HW2 - Arman Behnam - A20522906

My Email: abehnam@hawk.iit.edu

1.5

a) To determine which processor has the highest performance expressed in instructions per second, we calculate the MIPS for each processor, by using the following formula:

MIPS = Clock Rate (GHz) / CPI

- P1: MIPS = 3 GHz / 1.5 = 2 MIPS

- P2: MIPS = 2.5 GHz / 1.0 = 2.5 MIPS

- P3: MIPS = 4.0 GHz / 2.2 = 1.818 MIPS

So, processor P2 has the highest performance expressed in instructions per second with 2.5 MIPS.

b) To find the number of cycles and the number of instructions for each processor when executing a program in 10 seconds:

Total Cycles = Clock Rate (GHz) * Execution Time (seconds)

Total Instructions = Total Cycles / CPI

- **For P1:**

- Total Cycles = 3 GHz * 10 seconds = 30 billion cycles

- Total Instructions = 30 billion cycles / 1.5 = 20 billion instructions

- **For P2:**

- Total Cycles = 2.5 GHz * 10 seconds = 25 billion cycles

- Total Instructions = 25 billion cycles / 1.0 = 25 billion instructions

- **For P3:**

- Total Cycles = 4.0 GHz * 10 seconds = 40 billion cycles

- Total Instructions = 40 billion cycles / 2.2 = 18.18 billion instructions

c) To reduce the execution time by 30% while increasing the CPI by 20%, we find the new execution time (Tnew) in seconds:

Tnew = Old Execution Time * (1 + Percent Reduction in Time) = 10 seconds * (1 - 0.30) = 7 seconds

CPInew = Old CPI * (1 + Percent Increase in CPI) = 2.2 * (1 + 0.20) = 2.64

To calculate the required clock rate (New Clock Rate) to achieve the new execution time with the new CPI, we can use the following formula:

New Clock Rate = Total Instructions / (CPInew * Tnew) = 18.18 billion instructions / (2.64 * 7 seconds) ≈ 976.13 MHz

So, to reduce the execution time by 30% while increasing the CPI by 20%, we need a clock rate of approximately 976.13 MHz.

1.6

To determine which implementation is faster between P1 and P2, we need to compare their execution times based on the given program with a dynamic instruction count and CPIs for different instruction classes. We'll calculate the global CPI (Cycles Per Instruction) and the clock cycles required for both implementations.

- P1: Clock rate = 2.5 GHz, CPIs (class A, B, C, D) = (1, 2, 3, 3)

- P2: Clock rate = 3 GHz, CPIs (class A, B, C, D) = (2, 2, 2, 2)

- Program dynamic instruction count = 1.0E6 instructions

- Instruction distribution: 10% class A, 20% class B, 50% class C, 20% class D

a) Calculate the global CPI for each implementation:
   - For P1:

Global CPI_P1 = (Fraction of class A instructions * CPI_A + Fraction of class B instructions * CPI_B + Fraction of class C instructions * CPI_C + Fraction of class D instructions * CPI_D) = (0.10 * 1 + 0.20 * 2 + 0.50 * 3 + 0.20 * 3) = 2.5

   - For P2:

Global CPI_P2 = (Fraction of class A instructions * CPI_A + Fraction of class B instructions * CPI_B + Fraction of class C instructions * CPI_C + Fraction of class D instructions * CPI_D) = (0.10 * 2 + 0.20 * 2 + 0.50 * 2 + 0.20 * 2) = 2.0

b) Calculate the clock cycles required for both implementations:

Clock Cycles = (Dynamic Instruction Count) * (Global CPI) / (Clock Rate)

   - For P1:

Clock Cycles_P1 = (1.0E6 instructions) * (2.5) / (2.5 GHz) = 1.0E6 cycles

   - For P2:

Clock Cycles_P2 = (1.0E6 instructions) * (2.0) / (3.0 GHz) ≈ 666,667 cycles

Now, comparing the clock cycles required, P2 requires fewer clock cycles (approximately 666,667 cycles) compared to P1 (1.0E6 cycles). Therefore, implementation P2 is faster for the given program and instruction distribution.

1.7

a. To find the average CPI (Cycles Per Instruction) for each program:

CPI = (Total Cycles) / (Total Instructions)

- For compiler A:

Total Cycles_A = (Execution Time_A) / (Clock Cycle Time)

Total Cycles_A = 1.1 seconds / (1E-9 seconds/cycle) = 1.1E9 cycles

Average CPI_A = Total Cycles_A / (Dynamic Instruction Count_A)

Average CPI_A = 1.1E9 cycles / 1.0E9 instructions = 1.1 CPI

- For compiler B:

Total Cycles_B = (Execution Time_B) / (Clock Cycle Time)

Total Cycles_B = 1.5 seconds / (1E-9 seconds/cycle) = 1.5E9 cycles

Average CPI_B = Total Cycles_B / (Dynamic Instruction Count_B)

Average CPI_B = 1.5E9 cycles / 1.2E9 instructions = 1.25 CPI

b. If the execution times on two processors running the compiled programs are the same:

Execution Time = (Dynamic Instruction Count) * (Average CPI) * (Clock Cycle Time)

Let's denote the clock cycle time of the processor running compiler A's code as T_A and the clock cycle time of the processor running compiler B's code as T_B.

