cortex-X4 | Cortex-A720 | | Cortex-A720 | | Cortex-A520 |
| 3.3 GHz | 3.15 GHz | | 2.96 GHz | | 2.26 GHz |
| Apple A17 Pro | Two cores | | | Four cores | | |
| Everest | | | Sawtooth | | |
| 3.78 GHz | | | 2.11 GHz | | |
| Google Tensor G3 Pro | One core | | Four cores | | Four cores | |
| Cortex-X3 | | Cortex-A715 | | Cortex-A510 | |
| 2.91 GHz | | 2.37 GHz | | 1.7 GHz | |

1. (10 points) Suppose that we run three computer graphics and multimedia programs on all three smartphones:

**Program A**: Renders 114,000,000 pixels when viewing HW1.pdf.

**Program B**: Blurs the background of 2,000 images in the image gallery.

**Program C**: Processes 4,000 images in the image gallery.

For simplicity, we assume that the program only runs on a single core. The Samsung Galaxy S24 Ultra uses Cortex-X4, the Apple iPhone 15 Pro Max uses Everest, and the Google Pixel 8 Pro uses Cortex- X3. Furthermore, there is no other overhead. We are interested in the execution time (in seconds) and the clock cycles (in millions) of each smartphone. Use the provided information in the table and your answer in (a) to complete the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Smartphone | Program A | | Program B | | Program C | |
| Seconds | Clock Cycles | Seconds | Clock Cycles | Seconds | Clock Cycles |
| Samsung (Cortex-X4) | 0.50 | 1650 | 74.91 | 247,203 | 61.63 | 203,379 |
| Apple (Everest) | 0.64 | 2419.2 | 71.68 | 270,950.4 | 50.57 | 191,154.6 |
| Google (Cortex-X3) | 0.75 | 2182.5 | 133.33 | 387,990.3 | 85.11 | 247,670.1 |

1. (10 points) We are interested in comparing the performances of the three smartphones. Calculate the relative performance of the three smartphones, with each phone as the reference for comparison. Use your answer in (b) to complete the table below and summarize the performance results by calculating the **geometric mean of the performance ratio** of the three benchmark programs (Program A, Program B, and Program C). **Hint**: you might only need to compute some of the six values from scratch.

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Performance Ratio | | |
| Samsung Galaxy S24  Ultra | Apple iPhone 15 Pro  Max | Google Pixel 8 Pro |
| Samsung Galaxy S24 Ultra | 1 | 0.998 | 0.647 |
| Apple iPhone 15 Pro Max | 1.002 | 1 | 0.648 |
| Google Pixel 8 Pro | 1.545 | 1.542 | 1 |

1. (10 points) ***Performance and speedup***

Assume that a program requires the execution of 100 × 106 FP instructions, 140 × 106 INT instructions, 110 × 106 L/S instructions, and 55 × 106 branch instructions. The CPI for each type of instruction is 3, 2, 5, and 3, respectively. Assume that the processor has a 5 GHz clock rate.

1. (5 points) By how much must we improve the CPI of INT instructions if we want the program to run two times faster? Please show the calculation procedure.

FP: 300 \* 106 clock cycles

INT: 280 \* 106 clock cycles

L/S: 110 \* 106 clock cycles

Branch: 165 \* 106 clock cycles

(300 + 280 + 110 + 165) / 2 = 427.5 > 280

Impossible

1. (5 points) By how much is the execution time of the program improved if the CPI of FP instructions is reduced by 28%, the CPI of INT instructions is reduced by 32% and the CPI of L/S instructions is reduced by 61% and the CPI of branch instructions is reduced by 64%? Please show the calculation procedure.

(300 \* 0.72 + 280 \* 0.68 + 110 \* 0.39 + 165 \* 0.36) / (300 + 280 + 110 + 165) ≒ 0.595

1 – 0.595 = 40.5%

The execution time is reduced by 40.5%

1. (15 points) ***Amdahl’s law and the eight great ideas of computer architecture***

One of the great ideas of computer architecture is parallelization. Amdahl’s law can be used to calculate the overall speedup of parallel executions. Amdahl's Law is defined as follow:

1

where

𝑆𝑙𝑎𝑡𝑒𝑛𝑐𝑦 =

(1 − 𝑝) + 𝑝

𝑠

|  |  |
| --- | --- |
| 𝑆𝑙𝑎𝑡𝑒𝑛𝑐𝑦 | : the theoretical speedup of the execution of the whole task, |
| 𝑠 | : the speedup of the part of the task that benefits from improved system resources, |
| 𝑝 | : the proportion of the execution time that the part benefiting from improved resources originally  occupied. |

The ideal speedup of a parallelized program is the number of processors used. However, the theoretical speedups have limitations by the percentage of the application that cannot be parallelized, which includes the communication costs. The problem is that the communication costs are not fixed but often vary based on the number of processors used. In the following, let us consider the communication costs separately from the non-parallelizable execution of the program.

1. (5 points) Suppose we have a method to parallelize the fast\_power\_iter() function in (1) using an arbitrary number of processors. Moreover, the execution time 𝑇 on one processor is the result obtained in (1)(c). Compute the parallel execution time of fast\_power\_iter() on 2, 4, 8 processors assuming 75% of the function is parallelizable and there is no communication cost.

2 processors: 0.25 \* 5.39 + 0.75 \* 5.39 / 2 = 3.36875 ps

4 processors: 0.25 \* 5.39 + 0.75 \* 5.39 / 4 = 2.358125 ps

8 processors: 0.25 \* 5.39 + 0.75 \* 5.39 / 8 = 1.8528125 ps

1. (5 points) Assuming the communication costs are 4% of the original execution time regardless of the number of cores, what is the speedup with 8 cores when 75% of the program is parallelizable?

Speedup = (5.39 \* 1.04) / (0.25 \* 5.39 \* 1.04 + 0.75 \* 5.39 \* 1.04 / 8)

= (5.39 \* 1.04) / (1.8528125 \* 1.04) ≒ 2.909

1. (5 points) Assuming the communication costs are increased by 2% of the original execution time every time the number of processors is doubled, what is the speedup with n cores when 75% of the program is parallelizable? Furthermore, what is the specific speedup value when n = 8 in this scenario?

Let k = log2n

Speedup with n cores = (5.39 \* 1.02) / (0.25 \* 5.39 \* 1.02 + 0.75 \* 5.39 \* 1.02k / n)

Speedup with 8 cores = (5.39 \* 1.02) / (0.25 \* 5.39 \* 1.02 + 0.75 \* 5.39 \* 1.023 / 8)

≒ 2.877

1. (10 points) ***Integrated circuit cost and manufacturing***

Assume that a 50mm diameter-wafer has a cost of $9 and contains 95 dies. The yield for this wafer is 90%.

1. (4 points) Find the defects per area for this wafer using the Equation on Page 28 of the textbook (or Page 28 of the slide).

(1 / 3) \* 2 / ((252π) / 95) ≒ 0.0323 defects / mm2

1. (2 points) Find the cost per die for this wafer.

9 / (95 \* 0.9) ≒ 0.105 dollars / die

1. (4 points) If the number of dies per wafer is increased by 10% and the defects per area unit increases by 25%, find the new die area and new yield.

New die area = (252π) / (95 \* 1.1) ≒ 18.789 mm2

New yield = 1 / (1 + (0.0323 \* 1.25) \* (252π / (95 \* 1.1 \* 2)))2 ≒ 0.526 = 52.6%