GPU-Offloading using OpenMP

Seminar Efficient Programming
by Department of Scientific Computing
at University of Hamburg

21.01.2021

by Yannik Könneker

Structure

- OpenMP
- GPU-Offloading
  - GPU's
  - Execution Model
  - Data Mapping

Introduction

Why should I care?

Problem

Scientists:

- need simulations
- require deep understanding for fast and efficient simulations
- rarely are computer scientists
- time is very precious

Problem – an example

```
int main() {
   uint32_t matA[MATSIZE][MATSIZE];
   uint32_t matB[MATSIZE][MATSIZE];
   uint32_t matC[MATSIZE][MATSIZE];

   // Init matA and matB with numbers, matC with 0's
   fill(matA, matB, matC);

   for (int i = 0; i < MATSIZE; i++)
     for (int j = 0; j < MATSIZE; j++)
       for(int k = 0; k < MATSIZE; k++)
         matC[i][j] += matA[i][k] * matB[k][j];
   return EXIT_SUCCESS;
 }
```

conventional, sequential matrix multiplication

Problem – an example

```
int main() {
   uint32_t matA[MATSIZE][MATSIZE];
   uint32_t matB[MATSIZE][MATSIZE];
   uint32_t matC[MATSIZE][MATSIZE];

   // Init matA and matB with numbers, matC with 0's
   fill(matA, matB, matC);

   for (int i = 0; i < MATSIZE; i++)
     for (int j = 0; j < MATSIZE; j++)
       for(int k = 0; k < MATSIZE; k++)
         matC[i][j] += matA[i][k] * matB[k][j];
   return EXIT_SUCCESS;
 }
```

MATSIZE	Time
100	0.0035s
1.000	2.7s
10.000	(~6200s)

$O(n^3)$

conventional, sequential matrix multiplication

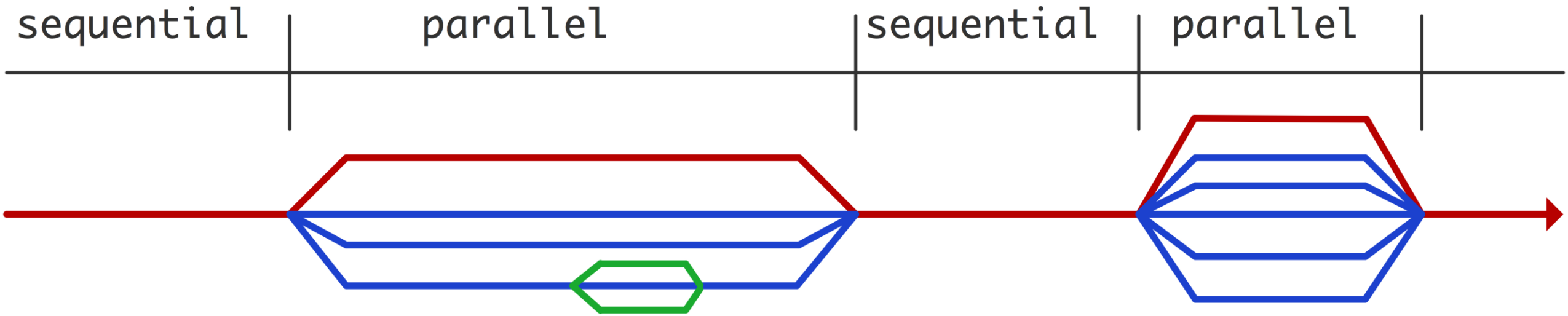
OpenMP

Detailed information in this reference sheet or in the specification

OpenMP

- API using compiler directives (#pragma omp)
- Defacto standard for shared memory parallel programming [1]
- C/C++ and Fortran
- Plattform independent

