CS 402 Homework 2

Muhammad Umar

1.5 Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

a. Which processor has the highest performance expressed in instructions per second?

```
library(kableExtra)
p1Stats = list("clock" = 3.0, "cpi" = 1.5)
p2Stats = list("clock" = 2.5, "cpi" = 1.0)
p3Stats = list("clock" = 4.0, "cpi" = 2.2)

cpuComparison <- data.frame(
  "clockGhz" = c(p1Stats$clock, p2Stats$clock, p3Stats$clock),
  "cpi" = c(p1Stats$cpi, p2Stats$cpi, p3Stats$cpi),
  "instructionsPerSecond" = c(
    10**9 * p1Stats$clock / p1Stats$cpi,
    10**9 * p2Stats$clock / p2Stats$cpi,
    10**9 * p3Stats$clock / p3Stats$cpi)
)

kable(cpuComparison,
      col.names = c(
        "Clock (GHz)",
        "CPI",
        "Instructions Per Second"
        )
      ) %>%
  kable_styling()
```

Clock (GHz)	CPI	Instructions Per Second
3.0	1.5	2000000000
2.5	1.0	2500000000
4.0	2.2	1818181818

Hence, P2, has the highest performance with 250 million instructions per second.

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

```
cpuComparisonB <- cpuComparison
cpuComparisonB$cyclesIn10Seconds <- 10 * cpuComparison$clockGhz * 10**9
cpuComparisonB$instructionsIn10Seconds <- 10 * cpuComparison$instructionsPerSecond
kable(cpuComparisonB,
      col.names = c(
        "Clock (GHz)", "CPI",
        "Instructions Per Second",
        "Cycles in 10 Seconds",
        "Instructions in 10 Seconds"
        )
      ) %>%
  kable_styling(full_width = F)
```

Clock (GHz)	CPI	Instructions Per Second	Cycles in 10 Seconds	Instructions in 10 Seconds
3.0	1.5	2000000000	3.0e+10	20000000000
2.5	1.0	2500000000	2.5e+10	25000000000
4.0	2.2	1818181818	4.0e+10	18181818182

c. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

New Execution time = 70% of Old Execution time

Execution time = Instructions * (CPI / Clock Rate)

Therefore,

0.7 * Execution Time = ((1.2 * Instructions) * CPI) / Clock Rate Execution Time = (1.2 * Instructions * CPI) / 0.7 * Clock Rate

So, New Clock Rate = (1.2 / 0.7) * Clock Rate New Clock Rate = 1.7142857 Clock Rate

Hence,

Clock Rates should be increased by 1.7142857

```
cpuComparisonC <- data.frame(
  "oldClocksGhz" = cpuComparison$clockGhz,
  "newClocksGhz" = cpuComparison$clockGhz *(1.2/0.7)
  )
kable(cpuComparisonC, col.names = c(
  "Old Clocks (GHz)",
  "New Clocks (GHz)"
  )
  ) %>%
  kable_styling()
```

Old Clocks (GHz)	New Clocks (GHz)
3.0	5.142857
2.5	4.285714
4.0	6.857143

1.6 Consider two diff erent implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2. Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

We can compare the execution time for each implementation to find out which is faster.

Execution time = (Instructions * CPI)/Clock Rate

Instructions = 10^6

Lets say "Instructions" = x

Cycles for P1 = 1 * 0.1x + 2 * 0.2x + 3 * 0.5x + 3 * 0.2x = 2.6x or 2.6 * 10^6

Cycles for P2 = 0.2x = 2 * 0.1x + 2 * 0.2x + 2 * 0.5x + 2 * 0.2x = 2x or 2 * 10^6

Execution Time for P1 = 2.6 * 10^6 / 2.5*10^9 = 1.0400e-03 seconds

Execution Time for P2 = 2 * 10^6 / 3.0 * 10^9 = 6.6667e-04 seconds

Hence, P2 is faster

a. What is the global CPI for each implementation?