- For compiler A: 1.1 seconds = 1.0E9 instructions * 1.1 CPI * T_A

T_A = 1.1 seconds / (1.0E9 instructions * 1.1 CPI) ≈ 1 ns

- For compiler B: 1.5 seconds = 1.2E9 instructions * 1.25 CPI * T_B

T_B = 1.5 seconds / (1.2E9 instructions * 1.25 CPI) ≈ 1 ns

Both processors have a clock cycle time of approximately 1 ns, so there is no significant difference in clock frequency between them.

c. To calculate the speedup of using the new compiler versus using compiler A or B on the original processor:

Speedup = (Execution Time with Old Compiler) / (Execution Time with New Compiler)

- For compiler A: Speedup_A = 1.1 seconds / Execution Time with New Compiler

- For compiler B: Speedup_B = 1.5 seconds / Execution Time with New Compiler

Now, we need to find the execution time with the new compiler. We are given that the new compiler uses 6.0E8 instructions and has an average CPI of 1.1:

Execution Time with New Compiler = (Dynamic Instruction Count with New Compiler) * (Average CPI with New Compiler) * (Clock Cycle Time) = (6.0E8 instructions) * (1.1 CPI) * (1 ns) = 6.6E8 ns

- For compiler A: Speedup_A = 1.1 seconds / 6.6E8 ns ≈ 1.6667
- For compiler B: Speedup_B = 1.5 seconds / 6.6E8 ns ≈ 2.2727

So, the speedup of using the new compiler versus using compiler A is approximately 1.6667, and the speedup versus using compiler B is approximately 2.2727.

1.9

1.9.1

To find the total execution time for the program on 1, 2, 4, and 8 processors and calculate the relative speedup, we'll need to consider the CPI (Cycles Per Instruction) for each instruction type and how the instruction counts change as the program is parallelized.

Given:

- Arithmetic CPI (CPI_arithmetic) = 1

- Load/Store CPI (CPI_load_store) = 12

- Branch CPI (CPI_branch) = 5

- Clock frequency (Clock Rate) = 2 GHz = 2.0E9 Hz

Initial instruction counts:

- Arithmetic instructions = 2.56E9

- Load/Store instructions = 1.28E9

- Branch instructions = 256 million = 256E6

On a single processor:

Total Cycles = (Arithmetic instructions * CPI_arithmetic + Load/Store instructions * CPI_load_store + Branch instructions * CPI_branch)

Total Cycles = (2.56E9 * 1 + 1.28E9 * 12 + 256E6 * 5) = 46.08E9 cycles

Execution Time (T_single) = Total Cycles / Clock Rate

T_single = 46.08E9 cycles / 2.0E9 Hz = 23.04 seconds

Now, as the program is parallelized, the number of arithmetic and load/store instructions per processor is divided by 0.7 * p (where p is the number of processors), but the number of branch instructions per processor remains the same.

Let's calculate for different numbers of processors (p = 1, 2, 4, 8):

- For 2 processors:

Arithmetic instructions per processor = (2.56E9 / (0.7 * 2)) = 1.82857E9

Load/Store instructions per processor = (1.28E9 / (0.7 * 2)) = 914.2857E6

Branch instructions per processor = 256E6 (unchanged)

Total Cycles for 2 processors = (Arithmetic instructions per processor * CPI_arithmetic + Load/Store instructions per processor * CPI_load_store + Branch instructions per processor * CPI_branch) = (1.82857E9 * 1 + 914.2857E6 * 12 + 256E6 * 5) = 22.97142E9 cycles

Execution Time (T_2_processors) = Total Cycles for 2 processors / Clock Rate

T_2_processors = 22.97142E9 cycles / 2.0E9 Hz = 11.49 seconds

- For 4 processors:

Arithmetic instructions per processor = (2.56E9 / (0.7 * 4)) = 914.2857E6

Load/Store instructions per processor = (1.28E9 / (0.7 * 4)) = 457.1429E6

Branch instructions per processor = 256E6 (unchanged)

Total Cycles for 4 processors = (Arithmetic instructions per processor * CPI_arithmetic + Load/Store instructions per processor * CPI_load_store + Branch instructions per processor * CPI_branch) = (914.2857E6 * 1 + 457.1429E6 * 12 + 256E6 * 5) = 11.48571E9 cycles

Execution Time (T_4_processors) = Total Cycles for 4 processors / Clock Rate

T_4_processors = 11.48571E9 cycles / 2.0E9 Hz = 5.74285 seconds

- For 8 processors:

Arithmetic instructions per processor = (2.56E9 / (0.7 * 8)) = 457.1429E6

Load/Store instructions per processor = (1.28E9 / (0.7 * 8)) = 228.5714E6

Branch instructions per processor = 256E6 (unchanged)

Total Cycles for 8 processors = (Arithmetic instructions per processor * CPI_arithmetic + Load/Store instructions per processor * CPI_load_store + Branch instructions per processor * CPI_branch) = (457.1429E6 * 1 + 228.5714E6 * 12 + 256E6 * 5) = 5.74285E9 cycles

Execution Time (T_8_processors) = Total Cycles for 8 processors / Clock Rate

T_8_processors = 5.74285E9 cycles / 2.0E9 Hz = 2.87142 seconds

Speedup_2_processors = T_single / T_2_processors, Speedup_4_processors = T_single / T_4_processors, Speedup_8_processors = T_single / T_8_processors, Speedup_2_processors ≈ 2.0, Speedup_4_processors ≈ 4.0, and Speedup_8_processors ≈ 8.0

So, the relative speedup for 2, 4, and 8 processors compared to a single processor is 2.0x, 4.0x, and 8.0x, respectively.