OpenMP - fork and join model



[3]

```
#pragma omp parallel
{ ... }
```

Automatically parallelizes code

OpenMP - example revisited

```
int main() {
   uint32_t matA[MATSIZE][MATSIZE];
   uint32_t matB[MATSIZE][MATSIZE];
   uint32_t matC[MATSIZE][MATSIZE];

   // Init matA and matB with numbers, matC with 0's
   fill(matA, matB, matC);

   #pragma omp parallel for shared(matA, matB, matC) collapse(2)
   for (int i = 0; i < MATSIZE; i++)
     for (int j = 0; j < MATSIZE; j++)
       for(int k = 0; k < MATSIZE; k++)
         matC[i][j] += matA[i][k] * matB[k][j];
   return EXIT_SUCCESS;
 }
```

parallel matrix multiplication

OpenMP - example revisited

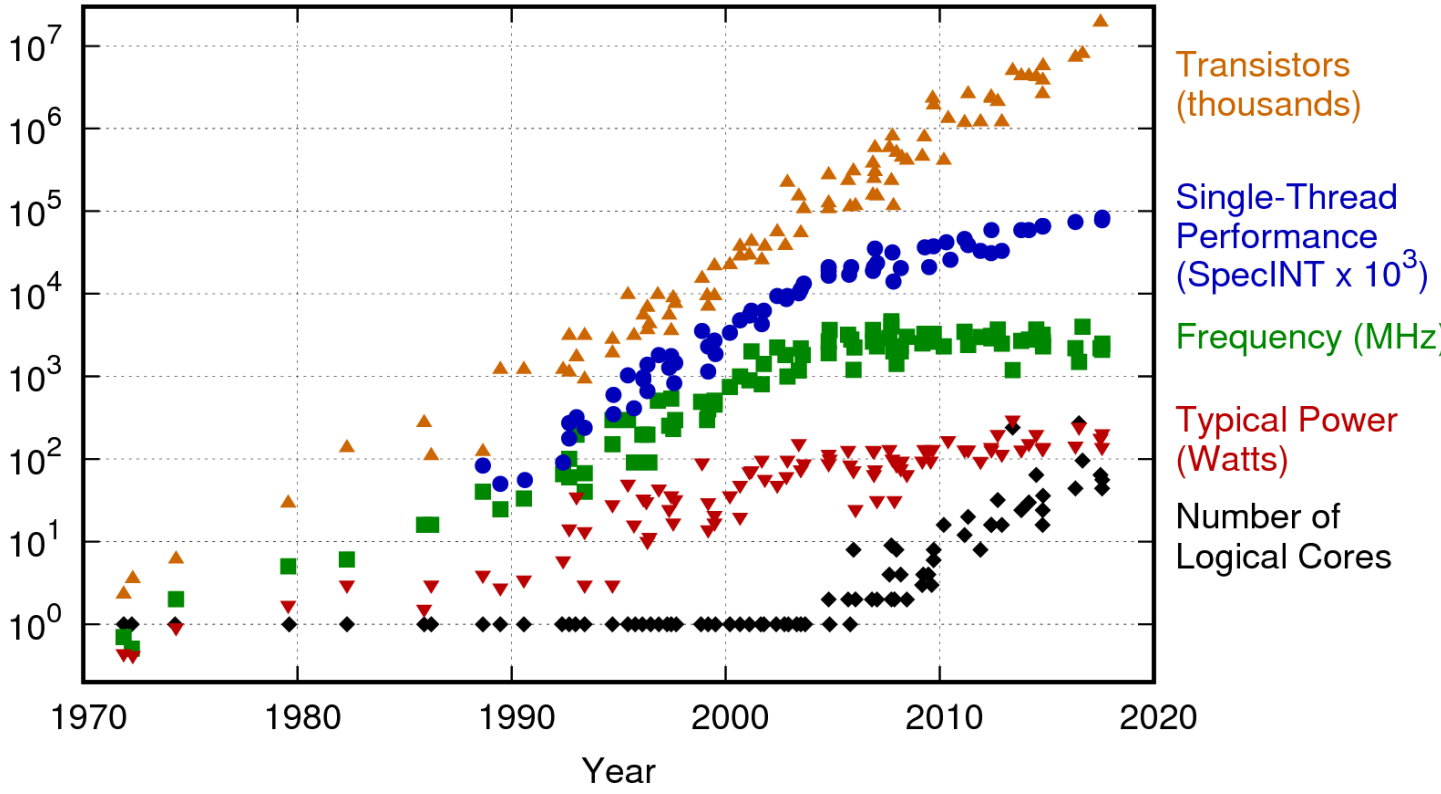
```
int main() {
   uint32_t matA[MATSIZE][MATSIZE];
   uint32_t matB[MATSIZE][MATSIZE];
   uint32_t matC[MATSIZE][MATSIZE];

   // Init matA and matB with numbers, matC with 0's
   fill(matA, matB, matC);

   int k;

   #pragma omp parallel for private(k) shared(matA, matB, matC) collapse(2)
   for (int i = 0; i < MATSIZE; i++)
     for (int j = 0; j < MATSIZE; j++)
       for(k = 0; k < MATSIZE; k++)
         matC[i][j] += matA[i][k] * matB[k][j];
   return EXIT_SUCCESS;
 }
```

