CFDS2 - HW3
2022-28981

1.
명령어: srun -p shpc22 -N 1 lscpu

결과:
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 43 bits physical, 48 bits virtual
CPU(s): 128
On-line CPU(s) list: 0-127
Thread(s) per core: 2
Core(s) per socket: 32
Socket(s): 2
NUMA node(s): 2
Vendor ID: AuthenticAMD
CPU family: 23
Model: 49
Model name: AMD EPYC 7502 32-Core Processor
Stepping: 0
Frequency boost: enabled
CPU MHz: 1909.135
CPU max MHz: 2500.0000
CPU min MHz: 1500.0000
BogoMIPS: 5000.09
Virtualization: AMD-V
L1d cache: 2 MiB
L1i cache: 2 MiB
L2 cache: 32 MiB
L3 cache: 256 MiB
NUMA node0 CPU(s): 0-31,64-95
NUMA node1 CPU(s): 32-63,96-127
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf: Not affected
Vulnerability Mds: Not affected
Vulnerability Meltdown: Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Full AMD retpoline, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Not affected

1-a. AMD EPYC 7502 32-Core Processor

1-b. 2

1-c. Base: 2500MHZ(2.5GHZ) / boost: 3350MHZ (3.35GHZ)
CPU's clock frequency는 매 초마다 CPU회로가 얼마나 도는지를 말하는데 예를 들어 3GHZ라면 매초마다 3백만 개의 bits of information을 처리할 수 있다는 것이다.
두 가지 clock frequency가 있는 이유는 기본적으로 cpu가 맞닥드리는 '평시적'상황과 demanding한 상황에서 쓸 수 있는 처리속도를 구분해놓기 위해서다. 평소에는 2.5GHZ의 속도로 처리하다가, demanding한 work가 주어지면 한정적으로 boost clock frequency로 작동할 수 있는 것이다.

1-d. Physical core는 32개, logical core는 64개다. FP32 theoretical peak performance계산은 Execution unit이 physical core단위로 있기 때문에 physical core의 값을 사용해야 한다.
병렬계산이 아닌 그 cpu 자체가 한번에 할 수 있는 최대 계산성능을 확인하고자 하는 것이므로 physical core의 값을 사용해 계산해야 한다.

1-e. 16

우선, 256 / 32 = 8:
256bit의 (fma를 구현하는 일종의 그릇)vector intrinsic에 32bit(float니까)씩 담아놓고 가면 8개씩 갈 수 있다 (SIMD) .
여기에 한 클록사이클 당 2개 avx2 instruction을 날릴 수 있으므로 8*2 =.16

1-f. 2560 GFLOPS
2.5*32*16*2 = 2560GFLOPS
(CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node).

2.

```
Initializing... done!
Calculating...(iter=0) 0.027500 sec
Validating...
Result: VALID
Avg. time: 0.027500 sec
Avg. throughput: 28.638128 GFLOPS
```

(사진 1)

```
Initializing... done!
Calculating...(iter=0) 0.184549 sec
Validating...
Result: VALID
Avg. time: 0.184549 sec
Avg. throughput: 270.995621 GFLOPS
```

(사진2)

(사진1, 사진2는 모드 run_validate.sh를 실행해 나왔던 각각 naive, fma이용한 방식의 계산결과이다)

2-a. 28.638128 GFLOPS로, 1-f에 비해 거의 100배정도 느려진 성능이 나온다. (사진1)

2-b.
Reference:
https://sites.utexas.edu/jdm4372/tag/high-performance-computing/

1) Peak performance는 기본 2개의 fma instruction per cycle을 기준으로 계산되기 때문에 당연히 fma instruction을 쓰지 않는 naive한 경우는 peak performance보다 떨어질 수밖에 없다.
“Since “peak performance” for the processor is two (512-bit SIMD) FMA instructions per cycle, any instructions that are not FMA instructions subtract directly from the available peak performance.”
2) 다른 프로그램이 쓰고 있어 CPU코어 중 사용 불가능한 것이 있을 수도 있을 것이며
3) 단순한 이유지만 루프가 들어가 있어 느릴 수밖에 없을 것 같다.

2-c. (FMA instruction이용) 270.889621 GLOPS (사진2)

2-d.
1) 마찬가지로 루프를 돌고 (물론 한 번 돌 때 처리하는 량은 늘어났다)

2) 병렬처리가 되지 않았다.

3.

3-a.

스레드는 행렬의 행에 따라 일을 나눈다.

연산을 진행할 때, 연산에 쓰일 데이터블록을 여러번 로드하지 않게 한다. 즉, 한 번 데이터를 메모리에 올렸을 때 필요한 만큼을 모두 계산해 놓아 다음에 해당 데이터가 필요하지 않도록 다음 계산target을 블록 크기만큼 옮겨서 작업한다.

3-b.

자원배분을 담당하는 리눅스의 커널(OS의 커널)이 담당할 듯 하다

Number of threads: 1	Avg. throughput: 9.529256 GFLOPS
Number of threads: 16	Avg. throughput: 89.140011 GFLOPS
Number of threads: 32	Avg. throughput: 102.980159 GFLOPS
Number of threads: 48	Avg. throughput: 92.586813 GFLOPS
Number of threads: 64	Avg. throughput: 120.454225 GFLOPS
Number of threads: 80	Avg. throughput: 84.757550 GFLOPS
Number of threads: 96	Avg. throughput: 78.784167 GFLOPS
Number of threads: 112	Avg. throughput: 60.626944 GFLOPS
Number of threads: 128	Avg. throughput: 74.095781 GFLOPS
Number of threads: 144	Avg. throughput: 71.033810 GFLOPS
Number of threads: 160	Avg. throughput: 55.715420 GFLOPS
Number of threads: 176	Avg. throughput: 26.895531 GFLOPS
Number of threads: 192	Avg. throughput: 46.221139 GFLOPS
Number of threads: 208	Avg. throughput: 44.378075 GFLOPS
Number of threads: 224	Avg. throughput: 51.510627 GFLOPS
Number of threads: 240	Avg. throughput: 62.521524 GFLOPS
Number of threads: 256	Avg. throughput: 59.306467 GFLOPS

3-c. 표(좌) 참고, 크기는 M=N=K=4096으로 고정하고 작업한 결과. 스레드 수가 증가함에 따라 gflops가 증가하지만, 64를 기점으로 점점 줄어들다가 또 192즈음에서 약간의 분산과 함께 50-60대에 머문다.

스레드를 일꾼에 비유하자면, 너무 많은 일꾼들이 작업에 참여하면 일꾼들이 일을 하고 이를 합쳐서 처리하는 데에s 대기시간에 지체가 생길 것이기 때문에 일을 처리하기 위한 적정한 스레드 수를 찾는 게 중요하다.

3-d. 64개에서 120.454225GFLOPS가 나온다. 알고리즘을 개선하여 할 듯 하다. O(N^3)미만으로 나오는 스트라센 알고리즘 등을 활용하면 더 성능이 개선될 것이다.

Reference:

https://sangggggg.github.io/matrix_multiplication/pthread/multicore/2020/05/28/matrix-multiplication-2.html