

COMPUTER ARCHITECTURE AND ORGANIZATION

CLASS 1:

-HW:

WATCH THE VIDEO: done

-IN CLASS:

Matrix multiply in Python write a summary how they can improve the speed of execution:

Python will take 5 minutes to run if the matrix is 960x960. But if the matrix is 4096x4096 then the processing time increases to about 6 hours.

There are multiple methods that can increase the speed at which the python matrix multiply application is executed.

First of all, the python version will run 200 times quicker if it is converted to C. By adopting this method, the speed-up of the standard Python, which is 1, will increase to roughly 175. Then, if we employ data-level parallelism, it will reach 1365 almost 8 times faster. Combining instruction level parallelism will boost execution performance by an additional factor of roughly 2, making the speedup to be about 2,457.

Additionally, applying memory hierarchy optimization will boost performance by around 1.5 times, resulting in a speedup of 3,686.

Using thread level parallelism in addition to the other ways will boost performance by approximately 12 to 17 times and accelerate things by about 44,226.

CLASS 2:

-HW:

3.3 GHz is the base clock speed and 80 watts is the TDP of the server-grade Intel Xeon E3-1225 v5 processor. It receives an 8,843 on CPU benchmark.

2.3 GHz is the base clock speed and 28 watts is the TDP of the Intel Core i5-8259U notebook processor. It received a score of 5,964 on the CPU benchmark.

Having a basic clock speed of 2.6 GHz and a TDP of 65 watts, the Intel Core i5-11400 is a desktop processor. It received a 7,837 rating on CPU Benchmark.

According to the benchmark results, of the three CPUs, the Intel Xeon E3-1225 v5 performs the best, followed by the Intel Core i5-11400, and then the Intel The Intel Xeon E3-1225 v5, Intel Core i5-11400, and Intel Core 15-8259U are the three CPUs with the highest performance according to the benchmark results. The Xeon will, however, use more power than the other two CPUs because it has the highest TDP.

I'd probably select the Intel Core i5-11400 if I had to pick one of these for my computer.

Even while it is the second-best performer out of the three, it has a more moderate TDP, making it a more energy-efficient choice. Furthermore, it is a desktop CPU, making it more suitable for usage in a computer other than a laptop.

-IN CLASS:

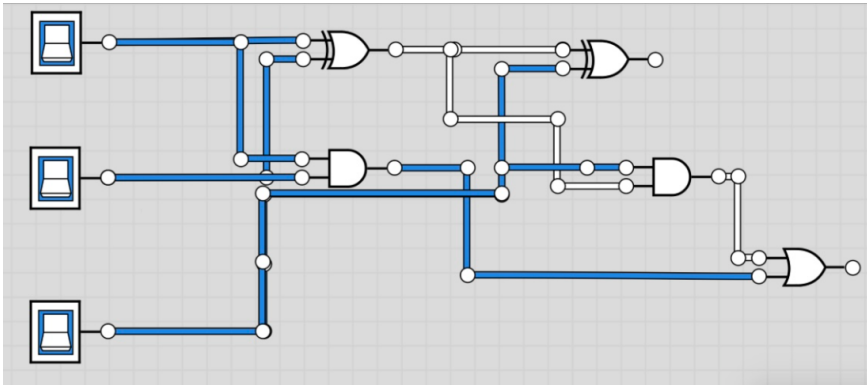
The screenshot displays the Venus Simulator interface, which is used for executing and debugging assembly code. The interface is divided into several sections:

- Top Bar:** Contains tabs for "Venus", "Editor", "Simulator" (which is active), and "Chocopy".
- Control Buttons:** A row of buttons including "Run", "Step", "Prev", "Reset", "Dump", "Trace", and "Re-assemble from Editor".
- Assembly Table:** A table with four columns: "PC", "Machine Code", "Basic Code", and "Original Code". It lists five instructions:

PC	Machine Code	Basic Code	Original Code
0x0	0x00130313	addi x6 x6 1	addi x6,x6,1
0x4	0x00238393	addi x7 x7 2	addi x7,x7,2
0x8	0x00340413	addi x8 x8 3	addi x8,x8,3
0xc	0x005304B3	add x9 x6 x5	add x9,x6,x5
0x10	0x00848533	add x10 x9 x8	add x10,x9,x8
- Registers:** A vertical list of registers on the right side, including gp (x3), tp (x4), t0 (x5), t1 (x6), t2 (x7), s0 (x8), s1 (x9), a0 (x10), a1 (x11), and a2. Each register has a corresponding input field showing its current value in decimal.
- Console Output:** A text area at the bottom left labeled "console output" for displaying program execution logs.
- Display Settings:** A dropdown menu at the bottom right set to "Decimal".

CLASS 3:

-IN CLASS:



-HW:

Venus Editor Simulator Chocopy

Run Step Prev Reset Dump Trace Re-assemble from Editor

PC	Machine Code	Basic Code	Original Code
0x0	0x00000193	addi x3 x0 0	addi x3,x0,0 # s = 0
0x4	0x00000213	addi x4 x0 0	addi x4,x0,0 # i = 0
0x8	0x00500293	addi x5 x0 5	addi x5,x0,5 # const 5
0xc	0x10000313	addi x6 x0 256	addi x6,x0,0x100 # base address of ax[]
0x10	0x00000413	addi x8 x0 0	addi x8,x0,0 # offset = 0
0x14	0x02525063	bge x4 x5 32	bge x4, x5, exit
0x18	0x008303B3	add x7 x6 x8	add x7, x6, x8 # compute effective address
0x1c	0x0003A483	lw x9 0(x7)	lw x9, 0(x7) # get ax[i]
0x70	0x0001A1A3	addi v2 v2 v0	addi v2 v2 v0 # c = c + av[i]

console output

Registers Memory Cache VDB

Integer (R) Floating (F)

Register	Value
zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	38
tp (x4)	5
t0 (x5)	5
t1 (x6)	256
t2 (x7)	272
s0	20

Display Settings: Decimal

VenusEditor

SimulatorChocopy

RunStepPrevResetDumpTraceRe-assemble from Editor

PC	Machine Code	Basic Code	Original Code
0x0	0x00000193	addi x3 x0 0	addi x3,x0,0 # s = 0
0x4	0x00000213	addi x4 x0 0	addi x4,x0,0 # i = 0
0x8	0x00500293	addi x5 x0 5	addi x5,x0,5 # const 5
0xc	0x10000313	addi x6 x0 256	addi x6,x0,0x100 # base address of ax[]
0x10	0x00000413	addi x8 x0 0	addi x8,x0,0 # offset = 0
0x14	0x00525E63	bge x4 x5 28	bge x4, x5, exit
0x18	0x008303B3	add x7 x6 x8	add x7, x6, x8 # compute effective address
0x1c	0x0003A483	lw x9 0(x7)	lw x9, 0(x7) # get ax[i]

Copy!Download!Clear!

console output

registersmemorycachevlog

Integer (R)Floating (F)

zero0

ra0(x1)

sp2147483632(x2)

gp35(x3)

tp5(x4)

t05(x5)

t1256(x6)

t2272(x7)

s020(x8)

Display SettingsDecimal