NYU Tandon School of Engineering

Fall 2023, ECE 6913

Homework Assignment 1 Solutions

1. Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.0 GHz and CPIs of 1, 2, 2, and 1, and P2 with a clock rate of 4 GHz and CPIs of 2, 3, 4, and 4.

a. 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 is faster: P1 or P2 (in total execution time)?

	P1	P2	Benchmark
Instruction Class	CPI	CPI	
A	1	2	10%
B	2	3	20%
C	2	4	50%
D	1	4	20%
Average CPI	1.7	3.6	

P1: 2GHz

P2: 4GHz

IC = 10^6 instructions

Execution Time = IC x Cycle Time x CPI

Execution Time P1 = 10^6 x 0.5ns x 1.7 = 8.5 x 10^{-4} (secs)

Execution Time P2 = 10^6 x 0.25ns x 3.6 = 9 x 10^{-4} (secs)

P1 is faster

b. What is the global CPI for each implementation?

CPI_{P1} = 1.7, CPI_{P2}=3.6

c. Find the clock cycles required in both cases.

#Cycles = Execution Time / cycle time

Cycles for P1 = 8.5 x 10^{-4}s /0.5ns = 1.70 x 10^6 cycles

Cycles for P2 = 9 x 10^{-4}s /0.25ns = 3.60 x 10^6 cycles

d. Which processor has the highest throughput performance (instructions per second) ?

IPS = CPS/CPI = Fclk/CPI = 2GHz/1.7 **(P1) = 1.176 x 10^9** and 4GHz/3.6**(P2) = 1.111 x 10^9**

e. Which processor do you think is more energy efficient? Why?

P1 completes the same benchmark in less time with fewer clock cycles

2. You are designing a system for a real-time application in which specific deadlines must be met. Finishing the computation faster gains nothing. You find that your system can execute the necessary code, in the worst case, twice as fast as necessary.

a. *How much energy do you save if you execute at the current speed and turn off the system when the computation is complete?*

Energy is dependent on the number of logic transitions in the system each of which cost $E = CV_{dd}^2$. More relevantly, energy consumed is independent of the speed at which these logic transitions complete. So, finishing the computation sooner does not benefit energy reduction.
So, how much energy is saved? **None**

b. *How much energy do you save if you set the voltage and frequency to be half as much?*

Setting Voltage to half its nominal value lowers energy to ¼ its original value given the quadratic dependence of Energy on operating voltage as we saw in 1a above. Setting frequency to half its nominal value lowers power by half but the energy consumption remains unchanged. **So setting voltage and frequency to half their nominal values lowers Energy to ¼ its nominal value or by 75%** and **Power to 1/8 its nominal value or by 87.5%**

3. Server farms such as Google and Yahoo! provide enough compute capacity for the highest request rate of the day. Imagine that most of the time these servers operate at only 60% capacity. Assume further that the power does not scale linearly with the load; that is, when the servers are operating at 60% capacity, they consume 90% of maximum power. The servers could be turned off, but they would take too long to restart in response to more load. A new system has been proposed that allows for a quick restart but requires 20% of the maximum power while in this "barely alive" state.

a. *How much power savings would be achieved by turning off 60% of the servers?*

Assuming a uniform distribution of load across all servers and assuming the load does not change in each server when turning off any fraction of the servers, 60% of servers turned off, **reduces power by 60%**

b. *How much power savings would be achieved by placing 60% of the servers in the "barely alive" state?*

40% of the servers would consume power that is unchanged. 60% of the servers would drop their power to 20% of maximum (instead of 90%).
So, total power consumption now is:
0.4 x PN + 0.6 x (0.2/0.9) x PN =
[0.4 + 0.133] PN
= 0.533 PN or **reduction to 53.3%** of nominal power consumption (PN)

c. *How much power savings would be achieved by reducing the voltage by 20% and frequency by 40%?*

CVV fclk [new] = C (0.8V)(0.8V)(0.6fclk)) = 0.384 x CVV (old)
savings of 61.6% in power achieved

d. *How much power savings would be achieved by placing 30% of the servers in the "barely alive" state and 30% off?*

40% of the servers would consume power that is unchanged.
30% are off
30% are in the 'barely alive' state

0.4 x PN + 0.3 x 0 x PN + 0.3 x (0.2/0.9) PN = 0.4666 PN or **reduction to 46.67% of nominal power**