1.9.2

Now, comparing the clock cycles required, P2 requires fewer clock cycles (approximately 666,667 cycles) compared to P1 (1.0E6 cycles). Therefore, implementation P2 is faster for the given program and instruction distribution. By doubling the CPI of Arithmetic Instructions:

- Original CPI of arithmetic instructions (CPI_arithmetic) = 1
- New CPI of arithmetic instructions (CPI_arithmetic_new) = 2

The impact on execution time for the program on 1, 2, 4, or 8 processors:

Execution Time_new = (Total Cycles) / Clock Rate

Where Total Cycles is determined by considering the new CPI for arithmetic instructions and the original CPI values for load/store and branch instructions.

The impact on execution time for 4 processors:

Total Cycles for 4 processors (with new CPI for arithmetic instructions):

Total Cycles_new = (Arithmetic instructions per processor * CPI_arithmetic_new + Load/Store instructions per processor * CPI_load_store + Branch instructions per processor * CPI_branch) = (Arithmetic instructions per processor * 2 + Load/Store instructions per processor * CPI_load_store + Branch instructions per processor * CPI_branch)

1.9.3

To match the performance of four processors using the original CPI values, the single processor should have an execution time equal to the execution time of the parallelized program on four processors.

Execution Time_single_processor = Execution Time_parallelized_program_on_4_processors

The execution time on a single processor can be calculated as:

Execution Time_single_processor = (Total Cycles_single_processor) / Clock Rate

Where Total Cycles_single_processor is determined by the original CPI values. By comparing the Execution **Time_new** with the doubled CPI for arithmetic instructions to the Execution **Time_single_processor** with the modified CPI for load/store instructions, we can determine the impact of these changes on execution time and the required CPI reduction for load/store instructions.

1.11

1.11.1

To find the CPI (Cycles Per Instruction):

CPI = (Total Cycles) / (Total Instructions)

Given the information:

- Instruction count = 2.389E12 instructions

- Execution time = 750 seconds

- Clock cycle time = 0.333 ns (which is equivalent to 0.333E-9 seconds)

Total Cycles = Execution Time / Clock Cycle Time = 750 seconds / 0.333E-9 seconds/cycle = 2.25E18 cycles

CPI = Total Cycles / Instruction count = 2.25E18 cycles / 2.389E12 instructions ≈ 941.24

So, the CPI for the SPEC CPU2006 bzip2 benchmark on an AMD Barcelona processor with a clock cycle time of 0.333 ns is approximately 941.24.

1.11.2

The SPECratio, is a metric used to compare the performance of a computer system when running SPEC CPU benchmarks to a reference system. SPECratio = (Reference Time) / (Execution Time)

Given the information: Reference time = 9650 seconds, Execution time = 750 seconds

SPECratio = 9650 seconds / 750 seconds ≈ 12.867

So, the SPECratio for the SPEC CPU2006 bzip2 benchmark on the AMD Barcelona processor is approximately 12.867. This indicates that the AMD Barcelona processor's performance on this benchmark is approximately 12.867 times better than the reference system.

1.11.3

If the number of instructions in the benchmark is increased by 10% without affecting the CPI (Cycles Per Instruction):

Increase in CPU Time = (New Instruction Count - Original Instruction Count) * CPI * Clock Cycle Time

Given that the benchmark originally had an instruction count of 2.389E12 instructions and no change in CPI, and the clock cycle time is 0.333 ns (0.333E-9 seconds):

Original Instruction Count = 2.389E12 instructions

New Instruction Count = 1.10 * Original Instruction Count = 1.10 * 2.389E12 = 2.6279E12 instructions

Increase in CPU Time = (2.6279E12 instructions - 2.389E12 instructions) * CPI * 0.333E-9 seconds/cycle

Since the CPI is unchanged, it remains the same as calculated previously, which is approximately 941.24.

Increase in CPU Time ≈ (2.6279E12 - 2.389E12) * 941.24 * 0.333E-9 seconds ≈ 80.74 seconds

So, increasing the number of instructions in the benchmark by 10% without affecting the CPI would result in an increase in CPU time of approximately 80.74 seconds.

1.11.4

To find the increase in CPU time when both the number of instructions and the CPI (Cycles Per Instruction) are changed:

Increase in CPU Time = (New Instruction Count - Original Instruction Count) * (New CPI - Original CPI) * Clock Cycle Time

Given:

- Original Instruction Count = 2.389E12 instructions

- CPI (Original) = 941.24

- Increase in Instructions = 10% increase = 1.10 * Original Instruction Count

- Increase in CPI = 5% increase = 1.05 * Original CPI

- Clock Cycle Time = 0.333 ns (0.333E-9 seconds)

New Instruction Count = 1.10 * Original Instruction Count = 1.10 * 2.389E12 = 2.6279E12 instructions

New CPI = 1.05 * Original CPI = 1.05 * 941.24 = 988.30

Increase in CPU Time = (2.6279E12 - 2.389E12) * (988.30 - 941.24) * 0.333E-9 seconds/cycle

Increase in CPU Time ≈ (2.389E11) * (47.06) * 0.333E-9 seconds ≈ 3.58 seconds

So, increasing both the number of instructions in the benchmark by 10% and the CPI by 5% would result in an increase in CPU time of approximately 3.58 seconds.