	Time	
MATSIZE	Sequential	OpenMP
100	0.0035s	0.00044s
1.000	2.7s	0.34s
10.000	(~6200s)	~780s

parallel matrix multiplication

New technology

- CPU speeds don't increase as fast anymore



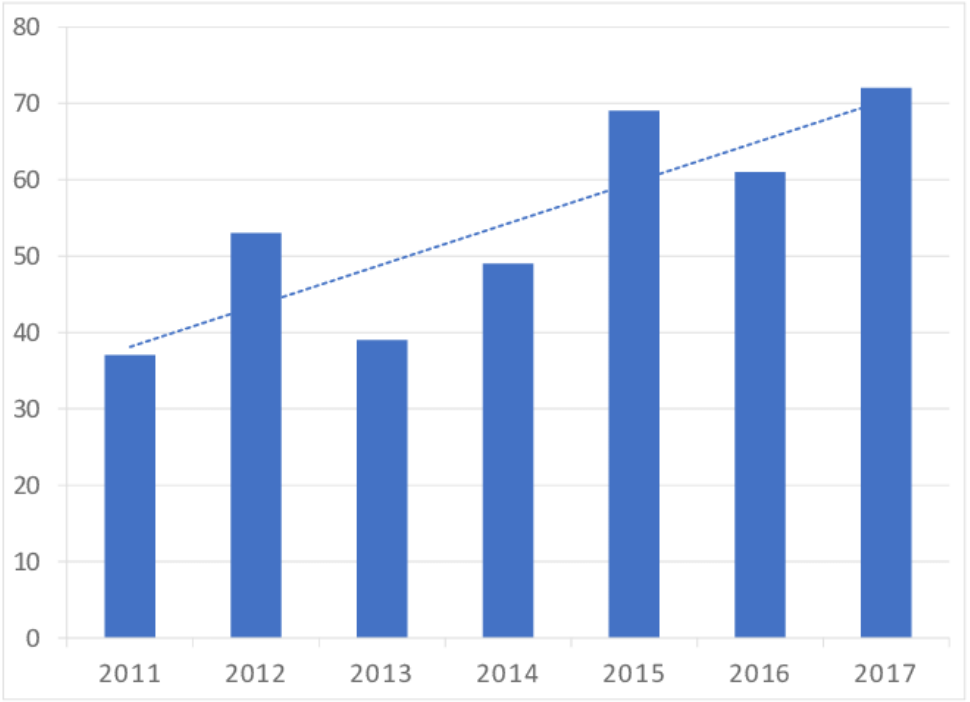
[4]

New technology

- CPU speeds don't increase as fast anymore [5]
- First widely used GPU's in the 80's [6]
- Now: GPU, Xeon Phi, AI-Coprocessors, …

- Promise great speedgains for certain applications

New technology



[7]

