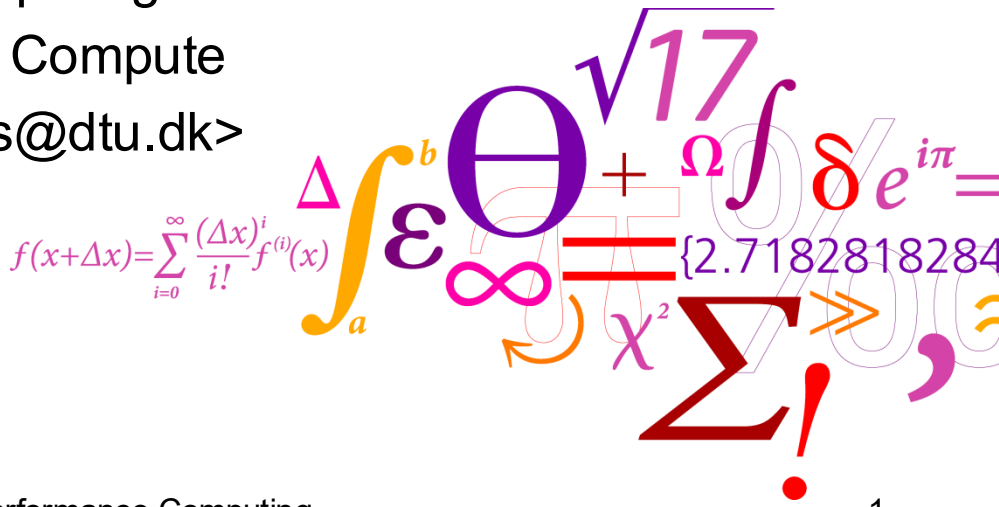
Introduction to GPU performance tuning


Hans Henrik Brandenborg Sørensen
DTU Computing Center
DTU Compute
<hhbs@dtu.dk>

DTU DCC Computing Center

GPUlab DTU Informatics



Overview

- Recap from week 1 (now with GPUs)
  - Performance metrics
  - Assessing your performance
- Speed-up GPU vs. CPU (and what is ‘fair’?)
- GPU performance tuning process
  - Transpose example
  - How many threads should I launch?
  - Latency hiding

GPU performance metrics

Performance tuning terminology

- Execution time [seconds]
  - Time to run the application (wall or cpu/gpu)
- Performance [Gflops]
  - How many floating point operations per second
- Latency [cycles or seconds]
  - Time from initiating a memory access or other action until the result is available
- Bandwidth [Gb/s]
  - The rate at which data can be transferred
- Blocking [blocksize]
  - Dividing matrices into tiles to fit memory hierarchy

You know these from week 1+2 of this course!

Performance tuning terminology

- Throughput [#/s or Gb/s]
  - Sustained rate for instructions executed or data reads + writes achieved in practice
- Occupancy [%]
  - Ratio of active threads to max possible in hardware
- Instruction level parallelism (ILP) [#]
  - How many independent instructions can be executed (=pipelining)
- Thread level parallelism (TLP) [#]
  - How many independent threads can be launched
- Coalescing
  - 32 neighbor threads in team are reading data from a contiguous, aligned, region of global memory

Common GPU optimization teminology.

Which performance metric to use?

- Compute bound
  - Limited by # flops * time per flop

Gflops = # floating point operations / 10^9 / runtime

Which performance metric to use?

- Compute bound
  - Limited by # flops * time per flop

Gflops = # floating point operations / 10^9 / runtime

- Memory bound
  - Limited by # bytes moved / bandwidth

Bandwidth = (Bytes_read + Bytes_written) / 10^9 / runtime

How to assess your performance?

- Compute bound
  - Compare with the theoretical peak performance
    - Run nvaccelinfo to find specs and calculate, e.g.

SP: 114 SM * 128 cores/SM * 1.755 GHz *2 flops = 51.2 Tflops

DP: (1/2) * 51.2 Tflops = 25.6 Tflops

How to assess your performance?

- Compute bound
  - Compare with the theoretical peak performance
    - Run nvaccelinfo to find specs and calculate, e.g.

SP: 114 SM * 128 cores/SM * 1.755 GHz *2 flops = 51.2 Tflops

DP: (1/2) * 51.2 Tflops = 25.6 Tflops

- Memory bound
  - Compare with the theoretical peak bandwidth
    - Run nvaccelinfo to find specs and calculate, e.g.

