Programming Techniques for Supercomputers

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Exercise 8 – GPUs

Erlangen National High Performance Computing Center (NHR@FAU)

Sommersemester 2026

Assignment 8 – Task 1

Performance: $P_{chip} = n_{core} \cdot n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f$

- Float
  - $P_{CPU} = (1 \cdot 144) \cdot 1\frac{inst}{cyc} \cdot 2\frac{Flop}{FMA} \cdot 16\frac{FMA}{inst} \cdot 2.8\,GHz = 12.9\frac{TFlop}{s}$
  - $P_{GPU} = (92 \cdot 4) \cdot 16\frac{inst}{cyc} \cdot 2\frac{Flop}{FMA} \cdot 1\frac{FMA}{inst} \cdot 1.2\,GHz = 14.1\frac{TFlop}{s}$
  - $P_{node} = 8 \cdot P_{GPU} = 113.0\frac{TFlop}{s}$
  - $P_{cluster} = 212 \cdot P_{node} = 24.0\frac{PFlop}{s}$

Assignment 8 – Task 1

Performance: $P_{chip} = n_{core} \cdot n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f$

- Double
  - $P_{CPU} = (1 \cdot 144) \cdot 1\frac{inst}{cyc} \cdot 2\frac{Flop}{FMA} \cdot 8\frac{FMA}{inst} \cdot 2.8\,GHz = 6.5\frac{TFlop}{s}$
  - $P_{GPU} = (92 \cdot 4) \cdot 1\frac{inst}{cyc} \cdot 2\frac{Flop}{FMA} \cdot 1\frac{FMA}{inst} \cdot 1.2\,GHz = 0.9\frac{TFlop}{s}$
  - $P_{node} = 8 \cdot P_{GPU} = 7.1\frac{TFlop}{s}$
  - $P_{cluster} = 212 \cdot P_{node} = 1.5\frac{PFlop}{s}$

Assignment 8 – Task 2

- #FLOP for a **D**ouble-Precision **G**eneral **M**atrix **M**atrix Multiplication (DGEMM) for matrices with dimensions MK x KN = MN
- $N_{flops} = 2 \cdot MNK$ †
- Assume compute boundness in the computational part
- Assume no overlap between
  - Host-device data transfers
  - Device-host data transfers
  - Computation

† technically $N_{\text{flops}} = MN(2K - 1)$

Assignment 8 – Task 2

- Float
  - $T_{CPU} = \frac{N_{Flops}}{P_{CPU}} = \frac{2 \cdot 4096^3 Flop}{12.9 \frac{TFlop}{s}} = 10.7\ ms$
  - $T_{GPU} = \frac{N_{Flops}}{P_{GPU}} = \frac{2 \cdot 4096^3 Flop}{113.0 \frac{TFlop}{s}} = 1.2\ ms$
  - $T_{PCIe} = \frac{V}{B} = \frac{3 \cdot 4096^2 \cdot 4\ B}{4 \cdot 32.0 \frac{GB}{s}} = 1.6\ ms$
  - $T_{Total} = T_{GPU} + T_{PCIe} = 1.2 + 1.6\ ms = 2.8\ ms$
  - $S = \frac{T_{CPU}}{T_{Total}} = 3.8$

Assignment 8 – Task 2

- Double
  - $T_{CPU} = \frac{N_{Flops}}{P_{CPU}} = \frac{2 \cdot 4096^3 Flop}{6.5 \frac{TFlop}{s}} = 21.3\, ms$
  - $T_{GPU} = \frac{N_{Flops}}{P_{GPU}} = \frac{2 \cdot 4096^3 Flop}{7.1 \frac{TFlop}{s}} = 19.5\, ms$
  - $T_{PCIe} = \frac{V}{B} = \frac{3 \cdot 4096^2 \cdot 8\, B}{4 \cdot 32.0 \frac{GB}{s}} = 3.1\, ms$
  - $T_{Total} = T_{GPU} + T_{PCIe} = 19.5 + 3.1\, ms = 22.6\, ms$
  - $S = \frac{T_{CPU}}{T_{Total}} = 0.9$