CPI = Time*Clock Rate / Instructions

CPI for P1 = 0.00104 * 2.5*10^9 / 10^6 = 2.6 CPI

CPI for P2 = 6.6666667×10^{-4} * 3.0*10^9 / 10^6 = 2.0 CPI

b. Find the clock cycles required in both cases.

Clock cycles = CPI * Instructions

Clock cycles for P1 = 2.6 * 10^6 cycles

Clock cycles for P2 = 2.0 * 10^6 cycles

1.7 Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of 1.0E9 and has an execution time of 1.1 s, while compiler B results in a dynamic instruction count of 1.2E9 and an execution time of 1.5 s

a. Find the average CPI for each program given that the processor has a clock cycle time of 1 ns

Time = Instructions * CPI / Clock Rate or Instructions * CPI * Cycle Time

CPI = Time / Instructions * Cycle Time

CPI for Compiler A = 1.1 / ((1* 10^9) * 1 * 10^-9) = 1.1 CPI CPI for Compiler B = 1.5 / ((1.2 * 10^9) * 1 * 10^-9) = 1.25 CPI

b. Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

Execution Time = Instructions * CPI / Clock Rate

(10^9 * 1.1)/ ClockRateA = (1.2 * 10^9) * 1.25 / ClockRateB

ClockRateA = 1.1 / (1.2*1.25) ClockRateB

ClockRateA = 0.733 ClockRateB

Hence, processor A is actually 26.7% slower than processor B or 73.3% the speed of processor B.

c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

CPU Time for New Compiler = Instructions 3 * CPI 3 / Clock Rate

CPU Time for New Compiler = 6 * 10^8 * 1.1 * 10^-9 / Clock Rate

CPU Time for New Compiler = 0.66 / Clock Rate

SpeedUp Compiler C versus A = 1.1 / 0.66 = 1.67

SpeedUp Compiler C versus B = 1.5 / 0.66 = 2.27

Hence, Compiler C is 1.67 times faster than Compiler A while it is 2.27 times faster than Compiler B

1.9 Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of 1, 12, and 5, respectively. Also assume that on a single processor a program requires the execution of 2.56E9 arithmetic instructions, 1.28E9 load/store instructions, and 256 million branch instructions. Assume that each processor has a 2 GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by 0.7 x p (where p is the number of processors) but the number of branch instructions per processor remains the same.

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

Cycles = (2.56 * 10^9) * 1 + (1.28 * 10^9) * 12 + (256 * 10^6) * 5

Cycles = 1.92×10^{10}

Execution Time = 1.92×10^{10}/2.0 GHz = 9.6 seconds

Parallel Cycles = (2.56 * 10^9)/0.7p * 1 + (1.28 * 10^9)/0.7p * 12 + (256 * 10^6) * 5 * 1

Parallel Cycles = 2.56×10^{10}p + 1.28×10^9

Parallel execution time = 2.56×10^{10}/p/2×10^9 + 1.28×10^9/2×10^9 = 12.8/p + 0.64

1.9.1 Find the total execution time for this program on 1, 2, 4, and 8 processors, and show the relative speedup of the 2, 4, and 8 processor result relative to the single processor result.

Execution Time for 1 Processor = 9.6s

Execution Time for 2 processors = 12.8/((2/2)* 2.0) + 0.64 = 7.04s

Execution Time for 4 processors = 12.8/((4/2) * 2.0) + 0.64 = 3.84s

Execution Time for 8 processors = 12.8/((8/2) * 2.0) + 0.64 = 2.24s

```
speedupComparison = data.frame (
  "processors" = c(1,2,4,8),
  "time" = c(executionTimeP1, executionTimeP2, executionTimeP3, executionTimeP4),
  "speedup" = c(
    (executionTimeP1/executionTimeP1),
    (executionTimeP1/executionTimeP2),
    (executionTimeP1/executionTimeP3),
    (executionTimeP1/executionTimeP4)
    )
  )

kable(speedupComparison, col.names = c(
  "Processors",
  "Time (s)",
  "Speed Up"
  )
  ) %>%
  kable_styling()
```