1.11.5

To find the change in the SPECratio for this change:

Change in SPECratio = (Reference Time) / (New Execution Time) - (Reference Time) / (Original Execution Time)

Given the information:

- Reference Time = 9650 seconds

- Original Execution Time (before the change) = 750 seconds

- Increase in CPU Time (from the previous calculation) = 3.58 seconds

- New Execution Time (after the change) = Original Execution Time + Increase in CPU Time

New Execution Time = 750 seconds + 3.58 seconds = 753.58 seconds

Change in SPECratio = (9650 seconds) / (753.58 seconds) - (9650 seconds) / (750 seconds)

Change in SPECratio ≈ 12.8124 - 12.8667 ≈ -0.0543

So, the change in SPECratio for this change is approximately -0.0543. This means that the SPECratio decreased by approximately 0.0543 compared to the original SPECratio, indicating a slight decrease in performance for the benchmark after increasing both the number of instructions and the CPI.

1.11.6

To find the new CPI (Cycles Per Instruction) for the AMD Barcelona processor with a 4 GHz clock rate after adding additional instructions and reducing the instruction count by 15%:

SPECratio = (Reference Time) / (Execution Time)

Given:

- Reference Time (from the original AMD Barcelona) = 9650 seconds

- New Execution Time (after adding instructions and reducing instruction count) = 700 seconds

- New Clock Rate = 4 GHz = 4.0E9 Hz

- The number of instructions has been reduced by 15%, which means the new instruction count (New Instruction Count) is 85% of the original instruction count (Original Instruction Count).

Using the SPECratio formula, we can rewrite it as:

SPECratio = (Reference Time) / (Execution Time) = (Reference Time) / [(New Instruction Count) * CPI_new * (1 / New Clock Rate)]

CPI_new = (Reference Time) / [(New Instruction Count) * SPECratio * (1 / New Clock Rate)] = (9650 seconds) / [(0.85 * Original Instruction Count) * 13.7 * (1 / 4.0E9 Hz)] = (9650 seconds) / [(0.85 * 2.389E12 instructions) * 13.7 * (1 / 4.0E9 Hz)] ≈ 1.110

So, the new CPI for the AMD Barcelona processor with a 4 GHz clock rate, after adding additional instructions and reducing the instruction count by 15%, is approximately 1.110.

1.11.7

The increase in CPI from 1.11 (previous calculation with a 3 GHz clock rate) to approximately 1.110 (current calculation with a 4 GHz clock rate) does not follow a proportional increase with the clock rate. Instead, it's slightly lower than 1.11, even though the clock rate has increased.

This behavior is because CPI is influenced by various factors, including the microarchitecture of the processor, the mix of instructions in the workload, and the efficiency of the instruction execution pipeline. When the clock rate increases, it may lead to reduced cycle times, which can result in fewer cycles per instruction for some instructions, offsetting the CPI increase that might be expected solely due to a clock rate increase.

However, the specific impact on CPI can vary depending on the design changes made to the processor and the workload characteristics. A lower CPI with a higher clock rate suggests that the microarchitecture enhancements and instruction set changes were effective in improving instruction execution efficiency, but it doesn't necessarily imply a linear relationship between clock rate and CPI.

1.11.8

To calculate how much the CPU time has been reduced:

Change in CPU Time = Original CPU Time - New CPU Time

Given:

- Original CPU Time (before the change) = 9650 seconds (reference time)

- New CPU Time (after the change) = 700 seconds (from the provided information)

Change in CPU Time = 9650 seconds - 700 seconds = 8950 seconds

The CPU time has been reduced by 8950 seconds.

1.11.9

To determine the number of instructions for the libquantum benchmark when the execution time is reduced by an additional 10% without affecting the CPI and with a clock rate of 4 GHz:

Execution Time = (Total Instructions) * (CPI) * (Clock Cycle Time)

Given:

- Original Execution Time (before the reduction) = 960 ns = 0.960 μs (microseconds)

- Original CPI (before the reduction) = 1.61

- Original Clock Rate (before the reduction) = 3 GHz = 3.0E9 Hz

- New Clock Rate (after the reduction) = 4 GHz = 4.0E9 Hz

We want to find the new total instructions (New Total Instructions) after the execution time reduction:

New Execution Time (after the reduction) = 0.9 * Original Execution Time (reduced by 10%) = 0.9 * 0.960 μs = 0.864 μs

New Total Instructions = New Execution Time / (CPI * Clock Cycle Time)

First, calculate the Clock Cycle Time for the original and new clock rates:

- Original Clock Cycle Time = 1 / Original Clock Rate = 1 / (3.0E9 Hz) = 0.333 ns (nanoseconds)

- New Clock Cycle Time = 1 / New Clock Rate = 1 / (4.0E9 Hz) = 0.25 ns (nanoseconds)

New Total Instructions = (0.864 μs) / (1.61 * 0.25 ns)

New Total Instructions = (0.864 * 1000 ns) / (1.61 * 0.25 ns) ≈ (864 / 40.65) ≈ 21.27E6 instructions

So, when the execution time is reduced by an additional 10% without affecting the CPI and with a clock rate of 4 GHz, the number of instructions for the libquantum benchmark is approximately 21.27 million instructions.