Assignment 8 – Task 3

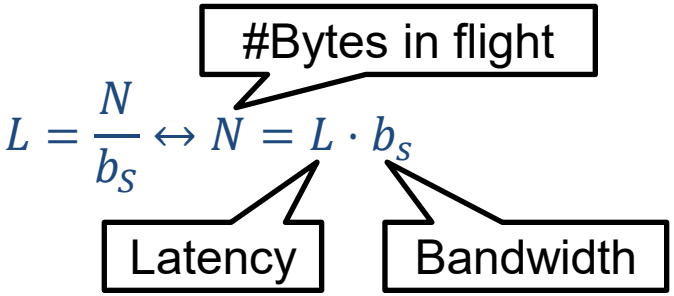
- Kernel code

```
__global__ void kernel(float* A, float* B, float* C, int size) {
  int tidx = threadIdx.x + blockDim.x * blockIdx.x;
  int tidy = threadIdx.y + blockDim.y * blockIdx.y;

  if (tidx >= size || tidy >= size) return;

  C[tidx + tidy * size] += A[tidx + tidy * size] * B[tidy];
}
```

Two DRAM loads and one DRAM store (assuming B fits into L2)

Assignment 8 – Task 3

- Little's Law



Assignment 8 – Task 3

- Little's Law
  - $L = \frac{N}{b_S} \leftrightarrow N = L \cdot b_S$
  - $L_{seconds} = \frac{L_{cycles}}{F}$
  - $N = N_{threads} \cdot V_{thread}$
  - $N = 1200.0 \frac{GB}{s} \cdot \frac{720}{1.2\ GHz} = 720\ KB$
  - $N_{threads} = \frac{N}{4 \cdot 3\frac{B}{thread}} = \frac{720\ KB}{12\frac{B}{thread}} = \mathbf{60000}$

Minimum number of threads required to saturate the DRAM bandwidth, actual number is usually higher

Assignment 8 – Task 3

- Kernel code

```
__global__ void kernel(float* A, float* B, float* C, int size) {
  int tidx = threadIdx.x + blockDim.x * blockIdx.x;
  int tidy = threadIdx.y + blockDim.y * blockIdx.y;

  if (tidx >= size || tidy >= size) return;

  C[tidx + tidy * size] += A[tidx + tidy * size] * B[tidy];
}
```

minimized cache traffic if block width allows full cache line access, i.e., multiple of 8

Threads with same tidy can share data in L1
-> blocks with large x-extent reduce L2 cache traffic

