Recitation 1

CS 3853: Computer Architecture

Introduction

- TA: Riad Akram
- Email: riad115@csebuet.org
- Office Hours:
  - TBD
- What to cover:
  - Practice problems covering class lectures

Amdahl's law

1. Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of a Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

Amdahl’s law

1. Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of a Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

Amdahl's law assumes that a program consists of a serial part and a parallelizable part.
The fraction of the program that is serial can be denoted as B.
So the parallel fraction becomes 1- B.

Amdahl’s law

1. Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of a Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

Amdahl's law assumes that a program consists of a serial part and a parallelizable part.
The fraction of the program that is serial can be denoted as B.
So the parallel fraction becomes 1- B.

If there is no additional overhead due to parallelization,
The speedup can therefore be expressed as $S(n) = [B + (1 -B)/n]^{-1}$

For the given value of B = 0.01 we get $S(61) = [0.01 + (1-0.01)/61]^{-1}$

$= 38.125$

Amdahl’s law

2. By Amdahl’s law, it does not make much sense to run a program on millions of cores, if there is only a small fraction of sequential code (which is often inevitable, i.e., reading input data). Why do people build such systems anyway?

Amdahl’s law

2. By Amdahl’s law, it does not make much sense to run a program on millions of cores, if there is only a small fraction of sequential code (which is often inevitable, i.e., reading input data). Why do people build such systems anyway?

Amdahl's law assumes that if the problem size is kept constant then by adding more processors the same problem gets solved faster.

In High Performance Computing (HPC) it is often the case that bigger computers are used to solve bigger problems, not to solve old problems faster.

If the sequential part of the program does not increase when increasing the input (or increases sub linearly) we can run on a large number of cores.

Amdahl’s law

3. Suppose that we are considering an enhancement that runs 10 times faster than the original machine but is usable only 40% of the time. What is the overall speedup gained by incorporating the enhancement?

Amdahl’s law

3. Suppose that we are considering an enhancement that runs 10 times faster than the original machine but is usable only 40% of the time. What is the overall speedup gained by incorporating the enhancement?

- According to Amdahl's law:

  $$\text{Speedup} = [\ 1 - FE + (\ FE/SE)]^{-1}$$

SE = 10

FE = 40/100 = 0.4

$$\text{Speedup} = [\ 1 - .4 + (\ .4/10)]^{-1}$$

Speedup = 1.56

Amdahl’s law

3. Speedup Floating point square root operation. Which is better: Design1 or Design2?

Design 1: The operation uses FPSQR hardware. FPSQR is responsible for 20% of the square root execution time. Speedup this component by a factor of 10.

Design 2: Make all the floating point (FP) instructions run faster. How much faster? 2 times faster. FP instructions are responsible for 50% of the square root execution time.

Amdahl's law

4. Speedup Floating point square root operation. Which is better: Design1 or Design2?

Design 1: The operation uses FPSQR hardware. FPSQR is responsible for 20% of the square root execution time. Speedup this component by a factor of 10.

Design 2: Make all the floating point (FP) instructions run faster. How much faster? 2 times faster. FP instructions are responsible for 50% of the square root execution time.

- According to Amdahl's law:

  $Speedup = [\ 1 - FE + (\ FE/SE)]^{-1}$

Design 1:

SE = 10

FE = 20/100 = 0.2

$Speedup = [\ 1 - .2 + (\ .2/10)]^{-1}$

Speedup = 1.22

Design 2:

SE = 2

FE = 50/100 = 0.5

$Speedup = [\ 1 - .5 + (\ .5/2)]^{-1}$

Speedup = 1.33

Design 2 is better.

CPU Performance

CPU time = Instruction count * CPI * Clock cycle time

CPU Performance

5. Suppose we have made the following measurements

Frequency of FP instructions: 25%

Average CPI of FP instructions: 4.0

Average CPI of other instructions: 1.33

Frequency of FPSQR = 2%

CPI of FPSQR = 20

Design 1: Reduce CPI of FPSQR from 20 to 2.

Design 2: Reduce average CPI of all FP instruction to 2.

Compare these two designs using CPU Perf. Equation.

CPU Performance

5. Suppose we have made the following measurements

Frequency of FP instructions: 25%

Average CPI of FP instructions: 4.0

Average CPI of other instructions: 1.33

Frequency of FPSQR = 2%

CPI of FPSQR = 20

Design 1: Reduce CPI of FPSQR from 20 to 2.

Assume, Instruction count = 1, Clock cycle time = 1

$CPI_{Original} = 75\% * 1.33 + 25\% * 4.0 = 2.0$

With Design 1 enhancement,

$Total\ CPI_{With\ new\ FPSQR} = 2.0 - 2\%*(20-2) = 1.64$

$Total\ CPU\ time_{With\ new\ FPSQR} = 1 * 1.64 * 1 = 1.64$

CPU Performance

5. Suppose we have made the following measurements

Frequency of FP instructions: 25%
Average CPI of FP instructions: 4.0
Average CPI of other instructions: 1.33
Frequency of FPSQR = 2%
CPI of FPSQR = 20

Design 2: Reduce average CPI of all FP instruction to 2.

Assume, Instruction count = 1, Clock cycle time = 1

$CPI_{Original} = 75\% * 1.33 + 25\% * 4.0 = 2.0$

With Design 2 enhancement,

$Total\ CPI_{new\ FP} = 75\% * 1.33 + 25\% * 2.0 = 1.50$

$Total\ CPU\ time_{new\ FP} = 1 * 1.50 * 1 = 1.50$

Design 2 is better as the CPU time of design 2 is less than design 1