# of supercomputers in top 500 with accellerators

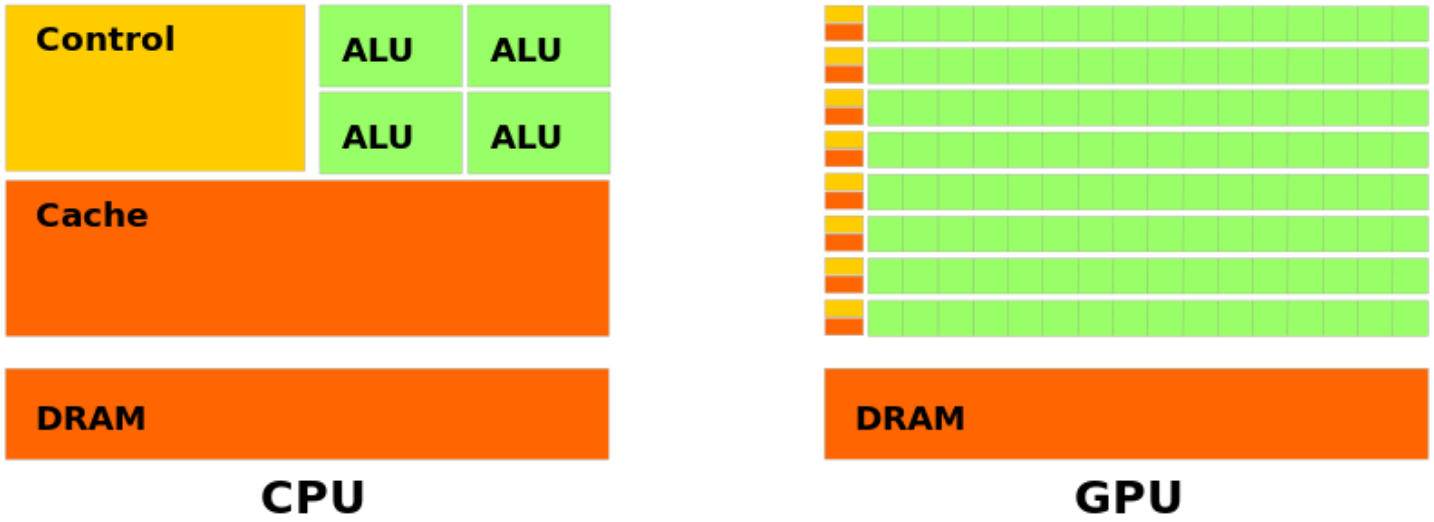
GPU-Offloading

GPU-Offloading - GPU

- Massive amount of cores
  - Up to 8192 + 64 (Nvidia A100) [8]
- Comparatively weak ISA [9]
- Lower clockspeeds

GPU-Offloading - GPU

- (Nvidia) GPU's divided into streaming multiprocessors (SM) [11]
  - Consist of registers, cache, schedulers and cuda cores
- e.g. my GTX 1080
  - 2560 cuda cores in 20 SM's -> 128 cuda cores per SM [12]



[10]

GPU-Offloading - Compiler

- Needs compiler with nvptx support
- Spack offers simple installation

  spack install gcc@10.2.0+nvptx

GPU-Offloading – execution

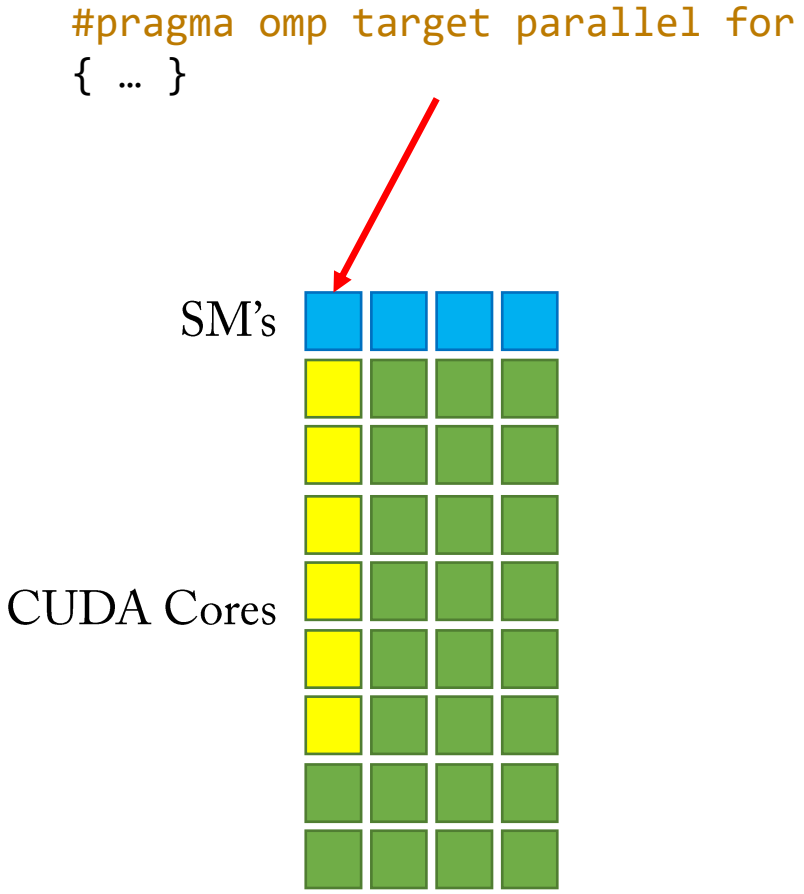
```
#pragma omp target …
{ … }
```

Automatically offloads computation

GPU-Offloading – execution

- SM's are mostly independent
- OpenMP threads are dependent

[1]