Peak bandwidth = 1.593 GHz*(5120 / 8) bytes*2 = 2039 GB/s

How to assess your performance?

- Compute bound
  - Compare with the theoretical peak performance
    - Run nvaccelinfo to find specs and calculate, e.g.

SP: 114 SM * 12 ops = 51.2 Tflops

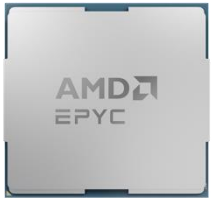
DP: ops = 25.6 Tflops

40-60%: okay
60-75%: good
>75%: excellent

- Memory b
  - Compare with the theoretical peak bandwidth
    - Run nvaccelinfo to find specs and calculate, e.g.

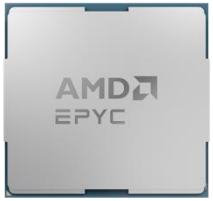
Peak bandwidth = 1.593 GHz*(5120 / 8) bytes*2 = 2039 GB/s

How to assess CPU performance?



- Find the CPU specs online: https://www.amd.com/en/products/cpu/amd-epyc-9354

How to assess CPU performance?



- Find the CPU specs online: https://www.amd.com/en/products/cpu/amd-epyc-9354
- Compute bound

SP: 32 cores * 3.75 GHz (AVX-512) * 16 (AVX-512) * 2 (FMA units) * 2 flops per core = 7.68 Tflops

DP: (1/2) * 7.68 Tflops = 3.84 Tflops

- Memory bound

Peak bandwidth = 4.8 GHz * (64 / 8) bytes * 12 = 460.8 GB/s

Speed-up

Speed-up (now with GPUs)

- GPU definition of speed-up is traditionally

$$\texttt{Speedup} = \frac{\texttt{CPUtime}[\mathrm{s}]}{\texttt{GPUtime}[\mathrm{s}]}$$

  and usually written in times (×) manner, e.g., 3.2 ×

- Useful for indicating performance without telling what the performance actually is(!)

Speed-up (now with GPUs)

- GPU definition of speed-up is traditionally

$$\texttt{Speedup} = \frac{\texttt{CPUtime}[\mathrm{s}]}{\texttt{GPUtime}[\mathrm{s}]}$$

and usually written in times (×) manner, e.g., 3.2 ×

- Useful for indicating performance without telling what the performance actually is(!)

But be fair when comparing to CPU times – speed-ups of 100 × -1000 × (see `http://www.nvidia.com/object/cuda_showcase_html.html`) are unrealistic when the hardware specs are taken into account

Speed-up (now with GPUs)

- Expected speed-up on our nodes
  - Compute bound: 25.6 / 3.84 = 6.7x
  - Memory bound: 2039 / 460.8 = 4.4x

Speed-up (now with GPUs)

- Expected speed-up on our nodes
  - Compute bound: 25.6 / 3.84 = 6.7x
  - Memory bound: 2039 / 460.8 = 4.4x
- Better speed-ups may be seen in practice
  - Execution model (SIMD / SIMT) – see next slide
  - Compiler matureness (vectorization is difficult)
  - …but not by orders of magnitude(!)

Executing of the GPU

- Execution model maps code to instructions
  - SIMD – Single Instruction, Multiple Data
  - SIMT – Single Instruction, Multiple Threads
- CPU (SIMD mapping)
  - The compiler takes care of mapping code to the most efficient instructions at compile time
- GPU (SIMT mapping)
  - Kernels look like serial functions for a SINGLE thread
  - CUDA will automatically launch on MANY threads
  - Code is mapped to instructions at runtime!