Processors	Time (s)	Speed Up
1	9.60	1.000000
2	7.04	1.363636
4	3.84	2.500000
8	2.24	4.285714

1.9.2 If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

Cycles = (2 * 2.56 * 10^9) * 1 + (1.28 * 10^9) * 12 + (256 * 10^6) * 5

Cycles = 2.176×10^{10}

Execution Time = 2.176×10^{10}/2.0 GHz = 10.88 seconds

Parallel Cycles = (2 * 2.56 * 10^9)/0.7p * 1 + (1.28 * 10^9)/0.7p * 12 + (256 * 10^6) * 5 * 1

Parallel Cycles = 2.9257143×10^{10}p + 1.28×10^9

Parallel Execution Time = 2.9257143×10^{10}/p/2×10^9 + 1.28×10^9/2×10^9

Parallel Execution Time = 14.6285714/p + 0.64

Execution Time for 1 Processor = 10.88s

Execution Time for 2 processors = 12.8/((2/2)* 2.0) + 0.64 = 7.9542857

Execution Time for 4 processors = 14.6285714/((4/2) * 2.0) + 0.64 = 4.2971429

Execution Time for 8 processors = 14.6285714/((8/2) * 2.0) + 0.64 = 2.4685714

```
speedupComparison2 = data.frame (
  "processors" = c(1,2,4,8),
  "time" = c(
    executionTime2P1,
    executionTime2P2,
    executionTime2P3,
    executionTime2P4
    ),
  "speedup" = c(
    (executionTime2P1/executionTime2P1),
    (executionTime2P1/executionTime2P2),
    (executionTime2P1/executionTime2P3),
    (executionTime2P1/executionTime2P4)
    )
  )

kable(speedupComparison2, col.names = c(
  "Processors",
  "Time (s)",
  "Speed Up")
  ) %>%
  kable_styling()
```

Processors	Time (s)	Speed Up
1	10.880000	1.000000
2	7.954286	1.367816
4	4.297143	2.531915
8	2.468571	4.407407

Surprisingly, the impact of doubling the instructions isn't significant and only seems to have some impact in speedup at 8 processors.

1.9.3 To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

For this to be true, we must assume the execution time of 1 processor to be equivalent to that of 4 processors.

New Execution time for 1 processor = 3.84s

= New Clock Cycles / 2.0 Ghz = 3.84s

= New Clock Cycles = 7.68×10^9

Hence,

New CPI = (2.56 * 10^9) * 1 + (1.28 * 10^9) * New CPI + (256 * 10^6) * 5 = 7.68×10^9 CPI

3.84×10^9 + 1.28×10^9new CPI = 7.68×10^9

new CPI = 3.84×10^9/(1.28 * 10^9)

new CPI for Load/Store = 3 CPI

Hence, the CPI for load/store should be reduced to 3 from 12 for 1 processor to match the performance of 4 processors.

1.11 The results of the SPEC CPU2006 bzip2 benchmark running on an AMD Barcelona has an instruction count of 2.389E12, an execution time of 750 s, and a reference time of 9650 s.

1.11.1 Find the CPI if the clock cycle time is 0.333 ns.

Execution time = (Instructions * CPI) * Clock Cycle or Cycles * Clock Cycle

CPI = Execution Time/(Clock Cycle * Instructions)

CPI = 750 / ((0.333 * 10^-9) * (2.389 * 10^12))

CPI = 0.9427594 CPI

1.11.2 Find the SPECratio.

SPEC ratio = Reference time / Execution Time

SPEC ratio = 9650 / 750 = 12.8666667

1.11.3 Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% without affecting the CPI.

Execution Time (CPU Time) = 1.1 * (2.389 * 10^12) * 0.94 * (0.333 * 10^-9)

Execution Time (CPU Time) = 822.585258 seconds

1.11.4 Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 5%.