1.11.10

To achieve a further 10% reduction in CPU time while maintaining the number of instructions and with the CPI unchanged:

New Execution Time = 0.9 * Original Execution Time

We want to find the clock rate required for this reduced execution time. Let's denote the new clock rate as New Clock Rate.

Original Execution Time = Total Instructions * CPI * Clock Cycle Time

New Execution Time = Total Instructions * CPI * New Clock Cycle Time

Total Instructions * CPI * Clock Cycle Time = Total Instructions * CPI * New Clock Cycle Time

Clock Cycle Time = New Clock Cycle Time

So, to achieve a further 10% reduction in CPU time while maintaining the number of instructions and with the CPI unchanged, you would need to reduce the clock cycle time to 90% of its original value.

New Clock Rate = 1 / New Clock Cycle Time

Original Clock Cycle Time = 0.25 ns (as calculated previously)

New Clock Cycle Time = 0.9 * 0.25 ns = 0.225 ns

New Clock Rate = 1 / (0.225 ns) = 4.444 GHz

So, to achieve a further 10% reduction in CPU time while maintaining the number of instructions and with the CPI unchanged, you would need a clock rate of approximately 4.444 GHz.

1.11.11

To determine the required clock rate when the CPI is reduced by 15% and the CPU time is reduced by 20% while keeping the number of instructions unchanged:

New Execution Time = 0.8 * Original Execution Time (20% reduction)

New CPI = 0.85 * Original CPI (15% reduction)

Execution Time = Total Instructions * CPI * Clock Cycle Time

Given that the number of instructions (Total Instructions) is unchanged, we can write:

Original CPI * Clock Cycle Time = New CPI * New Clock Cycle Time

Original CPI * Clock Cycle Time = 0.85 * Original CPI * New Clock Cycle Time

Clock Cycle Time = 0.85 * New Clock Cycle Time

Now, we know that the new execution time is 80% of the original execution time, and the clock cycle time is 85% of the new clock cycle time:

Original Execution Time = Total Instructions * Original CPI * Clock Cycle Time

New Execution Time = Total Instructions * New CPI * New Clock Cycle Time

Total Instructions * Original CPI * Clock Cycle Time = Total Instructions * New CPI * New Clock Cycle Time

Original CPI * Clock Cycle Time = 0.85 * New CPI * 0.85 * New Clock Cycle Time

Clock Cycle Time = 0.85 * 0.85 * New Clock Cycle Time

New Clock Cycle Time = Clock Cycle Time / (0.85 * 0.85)

New Clock Rate = 1 / New Clock Cycle Time

Clock Cycle Time (initial) = 1 / (3.0E9 Hz) = 0.333 ns

New Clock Cycle Time = 0.333 ns / (0.85 * 0.85) ≈ 0.4641 ns

New Clock Rate = 1 / New Clock Cycle Time ≈ 1 / (0.4641 ns) ≈ 2.155 GHz

So, to achieve a 15% reduction in CPI and a 20% reduction in CPU time while keeping the number of instructions unchanged, the required clock rate would be approximately 2.155 GHz.

1.12

1.12.1

To check if the computer with the largest clock rate has the largest performance between P1 and P2, we need to calculate the execution time for each processor and then compare their performances. Performance can be defined as the reciprocal of execution time. Therefore, lower execution time implies better performance.

- For P1:

- Clock Rate (P1) = 4 GHz = 4.0E9 Hz

- Average CPI (P1) = 0.9

- Instructions (P1) = 5.0E9

Execution Time (P1) = (Instructions * CPI) / Clock Rate = (5.0E9 * 0.9) / (4.0E9 Hz) = 1.125 seconds

- For P2:

- Clock Rate (P2) = 3 GHz = 3.0E9 Hz

- Average CPI (P2) = 0.75

- Instructions (P2) = 1.0E9

Execution Time (P2) = (Instructions * CPI) / Clock Rate = (1.0E9 * 0.75) / (3.0E9 Hz) = 0.25 seconds

- P1: Execution Time (P1) = 1.125 seconds

- P2: Execution Time (P2) = 0.25 seconds

Clearly, P2 has a significantly lower execution time compared to P1, which means P2 performs better in terms of execution speed, despite having a lower clock rate. This illustrates the pitfall of assuming that a computer with the largest clock rate has the largest performance.

1.12.2

To determine the number of instructions that processor P2 (with CPI unchanged) can execute in the same time that processor P1 needs to execute 1.0E9 instructions, we can use the concept of execution time and rearrange it for P2.

The formula for execution time is:

Execution Time = (Instructions * CPI) / Clock Rate

We are given that processor P1 executes 1.0E9 instructions. Let's denote the number of instructions that P2 can execute in the same time as "X." For P1:

- Instructions (P1) = 1.0E9

Execution Time (P1) = (Instructions * CPI) / Clock Rate = (1.0E9 * CPI_P1) / Clock Rate_P1

Now, we want to find the number of instructions (X) that P2 can execute in the same time:

Execution Time (P1) = Execution Time (P2)

(1.0E9 * CPI_P1) / Clock Rate_P1 = (X * CPI_P2) / Clock Rate_P2

We know that CPI_P1 = CPI_P2 (CPI doesn't change), and we want to find X: (1.0E9 * 1) / Clock Rate_P1 = (X * 1) / Clock Rate_P2

X = (1.0E9 * Clock Rate_P2) / Clock Rate_P1 = (1.0E9 * 3.0E9 Hz) / 4.0E9 Hz = (3.0E18) / (4.0E9) = 750 million

So, processor P2 can execute 750 million instructions in the same time that processor P1 needs to execute 1.0E9 instructions, assuming the CPI remains unchanged for both processors.