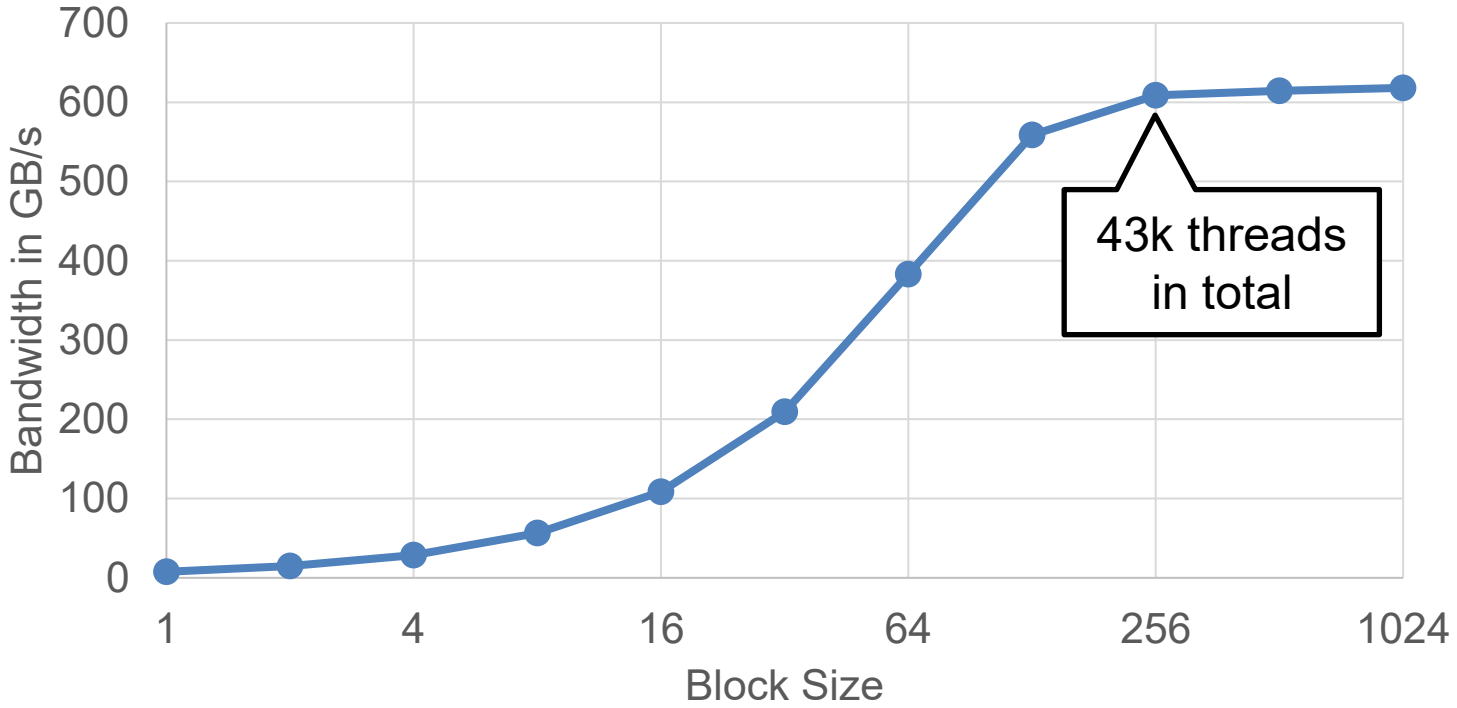
Assignment 8 – Task 4

- Kernel code for `a[i] = a[i] * 2.0`

```
__global__ void kernel(double* a, int size) {
  int x = threadIdx.x + blockDim.x * blockIdx.x;
  int stride = gridDim.x * blockDim.x;

  for ( ; x <= size; x += stride)
    a[x] *= 2;
}
```