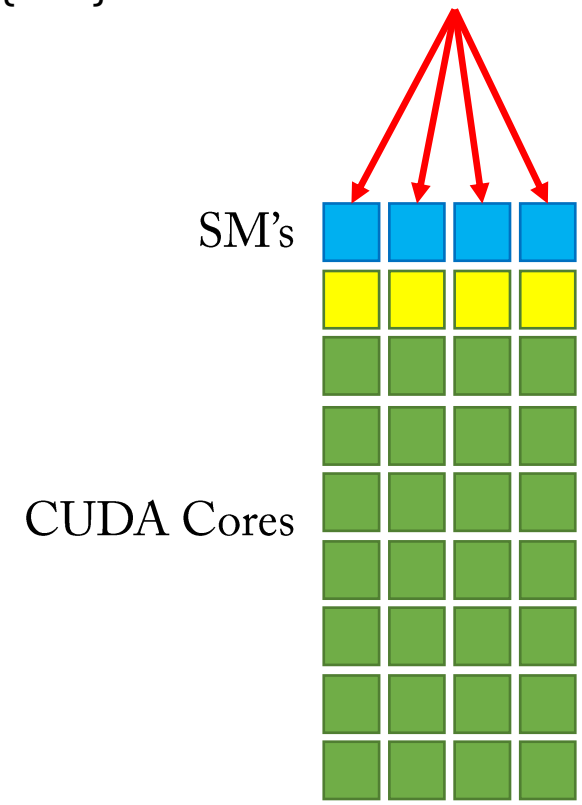


GPU-Offloading – execution

- SM's are mostly independent
- OpenMP threads are dependent
- teams creates independent threads

[1]

```
#pragma omp target teams parallel for …
{ … }
```

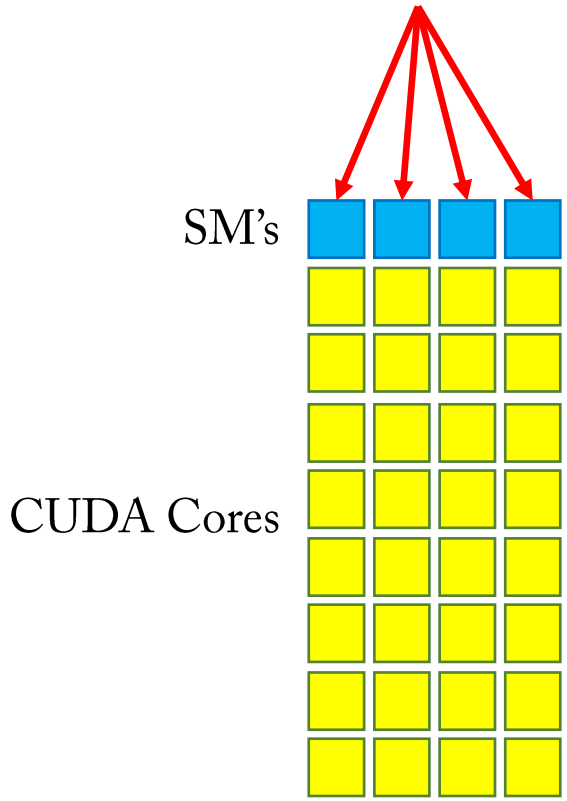


GPU-Offloading – execution

- SM's are mostly independent
- OpenMP threads are dependent
- teams creates independent threads
- distribute the iterations across cores

[1]

```
#pragma omp target teams distribute parallel for …
{ … }
```



GPU-Offloading – data mapping

- Different memory (RAM vs. VRAM)
  - Memory access between CPU and GPU is very expensive
  - Mapping required