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$

```
double pi(long N)
{
   double sum = 0.0;
   double h = 1.0/N;
   #pragma omp parallel for \
       reduction(+: sum)
   for(long i=1; i<=N; i++) {
      double x = h*(i-0.5);
      sum += 4.0/(1.0+x*x);
   }
   return h*sum;
}
```

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$


```
double pi(long N)
{
    double sum = 0.0;
    double h = 1.0/N;
    #pragma omp parallel for \
        reduction(+: sum)
    for(long i=1; i<=N; i++) {
       double x = h*(i-0.5);
       sum += 4.0/(1.0+x*x);
    }
    return h*sum;
}
```

```
icx -O3 -fopenmp pi_cpu.cpp
```

OMP_NUM_THREADS	-O3 (s)
1	99.5
2	50.2
4	25.6
8	12.9
16	6.5
32	3.3
64	3.1

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$


```
double pi(long N)
{
    double sum = 0.0;
    double h = 1.0/N;
    #pragma omp parallel for \
        reduction(+: sum)
    for(long i=1; i<=N; i++) {
       double x = h*(i-0.5);
       sum += 4.0/(1.0+x*x);
    }
    return h*sum;
}
```

```
icx -O3 -fopenmp pi_cpu.cpp
icx -xcore-avx512 -O3 …
```

OMP_NUM_THREADS	-O3 (s)	avx (s)
1	99.5	30.3
2	50.2	15.1
4	25.6	7.6
8	12.9	3.8
16	6.5	1.9
32	3.3	1.0
64	3.1	0.9

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$


```
double pi(long N)
{
    double h = 1.0/N;
    double sum = 0.0;
    double x;
    #pragma omp parallel for \
    for (long i=0; i<N; i++) {
        x = h*(i+0.5);
        sum += 4.0/(1.0+x*x);
    }
    return h*sum;
}
```

remark #15305: vectorization support: vector length **2**
remark #15399: vectorization support: unroll factor set to 4
remark #15300: LOOP WAS VECTORIZED
remark #15486: divides: 1

remark #15305: vectorization support: vector length **8**
remark #15399: vectorization support: unroll factor set to 4
remark #15300: LOOP WAS VECTORIZED
remark #15486: divides: 1

```
icx -O3 -fopenmp pi_cpu.cpp
icx -xcore-avx512 -O3 …
```

OMP_NUM_THREADS	-O3 (s)	avx (s)
1	99.5	30.3
2	50.2	15.1
4	25.6	7.6
8	12.9	3.8
16	6.5	1.9
32	3.3	1.0
64	3.1	0.9

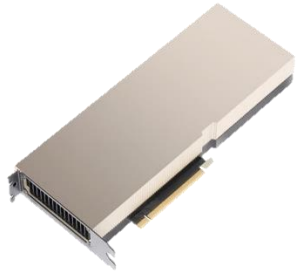
Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$

```
double pi(long N)
{
   double sum = 0.0;
   double h = 1.0/N;
   #pragma omp target teams \
       num_teams(16384) \
       thread_limit(128) \
       distribute parallel for \
       reduction(+: sum)
   for(long i=1; i<=N; i++) {
      double x = h*(i-0.5);
      sum += 4.0/(1.0+x*x);
   }
   return h*sum;
}
```

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$



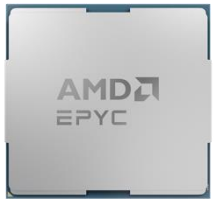
```
double pi(long N)
{
   double sum = 0.0;
   double h = 1.0/N;
   #pragma omp target teams \
       num_teams(16384) \
       thread_limit(128) \
       distribute parallel for \
       reduction(+: sum)
   for(long i=1; i<=N; i++) {
      double x = h*(i-0.5);
      sum += 4.0/(1.0+x*x);
   }
   return h*sum;
}
```

```
nvc++ -fast -mp=gpu pi_gpu.cpp
```

num_teams	thread_limit	nvc (s)
16384	1	6.17
16384	2	3.08
16384	4	1.54
16384	8	0.77
16384	32	0.19
16384	64	0.18
16384	128	0.18