4. In a server farm such as that used by Amazon or eBay, a single failure does not cause the entire system to crash. Instead, it will reduce the number of requests that can be satisfied at any one time.

a. *If a company has 10,000 computers, each with a MTTF of 35 days, and it experiences catastrophic failure only if 1/3 of the computers fail, what is the MTTF for the system?*

$MTTF_{computer}$ = 35 days => FITcomputer = (1/35) per day
for 1 of 3 computers to fail, FITsystem = 3(1/35) failures/day
MTTFsystem = 1/FITsystem = (35/3) days = 11.67 days

b. *If it costs an extra $1000, per computer, to double the MTTF, would this be a good business decision? Show your work.*
Cost to double MTTF = $1000 per computer or $ 10M for system. So if a computer costs more than $1000 its cheaper to simply double the MTTF rather than replace the system

5. In this exercise, assume that we are considering enhancing a machine by adding vector hardware to it. When a computation is run in vector mode on the vector hardware, it is 10 times faster than the normal mode of execution. We call the percentage of time that could be spent using vector mode the *percentage of vectorization.* Vectors are discussed in Chapter 4, but you don't need to know anything about how they work to answer this question!

a. *Draw a graph that plots the speedup as a percentage of the computation performed in vector mode. Label the y-axis "Net speedup" and label the x-axis "Percent vectorization."*

b. *What percentage of vectorization is needed to achieve a speedup of 2?*

Assume T_0 is the time taken for the non vectorized code to run.

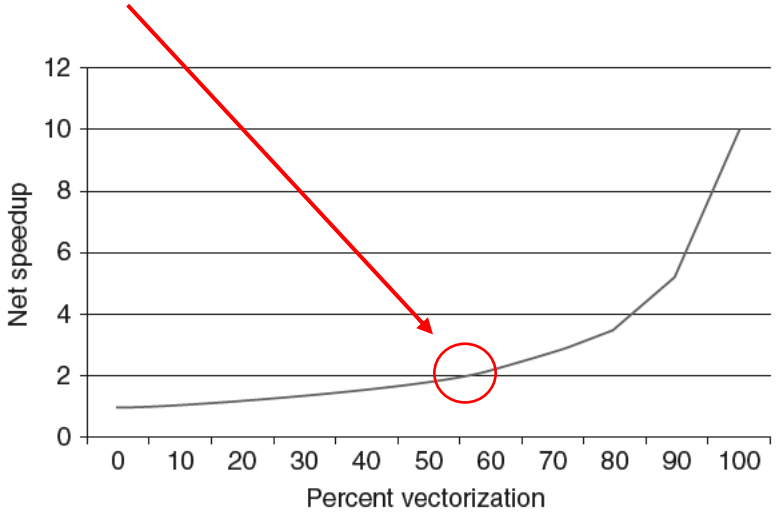
assume 'x' equals the percentage of the code vectorized. This piece takes 1/10 the time to run as the non-vectorized code. So, `[100-x]%` of the code is not vectorized.

total time to run code with x % vectorized = `[100 - x]% T0 + [x/10]%T0`
if `x=0%,` then the code takes 100% of T_0 time to execute
if `x=100%,` then code takes 10% of T_0 time to execute

`Speedup = 100%T0 /{[100 - x]% T0 + [x/10]%T0 }`
for speedup of 2 (code runs twice as fast, that is, takes half as long

`2 = 100%T0 /{[100 - x]% T0 + [x/10]%T0 }`
`2 = 100 /{[100 - x] + [x/10] }`

So, **`x = 55.55%`**



`Plot of the equation: y = 100/[(100-x)+x/10]`

c. *What percentage of the computation run time is spent in vector mode if a speedup of 2 is achieved?*

`[x/10] / [ (x/10)+ (100-x) ] = 5.55/[ 5.55 + 44.45 ]`
`= 5.55/50`
`= 0.111 x 100 = 11.11%`

d. *What percentage of vectorization is needed to achieve one-half the maximum speedup attainable from using vector mode?*

maximum speedup from using vector mode = 10x

half of maximum speedup = 5x

$5 = 100\%T_0 / \{[100 - x]\% T_0 + [x/10]\%T_0 \}$

So, **x = 88.88%**

i.e., 88.88% of code must be vectorized for speedup to be half of maximum:
that is =1/2 x 10 = 5

e. *Suppose you have measured the percentage of vectorization of the program to be 70%. The hardware design group estimates it can speed up the vector hardware even more with significant additional investment. You wonder whether the compiler crew could increase the percentage of vectorization, instead. What percentage of vectorization would the compiler team need to achieve in order to equal an addition 2× speedup in the vector unit (beyond the initial 10×)?*

with a 70% vectorization, speedup would be 100/[30 + 7] = 2.7x

we have a choice of either
(i) The *hardware design group* increasing the *speedup of the vector hardware to more than 10X* with significant investment, or
(ii) The *compiler group* increasing the speedup with a higher percentage vectorization of the code