GPU-Offloading – data mapping

```
#pragma omp target map(to: A[0:N][0:N], myint) map(tofrom: result)
{ ... }
```

```
#pragma omp target enter data map(to: A[0:N])
...
#pragma omp target exit data map(from: A)
```

Scalars and the pointed-to data are mapped by the compiler implicitly,
non-scalars like arrays are not!
Variables are shared per default across all threads

GPU-Offloading – data mapping

explicit mapping

```
uint32_t mat[N][N];
#pragma omp target map(tofrom: mat[0:N][0:N])
{ ... }
```

vs.

implicit mapping

```
std::array<std::array<uint32_t, N>, N> mat;
uint32_t *matP = mat.data();
#pragma omp target
{ ... matP ... }
```

Scalars and the pointed-to data are mapped by the compiler implicitly,
non-scalars like arrays are not!
Variables are shared per default across all threads

GPU-Offloading – example revisited

```
#define MATSIZE 10000
int main() {
  uint32_t matA[MATSIZE][MATSIZE];
  uint32_t matB[MATSIZE][MATSIZE];
  uint32_t matC[MATSIZE][MATSIZE];

  // Generating random values in matA and matB, initializing matC with 0s
  fill(matA, matB, matC);

  #pragma omp target data map(to: matA[0:MATSIZE][0:MATSIZE], matB[0:MATSIZE][0:MATSIZE]) \
                          map(tofrom: matC[0:MATSIZE][0:MATSIZE]) device(0)

  #pragma omp target teams distribute \
          parallel for shared(matA, matB, matC) collapse(2) device(0)
  for (int i = 0; i < MATSIZE; i++)
    for (int j = 0; j < MATSIZE; j++)
      for(int k = 0; k < MATSIZE; k++)
        matC[i][j]+= matA[i][k] * matB[k][j];
  return EXIT_SUCCESS;
}
```

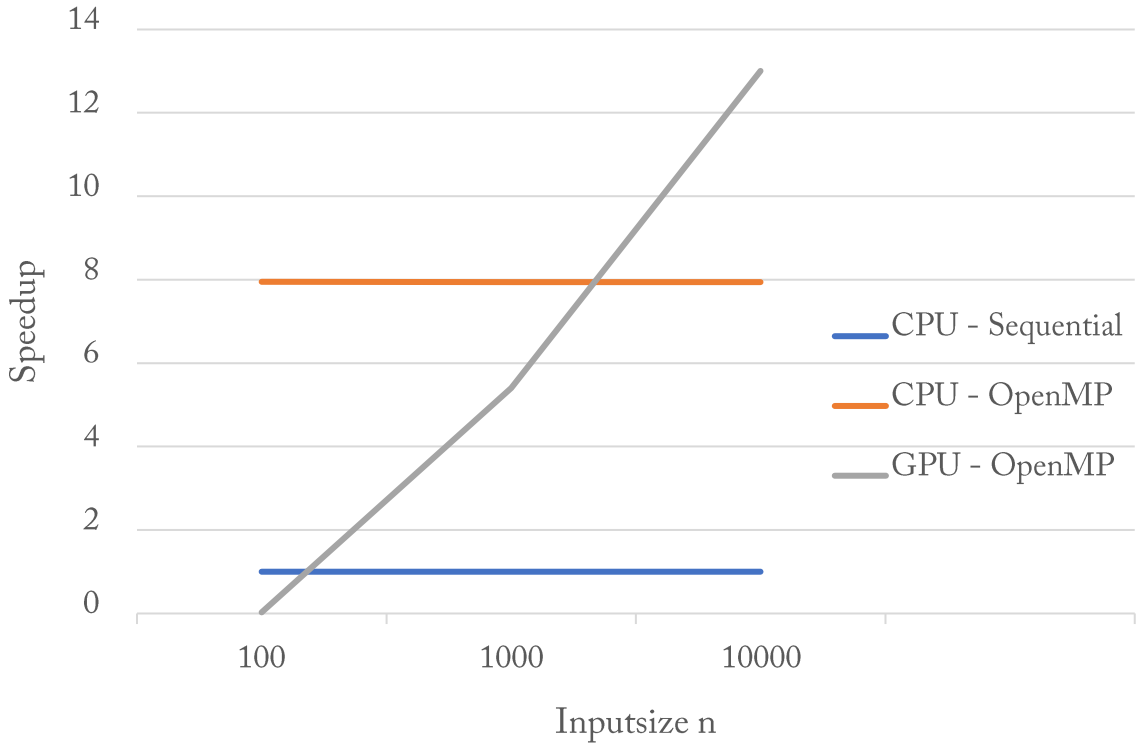
GPU-offloaded parallel matrix multiplication