1.12.3

MIPS is a measure of instruction throughput, but it doesn't account for factors like CPI (Cycles Per Instruction) or the complexity of instructions. To check if the processor with the largest MIPS has the largest performance between P1 and P2, we can calculate the MIPS for both processors and see if it accurately reflects their relative performance.

MIPS = (Clock Rate / 1,000,000) * (1 / CPI)

For P1:

- Clock Rate (P1) = 4 GHz = 4,000,000,000 Hz

- Average CPI (P1) = 0.9

MIPS (P1) = (4,000,000,000 / 1,000,000) * (1 / 0.9) ≈ 4,444.44 MIPS

For P2:

- Clock Rate (P2) = 3 GHz = 3,000,000,000 Hz

- Average CPI (P2) = 0.75

MIPS (P2) = (3,000,000,000 / 1,000,000) * (1 / 0.75) = 4,000 MIPS

- P1: MIPS (P1) ≈ 4,444.44 MIPS

- P2: MIPS (P2) = 4,000 MIPS

Based on the MIPS values, P1 has a significantly higher MIPS rating compared to P2. However, this doesn't necessarily mean that P1 has the largest overall performance. As we've seen earlier, performance is determined by various factors, including CPI, clock rate, and the number of instructions executed.

In the previous analysis, we found that P2 (with a lower clock rate) can execute 750 million instructions in the same time that P1 executes 1.0E9 instructions. This indicates that P2 can potentially have better overall performance for certain workloads, even though it has a lower MIPS rating. Therefore, relying solely on MIPS can be misleading when comparing processors, and it's important to consider a combination of factors to assess overall performance accurately.

1.12.4

To find the MFLOPS figures for both processors P1 and P2:

MFLOPS = (No. FP operations) / (execution time * 1E6)

Given that 40% of the instructions executed on both P1 and P2 are floating-point instructions, let's denote the total number of instructions executed as "Total Instructions" and the number of floating-point operations per instruction as "FP Ops per Instruction."

We can calculate the number of floating-point operations for P1 and P2 as follows:

- For P1:

- Total Instructions (P1) = 5.0E9 instructions

- FP Ops per Instruction = 0.4 (40% of instructions are floating-point)

No. FP operations (P1) = Total Instructions (P1) * FP Ops per Instruction

MFLOPS (P1) = (No. FP operations (P1)) / (execution time (P1) * 1E6)

- For P2:

- Total Instructions (P2) = 1.0E9 instructions

- FP Ops per Instruction = 0.4 (40% of instructions are floating-point)

No. FP operations (P2) = Total Instructions (P2) * FP Ops per Instruction

MFLOPS (P2) = (No. FP operations (P2)) / (execution time (P2) * 1E6)

Let's calculate both MFLOPS figures:

- For P1:

No. FP operations (P1) = 5.0E9 * 0.4 = 2.0E9 FP operations

MFLOPS (P1) = (2.0E9) / (1.125 * 1E6) = 1,777.78 MFLOPS

- For P2:

No. FP operations (P2) = 1.0E9 * 0.4 = 4.0E8 FP operations

MFLOPS (P2) = (4.0E8) / (0.25 * 1E6) = 1,600 MFLOPS

So, for the given programs running on processors P1 and P2, the MFLOPS figures are as follows:

- P1: 1,777.78 MFLOPS

- P2: 1,600 MFLOPS

These figures provide an estimate of the floating-point processing capabilities of the two processors, taking into account the number of floating-point instructions and execution time.

1.13

1.13.1

To calculate how much the total time is reduced if the time for FP operations is reduced by 20%, we need to find the original total time and the reduced total time, and then calculate the difference.

Given:

- Original total time (before the reduction) = 250 seconds

- Time spent on FP operations (before the reduction) = 70 seconds

Reduced time for FP operations = 0.8 * Original time for FP operations = 0.8 * 70 seconds = 56 seconds

Reduced total time = Time for FP operations (reduced) + Time for L/S (Load/Store) operations + Time for branch operations = 56 seconds + 85 seconds + 40 seconds = 181 seconds

Reduction in total time = Original total time - Reduced total time = 250 seconds - 181 seconds = 69 seconds

So, the total time is reduced by 69 seconds if the time for FP operations is reduced by 20%.

1.13.2

To calculate how much the time for INT operations is reduced if the total time is reduced by 20%, we can use the fact that the total time is divided into time components for different types of operations. We already know that the total time before the reduction was 250 seconds.

We've also calculated that the total time after reducing the time for FP operations is 181 seconds.

Time for INT operations (reduced) = Reduced total time - Time for FP operations (reduced) = 181 seconds - 56 seconds = 125 seconds

Reduction in time for INT operations = Original time for INT operations - Time for INT operations (reduced) = 250 seconds - 125 seconds = 125 seconds

So, the time for INT operations is reduced by 125 seconds if the total time is reduced by 20%.