Example of speed-up calculation

$$\pi = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}.$$



1 CPU / # threads = # cores / `avx512`

Speed-up = 1.0 seconds / 0.18 seconds = 5.6 ×

1 GPU / best teams parallel configuration / deduct warm up

GPU performance tuning process

GPU performance tuning process

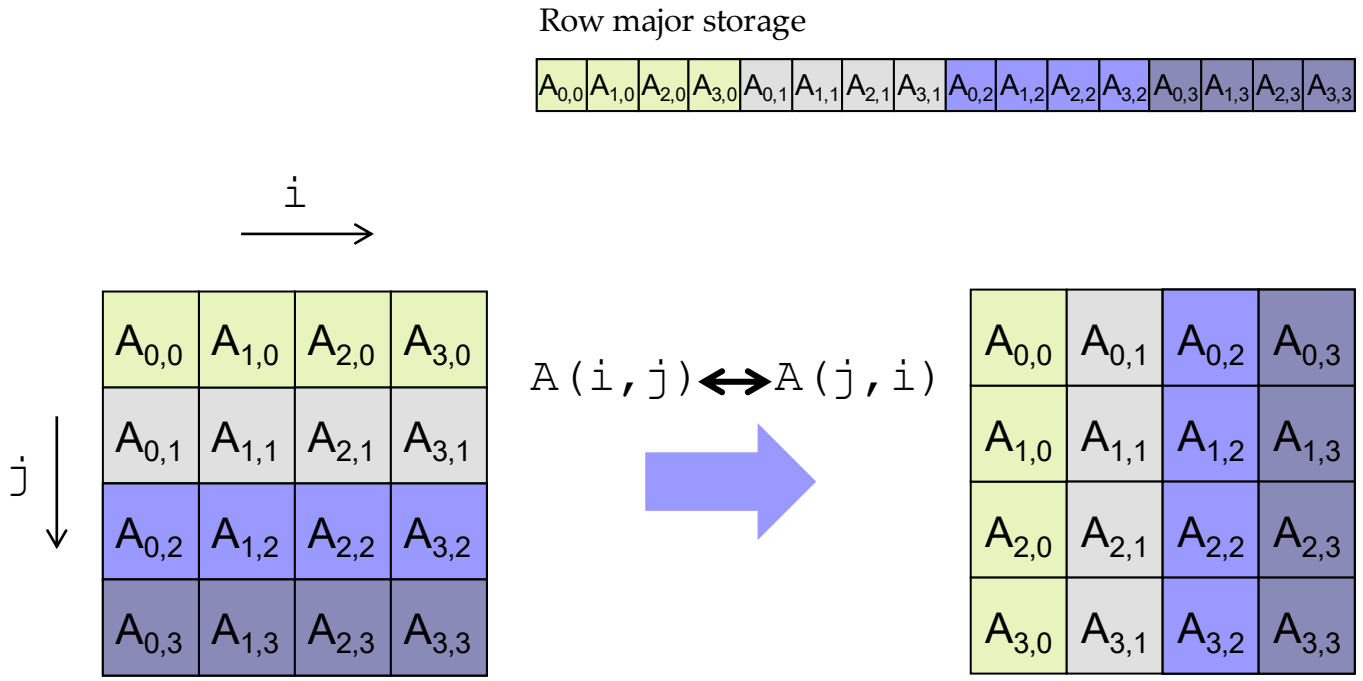
- Determine what limits the offload performance
  - Parallelism (concurrency bound)
  - Memory accesses (memory bound)
  - Floating point operations (compute bound)

… or a combination

- Use appropriate performance metric for the offload
  - Or use speed-up between kernel modifications
- Address the limiters in the order of importance
  - Determine how close you are to the theoretical peaks
  - Analyze
  - Apply optimizations

… and iterate with small steps

Transpose example



- We will use this example to illustrate the process of performance tuning an OpenMP offload code "step-by-step"

Transpose example (v1 seq)

```
// Reference sequential CPU transpose
void transpose(double **A, double **At)
{
    for(int i=0; i < N; i++)
        for(int j=0; j < N; j++)
            At[i][j] = A[j][i];
}
```

```
// Baseline sequential GPU transpose
#pragma omp declare target(transpose)

void transpose_seq(double **A, double **At)
{
    #pragma omp target
    transpose(A, At);
}
```

Transpose example (v1 seq)

```
#define N 7296
...
   // CPU reference transpose for checking result
   transpose(A, At_CPU);

   #pragma omp target enter data \
       map(to: A[0:N][0:N]) map(alloc: At[0:N][0:N])

    // GPU sequential version
    transpose_seq(A, At);

    #pragma omp target exit data \
        map(release: A[0:N][0:N]) map(from: At[0:N][0:N])

    check(At, At_CPU); // Check result
...
```