For (ii) above, if we want to double the speedup accomplished so far of 2.7x with a 70% vectorization, to 5.4x what % vectorization 'y' would be necessary?

1.4 = 100 / [100 – y + y/10]
y = 90.53% of the code would need to be vectorized by the compiler group (instead of 70%) to get a 2x improvement over the 2.7x speedup already accomplished with 70% vectorization

6. **a.** A program (or a program task) takes 150 million instructions to execute on a processor running at 2.7 GHz. Suppose that 50% of the instructions execute in 3 clock cycles, 30% execute in 4 clock cycles, and 20% execute in 5 clock cycles. What is the execution time for the program or task?

Inst Count = 150M, F_{CLK} = 2.7GHz, Cycle Time = 1/ [2.7GHz] = 0.37037 ns

	Percentage of code	CPI	Instruction Count	# of cycles	Exec Time
Instruction 1	50%	3	75M	225M	83.33 ms
Instruction 2	30%	4	45M	180M	66.66 ms
Instruction 3	20%	5	30M	150M	55.55 ms

Execution time = CPI x IC x Cycle Time summed over each instruction
= {83.33 + 66.66 + 55.55} ms = **205.54 ms**

b. Suppose the processor in the previous question part is redesigned so that all instructions that initially executed in 5 cycles and all instructions executed in 4 cycles now execute in 2 cycles. Due to changes in the circuitry, the clock rate also must be decreased from 2.7 GHz to 2.1 GHz. What is the overall percentage improvement?

Inst Count = 150M, F_{CLK} = **2.1GHz**, Cycle Time = 1/ [2.1GHz] = **0.47619 ns**

	Percentage of code	CPI	Instruction Count	# of cycles	Exec Time
Instruction 1	50%	3	75M	225M	107.14 ms
Instruction 2	30%	2	45M	90M	42.86 ms
Instruction 3	20%	2	30M	60M	28.57 ms

Execution time = CPI x IC x Cycle Time summed over each instruction
= {107.14 + 42.86 + 28.57} ms = **178.57 ms**

Lowering the CPI for half of the instructions even though it required a longer cycle time paid off since these improved instructions can be considered the 'common case'
So, overall speedup = 205.54ms/178.57ms = 1.151 or a 15.1% improvement

7. Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

a. Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?

	CPI	# Instrs. (old)	# Inst. (new)	Exec Time old	Exec Time new
Load/Store	10	300M	same	3000M x Tcycle	3000M
Branch	3	100M	same	300M x Tcycle	300M
Arithmetic	1	500M	375M	500M x Tcycle	375M

Old Execution time = CPI x IC x Cycle Time summed over each instruction
= **{3800M x Tcycle}**
New Execution time = 3675M x Tcycle x 1.1 = **4042.5M x Tcycle**
New execution time is larger, so this is a bad design choice
speedup = 3800/4042.5 = 0.94 or 6% degradation

b. Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

	CPI	# Instrs. (old)	# Inst. (new)	Exec Time old	Exec Time old
Load/Store	10	300M	3000M	3000M x Tcycle	
Branch	3	100M	300M	300M x Tcycle	
Arithmetic	0.5	500M	250M	250M x Tcycle	

New execution time by doubling the performance of arithmetic instructions (half the original CPI) = 3550M x Tcycle; Speedup = 3800M/3550M = 1.0704 => **7.04%**

New execution time by improving performance of arithmetic instructions by 10X (1/10 of original CPI)

= 3350M x Tcycle; speedup = 3800M/3350M = 1.134 => **13.4% improvement**

8. After graduating, you are asked to become the lead computer designer at Hyper Computers, Inc. Your study of usage of high-level language constructs suggests that procedure calls are one of the most expensive operations. You have invented a scheme that reduces the loads and stores normally associated with procedure calls and returns. The first thing you do is run some experiments with and without this optimization. Your experiments use the same state-of-the-art optimizing compiler that will be used with either version of the computer. These experiments reveal the following information:

- The clock rate of the unoptimized version is 5% higher.
- 30% of the instructions in the unoptimized version are loads or stores.
- The optimized version executes 2/3 as many loads and stores as the unoptimized version. For all other instructions the dynamic counts are unchanged.
- All instructions (including load and store) take one clock cycle.