1.13.3

To determine if the total time can be reduced by 20% by reducing only the time for branch instructions:

Given:

- Original total time = 250 seconds

- Time for branch instructions (before reduction) = 40 seconds

We want to reduce the time for branch instructions. Let's calculate the reduced time for branch instructions if we achieve a 20% reduction:

Reduced time for branch instructions = 0.8 * Original time for branch instructions = 0.8 * 40 seconds = 32 seconds

Total time (reduced) = Original total time - Time for branch instructions reduction = 250 seconds - 8 seconds = 242 seconds

The total time after reducing only the time for branch instructions is 242 seconds.

Percentage reduction = [(Original total time - Total time (reduced)) / Original total time] * 100% = [(250 seconds - 242 seconds) / 250 seconds] * 100% = (8 seconds / 250 seconds) * 100% ≈ 3.2%

The total time is reduced by approximately 3.2% when only the time for branch instructions is reduced by 20%.

So, reducing only the time for branch instructions by 20% does not achieve the desired 20% reduction in the total time; it results in a smaller reduction of approximately 3.2%. To achieve a 20% reduction in the total time, other components of the execution time, such as FP and L/S instructions, would also need to be considered for reduction.

1.14

1.14.1

To determine how much we must improve the CPI of FP instructions in order to make the program run two times faster, we can use the formula for execution time and manipulate it to find the required CPI improvement.

Execution Time = (Total Instructions * CPI) / Clock Rate

We are given the following information:

- Total Instructions for each type of instruction:
  - FP Instructions = 50 × 10^6 instructions
  - INT Instructions = 110 × 10^6 instructions
  - L/S Instructions (Load/Store) = 80 × 10^6 instructions
  - Branch Instructions = 16 × 10^6 instructions
- CPI for each type of instruction:
  - CPI_FP = 1
  - CPI_INT = 1
  - CPI_L/S = 4
  - CPI_Branch = 2
- Clock Rate = 2 GHz = 2 × 10^9 Hz

We want to make the program run two times faster, which means the new execution time (T_new) will be half of the original execution time (T_original):

T_new = 0.5 * T_original

T_original = (Total Instructions * CPI) / Clock Rate

- For FP Instructions: T_FP_original = (50 × 10^6 * 1) / (2 × 10^9) = 25,000,000 / 2,000,000,000 = 0.0125 seconds
- For INT Instructions: T_INT_original = (110 × 10^6 * 1) / (2 × 10^9) = 55,000,000 / 2,000,000,000 = 0.0275 seconds

- For L/S Instructions: T_L/S_original = (80 × 10^6 * 4) / (2 × 10^9) = 320,000,000 / 2,000,000,000 = 0.16 seconds
- For Branch Instructions: T_Branch_original = (16 × 10^6 * 2) / (2 × 10^9) = 32,000,000 / 2,000,000,000 = 0.016 seconds

T_total_original = T_FP_original + T_INT_original + T_L/S_original + T_Branch_original = 0.0125 + 0.0275 + 0.16 + 0.016 = 0.216 seconds

T_new = 0.5 * T_total_original = 0.5 * 0.216 = 0.108 seconds

Now, we can calculate the required CPI improvement for FP Instructions (CPI_FP_new) to achieve this new execution time:

T_FP_new = (50 × 10^6 * CPI_FP_new) / (2 × 10^9)

CPI_FP_new = (T_FP_new * 2 × 10^9) / 50 × 10^6 = (0.108 * 2 × 10^9) / 50 × 10^6 = 2160 / 50 = 43.2

To make the program run two times faster, we must improve the CPI of FP Instructions to approximately 43.2.

1.14.2

To determine how much we must improve the CPI of L/S instructions in order to make the program run two times faster:

- Total Instructions for each type of instruction:

  - FP Instructions = 50 × 10^6 instructions

  - INT Instructions = 110 × 10^6 instructions

  - L/S Instructions (Load/Store) = 80 × 10^6 instructions

  - Branch Instructions = 16 × 10^6 instructions

- CPI for each type of instruction:

  - CPI_FP = 1

  - CPI_INT = 1

  - CPI_L/S = 4 (Original CPI for L/S instructions)

  - CPI_Branch = 2

- Clock Rate = 2 GHz = 2 × 10^9 Hz

We want to make the program run two times faster, which means the new execution time (T_new) will be half of the original execution time (T_original):

T_new = 0.5 * T_original

T_L/S_original = (80 × 10^6 * CPI_L/S) / (2 × 10^9) = (80 × 10^6 * 4) / (2 × 10^9) = 320,000,000 / 2,000,000,000 = 0.16 seconds

T_total_original = T_FP_original + T_INT_original + T_L/S_original + T_Branch_original = 0.0125 + 0.0275 + 0.16 + 0.016 = 0.216 seconds

T_new = 0.5 * T_total_original = 0.5 * 0.216 = 0.108 seconds

T_L/S_new = (80 × 10^6 * CPI_L/S_new) / (2 × 10^9)

CPI_L/S_new = (T_L/S_new * 2 × 10^9) / 80 × 10^6 = (0.108 * 2 × 10^9) / 80 × 10^6 = 2160 / 80 = 27

To make the program run two times faster, we must improve the CPI of L/S Instructions to approximately 27.