Execution Time (CPU Time) = 1.1 * (2.389 * 10^12) * 1.05 * 0.94 * (0.333 * 10^-9)

Execution Time (CPU Time) = 863.7145209 seconds

1.11.5 Find the change in the SPECratio for this change.

SPEC ratio = Reference time / Execution Time

New Spec Ratio = 9650 / 863.7145209 = 11.1726731

Change in Spec Ratio = 12.8666667 - 11.1726731 = 1.6939936

Hence the new SPEC ratio is 11.1726731 and the change from original SPEC ratio is 1.6939936

1.11.6 Suppose that we are developing a new version of the AMD Barcelona processor with a 4 GHz clock rate. We have added some additional instructions to the instruction set in such a way that the number of instructions has been reduced by 15%. Th e execution time is reduced to 700 s and the new SPECratio is 13.7. Find the new CPI.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

CPI = (Execution Time * Clock Rate)/ Instructions

CPI = (700 * 4 * 10^9) / (0.85 * 2.389 * 10^12)

CPI = 1.3788688 CPI

1.11.7 This CPI value is larger than obtained in 1.11.1 as the clock rate was increased from 3 GHz to 4 GHz. Determine whether the increase in the CPI is similar to that of the clock rate. If they are dissimilar, why?

Original Clock Rate = 1/0.333 * 10^-9 = 3.003003×10^9 or 3 GHz

Clock Rate Increase Ratio = 4 GHz/3 GHz = 1.3333333

CPI Increase ratio = 1.3788688 / 0.9427594 = 1.4625882

Hence, the increase in CPI is dissimilar to that of Clock Rate because we also reduced instructions by 15% for the new CPI.

1.11.8 By how much has the CPU time been reduced?

CPU Time Decrease Ratio = 700 / 750 = 0.933 or 93.3%

Hence, the CPU Time has decreased by 6.67%

1.11.9 For a second benchmark, libquantum, assume an execution time of 960 ns, CPI of 1.61, and clock rate of 3 GHz. If the execution time is reduced by an additional 10% without affecting to the CPI and with a clock rate of 4 GHz, determine the number of instructions.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Instructions = (Execution Time * Clock Rate)/ CPI

Instructions = (0.9 * 960 * 10^-9 * 4 * 10^9) / 1.61 = 2146.5838509

1.11.10 Determine the clock rate required to give a further 10% reduction in CPU time while maintaining the number of instructions and with the CPI unchanged.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Clock Rate = (Instructions * CPI) / Execution Time

Clock Rate = (2146.5838509 * 1.61) / 0.9 * (0.9 * 960)

Clock Rate = 4.4444444 GHz

1.11.11 Determine the clock rate if the CPI is reduced by 15% and the CPU time by 20% while the number of instructions is unchanged.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Clock Rate = (Instructions * CPI) / Execution Time

Clock Rate = (2146.5838509 * (0.85 * 1.61)) / (0.8 * (0.9 * (0.9 * 960))

Clock Rate = 4.7222222 GHz

1.12 Section 1.10 cites as a pitfall the utilization of a subset of the performance equation as a performance metric. To illustrate this, consider the following two processors. P1 has a clock rate of 4 GHz, average CPI of 0.9, and requires the execution of 5.0E9 instructions. P2 has a clock rate of 3 GHz, an average CPI of 0.75, and requires the execution of 1.0E9 instructions.

1.12.1 One usual fallacy is to consider the computer with the largest clock rate as having the largest performance. Check if this is true for P1 and P2.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Execution Time for P1 = (5 * 10^9 * 0.9) / (4.0*10^9) = 1.125s

Execution Time for P2 = (1 * 10^9 * 0.75) / (3.0*10^9) = 0.25s

This fallacy is false, as the term implies. P2 is quite a lot faster than P1.

1.12.2 Another fallacy is to consider that the processor executing the largest number of instructions will need a larger CPU time. Considering that processor P1 is executing a sequence of 1.0E9 instructions and that the CPI of processors P1 and P2 do not change, determine the number of instructions that P2 can execute in the same time that P1 needs to execute 1.0E9 instructions.