Which is faster? Justify your decision quantitatively.

CPU performance equation:

$$\text{Execution Time} = IC * CPI * Tcycle$$

The unoptimized version is 5% faster in clock cycle time

$$Tcycle_{unop} = 0.95 * Tcycle_{op}$$

Instruction Count (IC) of load/store instructions are 30% of total IC in the unoptimized version

$$IC_{ld/st\_unop} = 0.3 * IC_{unop}$$

IC of load store instructions in optimized version is 0.67 of IC of load/store instructions in unoptimized version

$$IC_{ld/st\_op} = 0.67 * IC_{ld/st\_unop}$$

IC of all other (non load/store) instructions in optimized version = IC of all other (non load/store) instructions in unoptimized version

$$IC_{others\_unop} = IC_{others\_op}$$

CPI = 1

$$\text{Execution Time}_{unop} = IC_{unop} * Tcycle_{unop}$$
$$= 0.95 * IC_{unop} * Tcycle_{op}$$

$$\text{Execution Time}_{op} = IC_{op} * Tcycle_{op}$$

$$IC_{op} = IC_{others\_op} + IC_{ld/st\_op}$$
$$IC_{op} = IC_{others\_unop} + IC_{ld/st\_op}$$
$$IC_{op} = 0.7 * IC_{unop} + 0.67 * IC_{ld/st\_unop}$$
$$IC_{op} = 0.7 * IC_{unop} + 0.67 * (0.3 * IC_{unop}) = 0.7 * IC_{unop} + 0.2*IC_{unop}$$
$$= 0.9 * IC_{unop}$$

```
Execution Time_op = IC_op * Tcycle_op
   = 0.9 * IC_unop * 1.05 * Tcycle_unop

   = 0.945 * IC_unop * Tcycle_unop
```

Improvement of 5.5%

9. General-purpose processes are optimized for general-purpose computing. That is, they are optimized for behavior that is generally found across a large number of applications. However, once the domain is restricted somewhat, the behavior that is found across a large number of the target applications may be different from general-purpose applications. One such application is deep learning or neural networks. Deep learning can be applied to many different applications, but the fundamental building block of inference—using the learned information to make decisions—is the same across them all. Inference operations are largely parallel, so they are currently performed on graphics processing units, which are specialized more toward this type of computation, and not to inference in particular. In a quest for more performance per watt, Google has created a custom chip using tensor processing units to accelerate inference operations in deep learning.1 This approach can be used for speech recognition and image recognition, for example. This problem explores the trade-offs between this process, a general-purpose processor (Haswell E5-2699 v3) and a GPU (NVIDIA K80), in terms of performance and cooling. If heat is not removed from the computer efficiently, the fans will blow hot air back onto the computer, not cold air. Note: The differences are more than processor—on-chip memory and DRAM also come into play. Therefore statistics are at a system level, not a chip level. **9 a:** Performance gain obtained by improving some portion of the computer speed

= (Execution Time for entire task without using the enhancement) / (Execution Time for entire task using the enhancement)

Speedup of computer A over B = Execution Time of B / Execution Time of A – (1)

From Table 2:

Speedup of TPU over GPU for workload A = 225000/13461 = 16.7 – (2)

Speedup of TPU over GPU for workload B = 280000/36465 = 7.7 – (3)

From (1 –3), considering 70% of a GPUs Execution Time is for workload A, 30% for B

0.7 x Ex Time (A+B) of GPU = Ex Time (A) of TPU x 16.7 (for workload A)

0.3 x Ex Time (A+B) of GPU = Ex Time (B) of TPU x 7.7 (for workload B)

So, Ex Time (A+B) of TPU

= [0.7 x Ex Time (A+B) of GPU /16.7 + 0.3 x Ex Time (A+B) of GPU/7.7]

Ex Time (A+B) TPU = [0.7/16.7 + 0.3/7.7] Ex Time (A+B) GPU

So, [Ex Time (A+B) GPU / Ex Time (A+B) TPU] = speedup of TPU over GPU = [from (1)

above]