GPU-Offloading – example revisited

	Time		
MATSIZE	Sequential	OpenMP CPU	OpenMP GPU
100	0.0035s	0.00044s	0.12s
1.000	2.7s	0.34s	0.5s
10.000	(~6201s)	~780s	480s

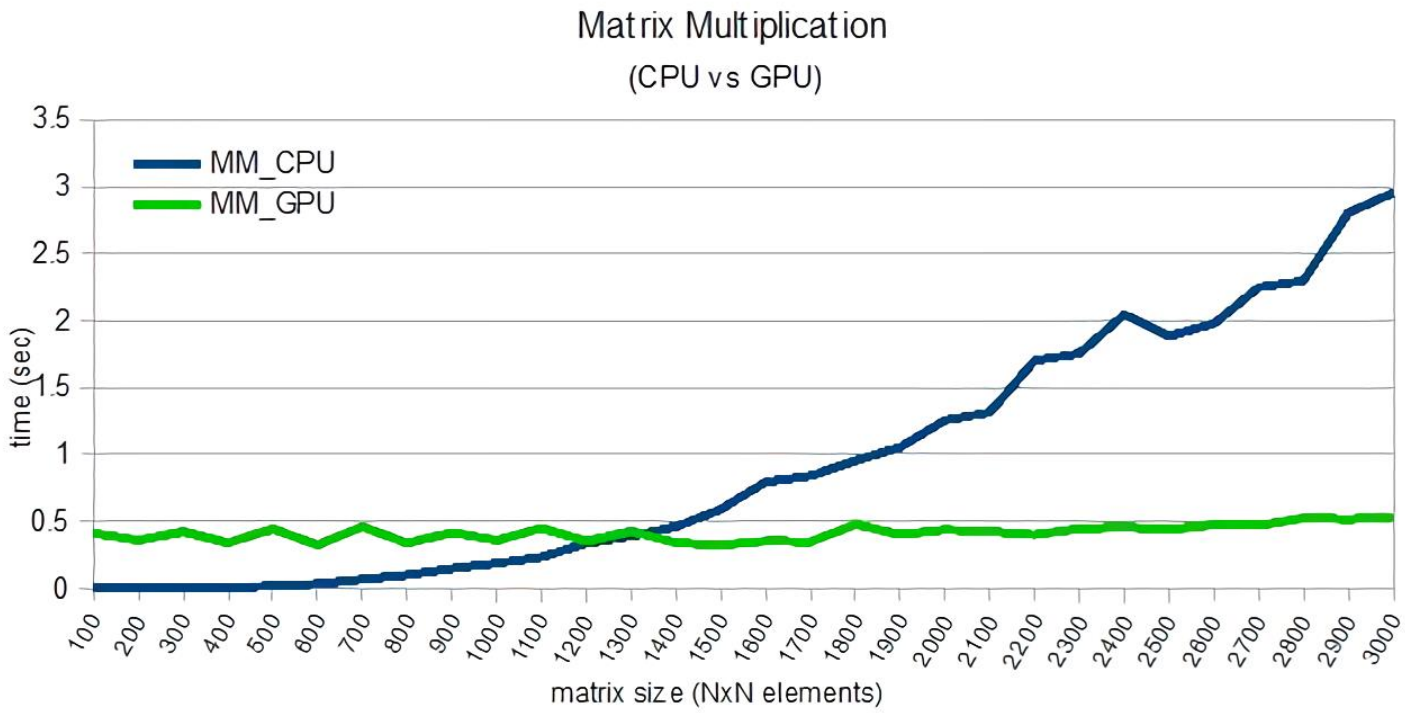
GPU-Offloading – example revisited

	Time		
MATSIZE	Sequential	OpenMP CPU	OpenMP GPU
100	0.0035s	0.00044s	0.12s
1.000	2.7s	0.34s	0.5s
10.000	(~6200s)	~780s	480s



