Undergraduate Computer Architecture, Fall 2024
Final Exam, 2024-12-17

If you agree with the following sentence, please sign your name below it.
I have not cheated nor have I received any help from other students in the exam.

Student ID: b11902038 *Name:* 鄧博允

1. (30 pts) Memory Hierarchy

(a)

a memory hierarchy is

locate data, find a space for data, fetch data, write-back data
(block)

(b)

first, we measure the size we need to store two matrices and the result matrices, assuming N.
then we set up a virtual memory and page tables to translate virtual address into physical address. The swap space should at least with size = N - main memory, to support all PTE has a corresponding physical page number or swap space location.

(c)

since matrix data often stored sequentially in blocks, we can use loop blocking to optimize cache utilization.

with blocking, we could performing the multiplication in blocks, so we can fetch the whole block into cache at once, and remove it after all data being used, reduce memory accessed.
number of

(d)

yes, since each loop of matrix multiplication is independent, that is, we can do A[i+1][j+1] × B[i+1][j+1] whatever A[i][j] × B[i][j] has been done or not.

therefore, we can dispatch each multiplication task into different threads, then collect their result after they're done.

(e)

the 3C stands for:

compulsory miss, which is due to first time access to a block.

capacity miss, which is due to finite cache size.

conflict miss, which is due to competition for entries in a set (not happened in fully associative cache)

(f)

I think the capacity miss might occur most frequently, since large language model is too large to store in cache, thus we might often found that the data we need is not in the cache.

We could increase the cache size or using multi-layer cache to decrease such miss, while it might slightly increase access time (but still faster than fetch from memory and storage)

2. (30 pts) Multiprocessor

(g)

we need to make sure the data will not be written when other processor is r/w, this can be done with locking mechanism, each processor need to acquire a lock to modify the data.

ISA should support these locking operation.

(h)

we can add caches in processors, thus reduce the memory access frequency and improve the memory bandwidth utilization.

(i)

we can use invalidating snooping, when some shared data being write, make all other same copies on other caches being invalid (by broadcasting), thus forced others to re-fetch the data from shared memory.

(j)

we could partition the space in $i = 0 \sim \frac{N}{4}-1$, $\frac{N}{4} \sim \frac{2N}{4}-1$, $\frac{2N}{4} \sim \frac{3N}{4}-1$, and $\frac{3N}{4} \sim N-1$

each processor read data A[i][k], B[k][j] and write data C[i][j].

multiple processor might read same B[k][j] at the same data, but since no one modify it, so it's fine.

for A[i][k] and C[i][j], since each processor responsible for different i, so it's fine, too.

there's no interprocessor data dependency.

(k)

we can use send() and receive() to pass data to a processor and get result after it's done.

It's more cost-effective, since we don't need to handle the problems come with shared-memory, each processor on the cluster is independent, we just dispatch a task to them.

(l)

assume we have 4 processor on the cluster, has processor number = 0~4

```
limit=4; half=4;
do
  half = (half+1)/2;
  if (Pn >= half && Pn < limit)
    send(Pn - half,
```

3. (20 pts) Edge AI and Roof-line Model

(m)

Context stage: $170 \times \frac{144}{150} = 163\ldots$ > peak performance at 162 $\Rightarrow$ 162 (TFLOPs)

Generation stage: $1 \times \frac{144}{150} = 0.96$ (TFLOPs)

(n)

Most of the execution time it at Generation stage, where it's AI is small, meaning it has higher proportion on memory access rather than floating-point operations.

therefore, Tiny Chat is mostly memory-bound.

(o)

Context stage: FP = 162×10^{12} (FLOPs) · 10×10^{-3} (s) = 162×10^{10}

#bytes accessed = $162 \times 10^{10} / 170 = 0.95 \times 10^{10}$

Generation stage: FP = 0.96×10^{12} (FLOPs) · 320×10^{-3} (s) = 30.72×10^{10}

#bytes accessed = $30.72 \times 10^{10} / 1 = 30.72 \times 10^{10}$

total: 1.9×10^{12} FP operations, 3.2×10^{11} bytes

(p)

using FP4 instead of FP16 decrease the bytes accessed as $\frac{1}{4}$, also the time of each FP operations. thus the AI for each stage increase as $4\times$ FLOPs / $\frac{1}{4}\times$ #bytes accessed = $16\times$

for Generation stage, AI raised to 160 and the peak throughput become 153.6. the execution time become $\frac{1}{76}$, that is $\approx$ 20ms. however, for context stage, even AI increase, it has already at peak throughput, thus no improvement.

4. (30 pts) More about Computer Organization

(q)

(r)

```
for (int si = 0~n , si += blocksize)
    for(int sj = 0~n , sj += blocksize)
        for(int sk = 0~n , sk += blocksize)
            matrix_multiplication (n, si, sj, sk, A, B, C)

matrix_multiplication(n, si, sj, sk, A, B, C):
    for( int i = si ~ si+ blocksize , i++)
        for( int j = sj ~ sj + blocksize , j++)
            for( int k = sk ~ sk + blocksize , k++)
                C[i][j] += A[k][i] × B[j][k]
```

this can be dispatch to different SIMD lanes

(s)

```
if data hit in fast stream buffer:
        perform task
else:
        fetch from memory
```

(t)

150 GBytes/sec

Please write down your student ID and your name here.

Student ID: b11902038 *Name:* 鄧博允