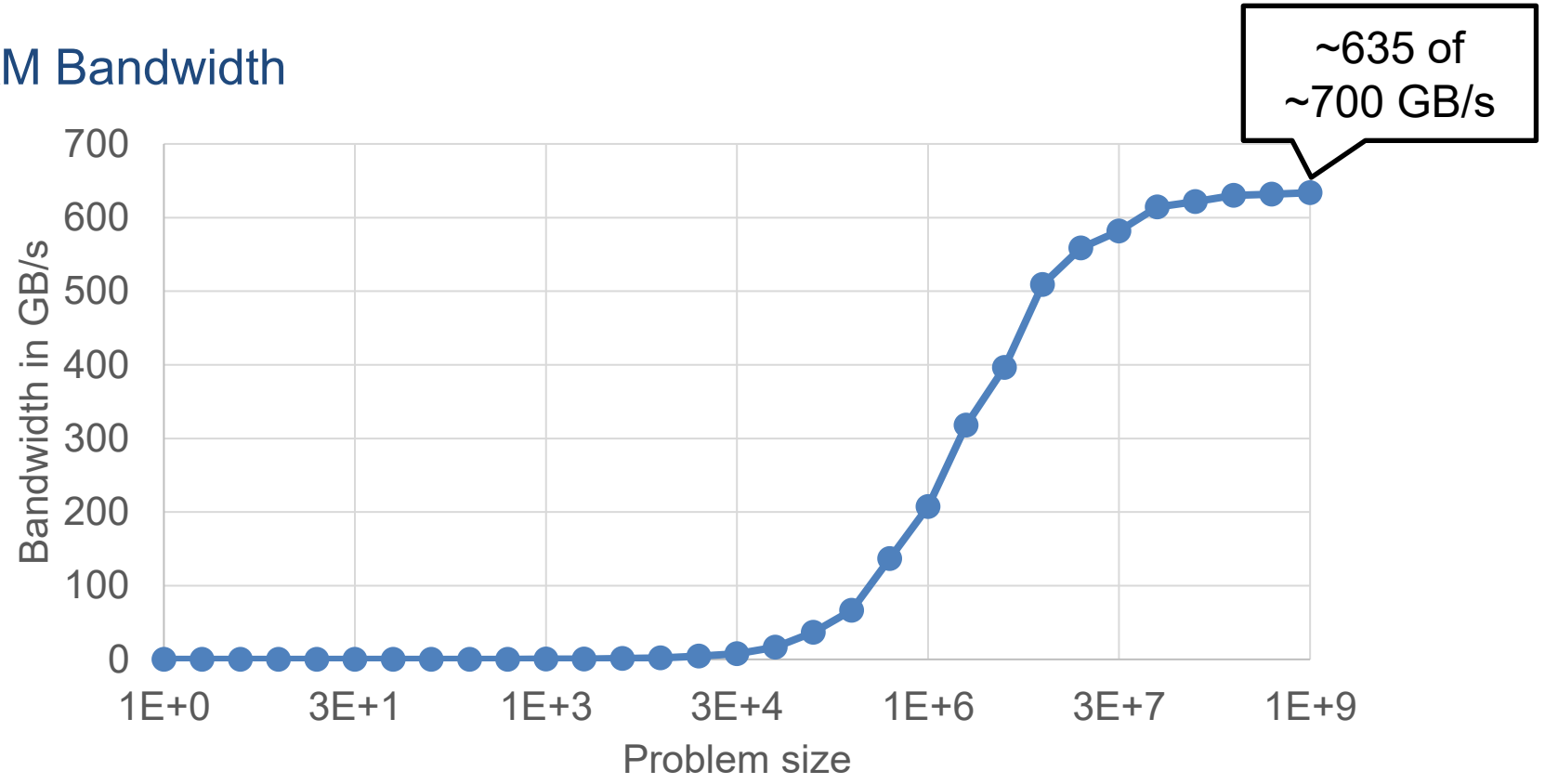
Assignment 8 – Task 5

- Block size investigation



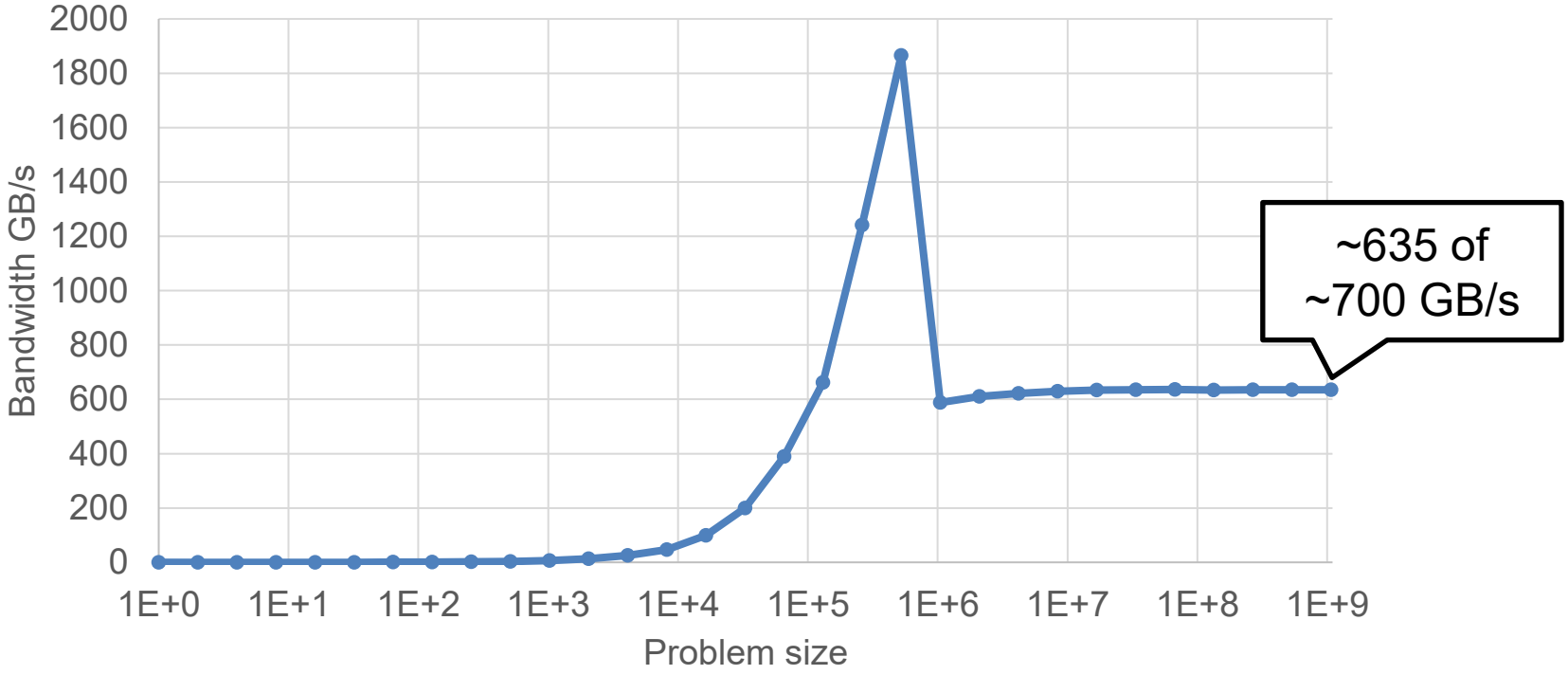
Assignment 8 – Task 6

- DRAM Bandwidth



Assignment 8 – Task 6

- DRAM Bandwidth (with one warm-up iteration, not part of exercise)



Assignment 8 – Task 6

- PCIe Bandwidth