Execution time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Execution Time for P1 = (1 * 10^9 * 0.9) / (4.0*10^9) = 0.225s

Instructions for P2 with P1 execution time = (0.225 * (3.0*10^9))/0.75

$= 9 \times 10^8$ instructions

1.12.3 A common fallacy is to use MIPS (millions of instructions per second) to compare the performance of two different processors, and consider that the processor with the largest MIPS has the largest performance. Check if this is true for P1 and P2.

MIPS = Clock Rate / (CPI * 10^6)

MIPS for P1 = (4 * 10^9) / (0.9 * 10^6) = 4444.4444444

MIPS for P2 = (3 * 10^9) / (0.75 * 10^6) = 4000

Yes, its a fallacy as P2 is faster in reality while the MIPS metric implies that P1 is faster.

1.12.4 Another common performance figure is MFLOPS (millions of floating-point operations per second), defined as MFLOPS = No. FP operations / (execution time x 1E6) but this figure has the same problems as MIPS. Assume that 40% of the instructions executed on both P1 and P2 are floating-point instructions. Find the MFLOPS figures for the programs.

MFLOPS = FLOP Instructions / (Time * Million)

MFLOPS for P1 = (0.4 * 5 * 10^9) / (1.125* 10^6) = 1777.7777778

MFLOPS for P2 = (0.4 * 1 * 10^9) / (0.25* 10^6) = 1600

Even the MFLOPS test seems to favor P1 while P2 is actually faster, hence MFLOPS is a fallacy.

1.13 Another pitfall cited is expecting to improve the overall performance of a computer by improving only one aspect of the computer. Consider a computer running a program that requires 250 s, with 70 s spent executing FP instructions, 85 s executed L/S instructions, and 40 s spent executing branch instructions.

1.13.1 By how much is the total time reduced if the time for FP operations is reduced by 20%?

Total Time reduced with 80% FP time = 70 - (0.8 * 70) = 14

Hence the total time will be reduced by 14 seconds bringing it to 236 seconds or by 5.6%.

1.13.2 By how much is the time for INT operations reduced if the total time is reduced by 20%?

Total reduced by 80% = 0.8 * 250 = 200

Assuming the remaining time after taking out FP, L/S and branch instructions is used for INT operations, then

Old time spent on INT operations = 250 - 70 - 40 - 85 = 55

New time spent on INT operations = 200 - 70 - 40 - 85 = 5

Hence the time spent on INT operations will be reduced by 50 seconds or by 90.9090909%.

1.13.3 Can the total time can be reduced by 20% by reducing only the time for branch instructions?

Assume we use minimal branch instructions or none, lets find the maximum time saved and see if the savings fall under the 20% threshold

Total time with minimal branch instructions = FP Time + L/S Time + INT Time

= 70 + 85 + 55 = 210

Percentage of savings = (1 - (210 / 250)) * 100 = 16%

Hence, no, the total time can only be reduced by a maximum of 18% by reducing branch instructions and not 20%

1.14 Assume a program requires the execution of 50 x 106 FP instructions, 110 x 106 INT instructions, 80 x 106 L/S instructions, and 16 x 106 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

Execution Time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Total Cycles = Instructions * CPI

Total Cycles = (50 * 106 * 1) + (110 * 106 * 1) + (80 * 106 * 4) + (16 * 106 * 2) = 5.4272×10^4

Execution Time = 5.4272×10^4 / (2 * 10^9) = 2.7136×10^{-5} s or 27.136us

1.14.1 By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

New Execution Time = Execution Time / 2 = 1.3568×10^{-5}

Execution Time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

New Total Cycles = (50 * 106 * NewCPI) + (110 * 106 * 1) + (80 * 106 * 4) + (16 * 106 * 2)

New Total Cycles = 5300 * NewCPI + 4.8972×10^4

Execution Time = Cycles / Clock Rate

Cycles = Execution Time * Clock Rate

5300 * newCPI + 4.8972×10^4 = 1.3568×10^{-5} * (2 * 10^9)

newCPI = (2.7136×10^4 - 4.8972×10^4) / 5300

newCPI = -4.12 CPI

It is impossible to run the program twice as fast by only improving FP instructions CPI as shown by the negative result.