1.14.3

To calculate how much the execution time of the program is improved when the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch instructions is reduced by 30%

Given:

- Original CPI values:
  - CPI_FP_original = 1 (for FP instructions)
  - CPI_INT_original = 1 (for INT instructions)
  - CPI_L/S_original = 4 (for L/S instructions)
  - CPI_Branch_original = 2 (for Branch instructions)

We want to reduce the CPI values for INT and FP instructions by 40% and reduce the CPI values for L/S and Branch instructions by 30%. Let's calculate the new CPI values:

- CPI_FP_new = CPI_FP_original * (1 - 40%) = 1 * 0.6 = 0.6 (40% reduction for FP)

- CPI_INT_new = CPI_INT_original * (1 - 40%) = 1 * 0.6 = 0.6 (40% reduction for INT)

- CPI_L/S_new = CPI_L/S_original * (1 - 30%) = 4 * 0.7 = 2.8 (30% reduction for L/S)

- CPI_Branch_new = CPI_Branch_original * (1 - 30%) = 2 * 0.7 = 1.4 (30% reduction for Branch)

T_new = (Total Instructions * New CPI) / Clock Rate

- For FP Instructions: T_FP_new = (50 × 10^6 * 0.6) / (2 × 10^9) = 0.015 seconds
- For INT Instructions: T_INT_new = (110 × 10^6 * 0.6) / (2 × 10^9) = 0.033 seconds
- For L/S Instructions: T_L/S_new = (80 × 10^6 * 2.8) / (2 × 10^9) = 0.112 seconds
- For Branch Instructions: T_Branch_new = (16 × 10^6 * 1.4) / (2 × 10^9) = 0.0112 seconds

T_total_new = T_FP_new + T_INT_new + T_L/S_new + T_Branch_new = 0.015 + 0.033 + 0.112 + 0.0112 = 0.1712 seconds

To find the improvement in execution time, compare the original execution time (T_total_original = 0.216 seconds) with the new execution time (T_total_new = 0.1712 seconds):

Improvement = T_total_original - T_total_new = 0.216 - 0.1712 = 0.0448 seconds

The execution time of the program is improved by approximately 0.0448 seconds when the CPI of INT and FP instructions is reduced by 40%, and the CPI of L/S and Branch instructions is reduced by 30%.

1.15

We should compute the per-processor execution time for 2, 4, 8, 16, 32, 64, and 128 processors, and calculate the corresponding speedup relative to a single processor, as well as the ratio between actual speedup versus ideal speedup (assuming no overhead).

Given:

- Total execution time on one processor (t) = 100 seconds

- Overhead time for each processor (overhead) = 4 seconds

For each case (p processors), the per-processor execution time (T_p) is given by:

T_p = (Total execution time on one processor + Total overhead time) / p = (t + overhead) / p

- For 2 processors (p = 2): T_2 = (100 seconds + 4 seconds) / 2 = 52 seconds
- For 4 processors (p = 4): T_4 = (100 seconds + 4 seconds) / 4 = 26 seconds
- For 8 processors (p = 8): T_8 = (100 seconds + 4 seconds) / 8 = 13 seconds
- For 16 processors (p = 16): T_16 = (100 seconds + 4 seconds) / 16 = 6.5 seconds
- For 32 processors (p = 32): T_32 = (100 seconds + 4 seconds) / 32 = 3.25 seconds
- For 64 processors (p = 64): T_64 = (100 seconds + 4 seconds) / 64 = 1.625 seconds
- For 128 processors (p = 128): T_128 = (100 seconds + 4 seconds) / 128 = 0.8125 seconds

The speedup for each case relative to a single processor (S_p), which is the ratio of the per-processor execution time on one processor to the per-processor execution time for p processors:

S_p = T_1 / T_p (where T_1 is the execution time on one processor)

- For 2 processors: S_2 = 100 seconds / 52 seconds ≈ 1.923
- For 4 processors: S_4 = 100 seconds / 26 seconds ≈ 3.846
- For 8 processors: S_8 = 100 seconds / 13 seconds ≈ 7.692
- For 16 processors: S_16 = 100 seconds / 6.5 seconds ≈ 15.385
- For 32 processors: S_32 = 100 seconds / 3.25 seconds ≈ 30.769
- For 64 processors: S_64 = 100 seconds / 1.625 seconds ≈ 61.538
- For 128 processors: S_128 = 100 seconds / 0.8125 seconds ≈ 123.077

Ideal speedup (I_p) is the speedup that would be achieved if there were no overhead and each processor contributed equally to the computation:

I_p = p (assuming perfect parallelism)

Now, we calculate the ratio between actual speedup (S_p) and ideal speedup (I_p) for each case:

- For 2 processors: S_2 / I_2 = 1.923 / 2 = 0.9615
- For 4 processors: S_4 / I_4 = 3.846 / 4 = 0.9615
- For 8 processors: S_8 / I_8 = 7.692 / 8 = 0.9615
- For 16 processors: S_16 / I_16 = 15.385 / 16 = 0.9615
- For 32 processors: S_32 / I_32 = 30.769 / 32 = 0.9615
- For 64 processors: S_64 / I_64 = 61.538 / 64 = 0.9615
- For 128 processors: S_128 / I_128 = 123.077 / 128 = 0.9615

In all cases, the ratio between actual speedup and ideal speedup is approximately 0.9615, which means that the overhead introduces a consistent factor of about 3.85% reduction in performance compared to the ideal speedup scenario where there is no overhead.