= 1/ [0.7/16.7 + 0.3/7.7] = 1/[0.0419 + .03896] = 12.37

9 b: IPS [Instructions/sec] = Clock rate [cycles/sec] / CPI [cycles/Instruction]

Datacenter spends 70% of its time in workload A & 30% of its time in workload B

The general purpose CPU can accomplish only 42% of its maximum IPS in workload A and 100% of its maximum IPS in workload B

so, maximum IPS it can possibly achieve = 42% x 70% + 100% x 30% = 0.294 + 0.3 = 0.594

In **GPU** maximum IPS in workload A, B = 37%, 100%

so, 37% x 70% + 100% x 30% = 0.37 * 0.7 + 1 * 0.3 = **0.559**

In Custom ASIC (**TPU**), maximum IPS in workload A, B = 80%, 100% so, 0.8 x 0.7 + 1 x 0.3 = **0.86**

9c: Linear dependence of power on IPS

at max IPS, processor is consuming 'busy' power

at 0 IPS, processor is consuming idle power

Idle power + [max power –idle power] x [% of max IPS] = Power

CPU: 159 W + (455 W-159 W) x0.594= 335 W

GPU: 357 W + (991 W-357 W) x0.559= **711 W**

TPU: 290 W + (384 W-290 W) x0.86= **371 W**

Ratio of performance per watt (PPW) of system1 to system 2

= [% max IPS/Power]1 / [% of max IPS/Power]2

for TPU:GPU comparison

PPW Ratio of TPU to GPU = **0.86** x **711** / **0.559** x **371** = 2.95

The TPU delivers almost **3X higher performance per watt** for this workload

9 d: Another data center spends 40% of its time on workload A, 10% of its time on workload B, and

50% of its time on workload C

Speedup of GPU over CPU for workload A = 13461 / 5482

Speedup of GPU over CPU for workload B = 36465 / 13194

Speedup of GPU over CPU for workload C = 15000 / 12000

So, **Speedup of GPU over general-purpose CPU** is

$$S = \frac{1}{\frac{0.4}{13461/5482} + \frac{0.1}{36465/13194} + \frac{0.5}{15000/12000}} = 1.669$$

Speedup of TPU over CPU for workload A = 225000 / 5482

Speedup of TPU over CPU for workload B = 280000 / 13194

Speedup of TPU over CPU for workload C = 2000 / 12000

So, **Speedup of TPU over general-purpose CPU** is:

$$S = \frac{1}{\frac{0.4}{225000/5482} + \frac{0.1}{280000/13194} + \frac{0.5}{2000/12000}} = 0.332$$

9e.

TDP or Thermal Design Power is higher than the maximum power dissipation from a processor and thus equal to the minimum rate of heat removal that must be exceeded by cooling so that junction temperatures in the chip do not exceed their max spec. T

A 'cooling door' removes 14kW of power from a rack holding multiple server boards and are expensive ($4K)

With a cooling door, 14kW/504W = # of CPUs that can be cooled = 27

With a cooling door, 14kW/1838W = # of GPUs that can be cooled = 7

With a cooling door, 14kW/861W = # of TPUs that can be cooled = 16

9f:

Maximum power per rack in a server farm: 200 W/ft2 x 11 ft2= 2200 W

Maximum number of servers per rack & number of cooling doors for this maximum:

CPU: 2200W / 504W = 4.37; 4 servers;

GPU: 2200W / 1838W = 1.19; 1 server;

TPU: 2200W / 861W = 2.55; 2 servers;

One cooling door is enough to dissipate 2200W

System	Chip	TDP	Idle power	Busy power
General-purpose	Haswell E5-2699 v3	504 W	159 W	455 W
Graphics processor	NVIDIA K80	1838 W	357 W	991 W
Custom ASIC	TPU	861 W	290 W	384 W

Figure 1.27 Hardware characteristics for general-purpose processor, graphical processing unit-based or custom ASIC-based system, including measured power

System	Chip	Throughput			% Max IPS		
		A	B	C	A	B	C
General-purpose	Haswell E5-2699 v3	5482	13,194	12,000	42%	100%	90%
Graphics processor	NVIDIA K80	13,461	36,465	15,000	37%	100%	40%
Custom ASIC	TPU	225,000	280,000	2000	80%	100%	1%

Figure 1.28 Performance characteristics for general-purpose processor, graphical processing unit-based or custom ASIC-based system on two neural-net workloads