Transpose example (v1 seq)

```
$ nvc++ -fast -Msafeptr -Minfo -mp=gpu -gpu=cc90 -acc -c -o transpose.o transpose.cpp
...
transpose_seq(double**, double**, long):
          17, Loop not vectorized/parallelized: not countable
     27, #omp target
         27, Generating "nvkernel__Z13transpose_seqPPdS0_l_F1L27_3" GPU kernel
...

& nsys profile --trace=cuda --stats=true ./transpose
...
[4/6] Executing 'gpukernsum' stats report

Time (%)  Total Time (ns)  Instances      Avg (ns)      ...                    Name
--------  ---------------  ---------  ----------------     ----------------------------------------
   100.0   12,305,926,981          1  12,305,926,981.0     nvkernel__Z13transpose_seqPPdS0_l_F1L27_3
     0.0          144,226          2          72,113.0     nvkernel__Z13malloc_2d_deviiPPd_F1L45_2
```

`nsys` command line profiling tool

Version	v1 seq
Time [ms]	12305

Transpose example (v2 per col)

```
// OpenMP offload transpose using one thread per col of A
void transpose_per_col(double **A, double **At)
{
    #pragma omp target teams distribute parallel for \
        num_teams(114) thread_limit(64)
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            At[i][j] = A[j][i];
}
```

Transpose example (v2 per col)

```
$ nvc++ -fast -Msafeptr -Minfo -mp=gpu -gpu=cc90 -acc -c -o transpose.o transpose.cpp
...
transpose_per_col(double **, double **):
     52, #omp target teams distribute parallel for num_teams(114) thread_limit(64)
         52, Generating "nvkernel__Z17transpose_per_colPPdS0_l_F1L52_13" GPU kernel
         58, Loop parallelized across teams and threads(128), schedule(static)
     58, Loop not vectorized/parallelized: not countable
     59, Loop not vectorized: unprofitable for target
         Loop unrolled 4 times

& nsys profile --trace=cuda --stats=true ./transpose
...
[4/6] Executing 'gpukernsum' stats report
```

```
Time (%)  Total Time (ns)  Instances   Avg (ns)   ...                  Name
--------  ---------------  ---------  -----------     ----------------------------------------
   100.0      419,244,419        100  4,192,444.2     nvkernel__Z17transpose_per_colPPdS0_l_F1L52_13
     0.0          144,226          2     72,113.0     nvkernel__Z13malloc_2d_deviiPPd_F1L45_2
```

Version	v1 seq	v2 per col
Time [ms]	12305	4.19

Transpose example (v3 per elm)

```
// OpenMP offload transpose using one thread per element
void transpose_per_elm(double **A, double **At)
{
    #pragma omp target teams distribute parallel for \
        collapse(2) num_teams(N*N/64) thread_limit(64)
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            At[i][j] = A[j][i];
}
```

Transpose example (v3 per elm)

```
$ nvc++ -fast -Msafeptr -Minfo -mp=gpu -gpu=cc90 -acc -c -o transpose.o transpose.cpp
...
transpose_per_elm(double **, double **):
     63, #omp target teams distribute parallel for num_teams(831744) thread_limit(64)
         63, Generating "nvkernel__Z17transpose_per_elmPPdS0_l_F1L63_17" GPU kernel
         72, Loop parallelized across teams and threads(128), schedule(static)
     73, Loop not vectorized/parallelized: not countable


& nsys profile --trace=cuda --stats=true ./transpose
...
[4/6] Executing 'gpukernsum' stats report
```

Time (%)	Total Time (ns)	Instances	Avg (ns)	..	Name
99.8	78,072,972	100	780,729.7	..	nvkernel__Z17transpose_per_elmPPdS0_l_F1L63_17
0.2	152,064	2	76,032.0	..	nvkernel__Z13malloc_2d_deviiPPd_F1L45_2

Version	v1 seq	v2 per col	v3 per elm
Time [ms]	12305	4.19	0.78

How many threads should I run?

- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`

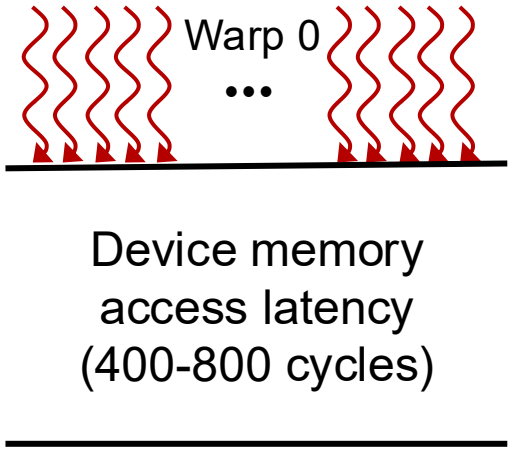
How many threads should I run?

- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`

How many threads should I run?

- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`

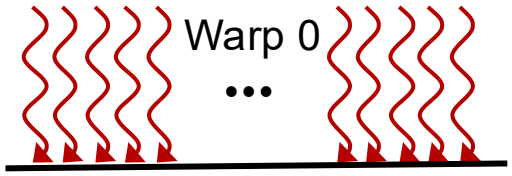
```
A[j][i];
```



How many threads should I run?

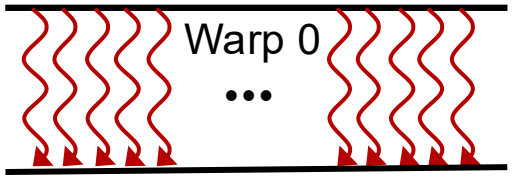
- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`

```
A[j][i];
```



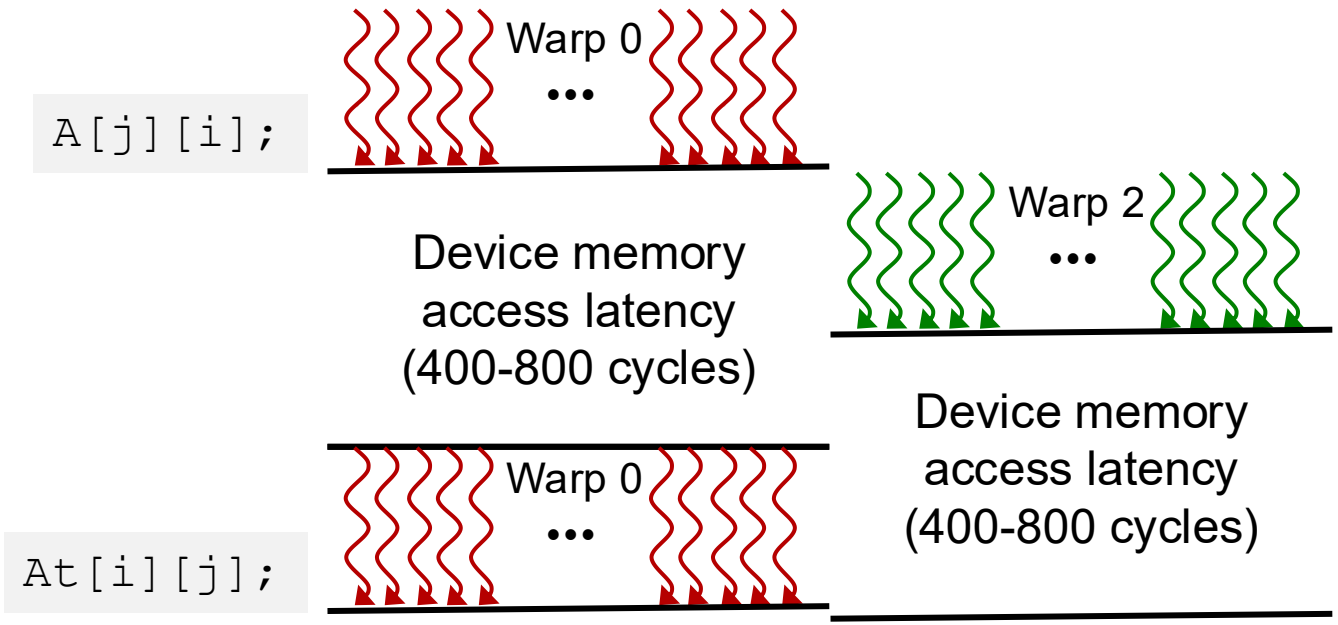
Device memory
access latency
(400-800 cycles)

```
At[i][j];
```