1.14.2 By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

New Execution Time = Execution Time / 2 = 1.3568×10^{-5}

Execution Time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

New Total Cycles = (50 * 106 * 1) + (110 * 106 * 1) + (80 * 106 * newCPI) + (16 * 106 * 2)

New Total Cycles = 2.0352×10^4 + 8480 * newCPI

Execution Time = Cycles / Clock Rate

Cycles = Execution Time * Clock Rate

8480 * newCPI + 2.0352×10^4 = 1.3568×10^{-5} * (2 * 10^9)

newCPI = (2.7136×10^4 - 2.0352×10^4) / 8480

newCPI = 0.8 CPI

Hence L/S instructions must be improved to decrease CPI from 4 to 0.8 or reduced by 80%

1.14.3 By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

Execution Time = (Instructions * CPI) / Clock Rate or Cycles / Clock Rate

Total Cycles = Instructions * CPI

Total Cycles = (0.6 * 50 * 106 * 1) + (0.6 * 110 * 106 * 1) + (0.7 * 80 * 106 * 4) + (0.7 * 16 * 106 * 2) = 3.62944×10^4

Execution Time = 3.62944×10^4 / (2 * 10^9) = 1.81472×10^{-5} s or 18.1472us

Hence the execution time improved from 2.7136×10^{-5} to 1.81472×10^{-5} or by 33.125%

1.15 When a program is adapted to run on multiple processors in a multiprocessor system, the execution time on each processor is comprised of computing time and the overhead time required for locked critical sections and/or to send data from one processor to another. Assume a program requires t = 100 s of execution time on one processor. When run p processors, each processor requires t/p s, as well as an additional 4 s of overhead, irrespective of the number of processors. Compute the per-processor execution time for 2, 4, 8, 16, 32, 64, and 128 processors. For each case, list the corresponding speedup relative to a single processor and the ratio between actual speedup versus ideal speedup (speedup if there was no overhead).

```
time <- 100
overhead <- 4
processors <- c(2, 4, 8, 16, 32, 64, 128)

comparison <- data.frame(processors)

# Compute Time For Each Processor (Divide time by number of processors)
comparison$timeInSeconds <- (time/comparison$processors) + overhead

# Compute Speed Up (Find ratio of time of each processor with original time)
comparison$speedUpPercentage <- (1-(comparison$timeInSeconds/time))*100

# Compute Ideal Time For Each Processor
comparison$idealTimeInSeconds <- time/comparison$processors

# Compute Ideal Speed Up
comparison$idealSpeedUpPercentage <- (1-(comparison$idealTimeInSeconds/time))*100

# Compute Speed Up vs Ideal Speed Up
comparison$actualVSIdealSpeedUp <-
  (comparison$speedUpPercentage/comparison$idealSpeedUpPercentage)

# Draw Table with clean column names
kable(comparison, col.names = c(
  "Processors",
  "Time (s)",
  "Speed Up (%)",
  "Ideal Time (s)",
  "Ideal Speed Up (%)",
  "Actual vs Ideal Speed Up Ratio")
  ) %>%
  kable_styling()
```

Processors	Time (s)	Speed Up (%)	Ideal Time (s)	Ideal Speed Up (%)	Actual vs Ideal Speed Up Ratio
2	54.00000	46.00000	50.00000	50.00000	0.9200000
4	29.00000	71.00000	25.00000	75.00000	0.9466667
8	16.50000	83.50000	12.50000	87.50000	0.9542857
16	10.25000	89.75000	6.25000	93.75000	0.9573333
32	7.12500	92.87500	3.12500	96.87500	0.9587097
64	5.56250	94.43750	1.56250	98.43750	0.9593651
128	4.78125	95.21875	0.78125	99.21875	0.9596850