GPU-Offloading – example revisited

IBM Results



[13]

IBM Power AC922
2x Intel Xeon: 44 cores / 88 threads
4x Nvidia Tesla V100: 20480 cuda cores

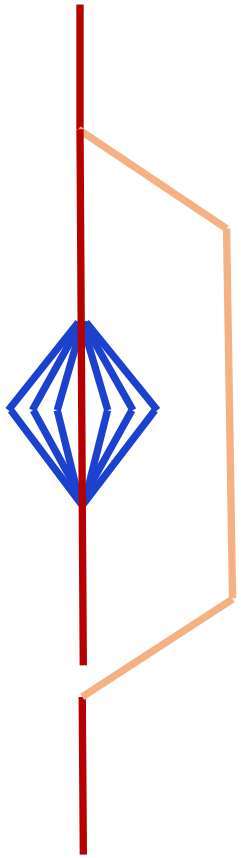
[14]

Image: https://developer.ibm.com/articles/gpu-programming-with-openmp/ (21.01.2021)

GPU-Offloading – asynchronous execution

```
#pragma omp … nowait

…

#pragma omp taskwait
```

GPU-Offloading – heterogeneous execution



```
…
// master thread creates asynchronous gpu task
#pragma omp target teams distribute parallel for ... nowait
for(...)
{ ... }

// a bunch of threads work
#pragma omp parallel for ...
for(...)
{ ... }

// wait for gpu task
#pragma omp taskwait

…
```

Summary

- OpenMP is an powerful API for parallel programming
- In certain scenarios GPU's are much faster than CPU's
- Simple offloading with just one directive:

```
#pragma omp target …
```

- To utilize all SM's and cores

```
#pragma omp target teams distribute parallel for …
```

- Data mapping is important, especially for arrays

```
#pragma omp target … map(tofrom: A[0:N])
```

Literature and recommendations

- The source for everything is [2] unless specified otherwise
- [1] OpenMP on GPUs, First Experiences and Best Practices by Jeff Larkin at GTC2018 Talk S8344 in March 2018 https://on-demand.gputechconf.com/gtc/2018/presentation/s8344-openmp-on-gpus-first-experiences-and-best-practices.pdf (21.01.2021)
- [2] OpenMP Specification: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf (21.01.2021)
- [3] OpenMP Introduction by University of Texas: https://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-basics.html (21.01.2021)
- [4] Moore's Law and other "trends": https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/ (21.01.2021)
- [5] Explanation for slow increase in CPU-speed: https://www.maketecheasier.com/why-cpu-clock-speed-isnt-increasing/ (21.01.2021)
- [6] History of GPU's: https://xoticpc.com/blogs/news/history-of-gpus (21.01.2021)
- [7] Siavashi, Ahmad & Momtazpour, Mahmoud. (2019). GPUCloudSim: an extension of CloudSim for modeling and simulation of GPUs in cloud data centers. The Journal of Supercomputing. 75. 10.1007/s11227-018-2636-7.
- [8] Nvidia A100 specsheet: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf (21.01.2021)
- [9] CPU vs. GPU differences: https://www.omnisci.com/technical-glossary/cpu-vs-gpu (21.01.2021)
- [10] Visualization of main differences: https://commons.wikimedia.org/wiki/File:Cpu-gpu.svg/ (21.01.2021)
- [11] Deeper explanation of CUDA programming, mainly SM's in this chapter: https://www.informit.com/articles/article.aspx?p=2103809 (21.01.2021)
- [12] Nvidia GTX 1080 specsheet https://www.nvidia.com/en-in/geforce/products/10series/geforce-gtx-1080/ (21.01.2021)
- [13] IBM OpenMP GPU programming: https://developer.ibm.com/articles/gpu-programming-with-openmp/ (21.01.2021) https://developer.ibm.com/technologies/systems/articles/gpu-programming-with-openmp-part-2/ (21.01.2021)
- [14] IBM AC922 specsheet: https://www.ibm.com/downloads/cas/EPNDE9D0 (21.01.2021)

Recommendations:

- OpenMP: The Next Step by Ruud van der Pas et. al. (ISBN: 9780262534789)
- OpenMP Examples: https://www.openmp.org/wp-content/uploads/openmp-examples-5.0.0.pdf (21.01.2021)