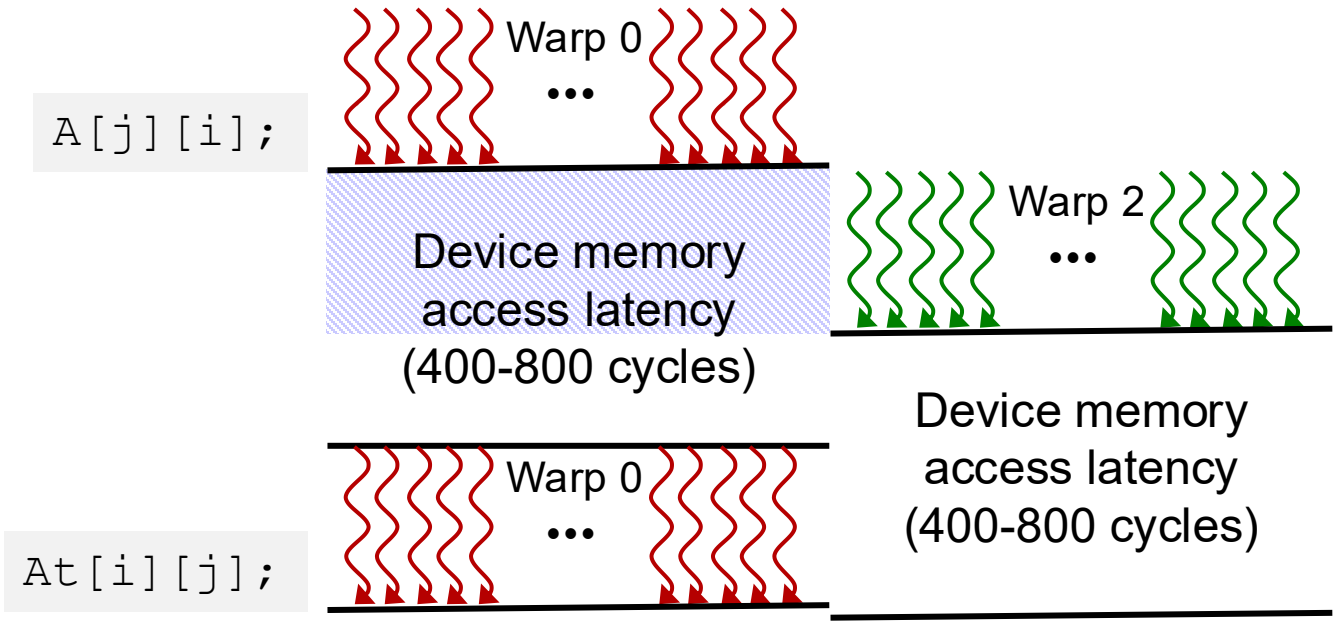
How many threads should I run?

- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`



How many threads should I run?

- Why is one thread per core not enough?
  - E.g. `num_teams(114) thread_limit(64)`



- Reason: We can hide memory latency by having idle warps to schedule while waiting for data

Transpose example (v4 warp)

```
// OpenMP offload transpose using one warp per col
void transpose_warp_per_col(double **A, double **At)
{
    #pragma omp target teams loop \
        num_teams(N) thread_limit(32)
    for(int i = 0; i < N; i++) {
        #pragma omp loop bind(parallel)
        for(int j = 0; j < N; j++)
            At[i][j] = A[j][i];
    }
}
```

Transpose example (v4 warp)

```
$ nvc++ -fast -Msafeptr -Minfo -mp=gpu -gpu=cc90 -acc -c -o transpose.o transpose.cpp
...
transpose_warp_per_col(double **, double **, long):
     80, #omp target teams loop num_teams(7296) thread_limit(32)
         80, Generating "nvkernel__Z22transpose_warp_per_colPPdS0_l_F1L80_19" GPU kernel
             Generating NVIDIA GPU code
           86, Loop parallelized across teams(7296) /* blockIdx.x */
           88, Loop parallelized across threads(32) /* threadIdx.x */
         80, Generating Multicore code
           86, Loop parallelized across threads
     88, Loop is parallelizable
         Loop not vectorized: unprofitable for target
         Loop unrolled 4 times

& nsys profile --trace=cuda --stats=true ./transpose
...
[4/6] Executing 'gpukernsum' stats report
```

Time (%)	Total Time (ns)	Instances	Avg (ns)	..	Name
99.8	67,454,944	100	674,549.4		nvkernel__Z22transpose_warp_per_rowPPdS0_l_F1L78_21
0.2	152,096	2	76,048.0.		nvkernel__Z13malloc_2d_deviiPPd_F1L45_2

Version	v1 seq	v2 per col	v3 per elm	v4 warp
Time [ms]	12305	4.19	0.78	0.67

End of lecture