Computational Science on Many-Core Architectures

360.252

Karl Rupp

Institute for Microelectronics
Vienna University of Technology
https://www.iue.tuwien.ac.at/

Zoom Channel 95028746244
Wednesday, November 4, 2020

Agenda for Today

Exercise 2 Recap

Performance Modeling

Benchmarks: Thread Block and Grid Sizes

Sparse Matrices - Intro

Exercise 3

Exercise 2 Recap

https://xkcd.com/138/

Exercise 2 Recap

Feedback Time

- How was your experience?

Exercise 2 Recap

Feedback Time

- How was your experience?
- Log-log plots preferred for time vs. problem size

Exercise 2 Recap

Feedback Time

- How was your experience?
- Log-log plots preferred for time vs. problem size
- 100k vector entries still **tiny** for a GPU

Exercise 2 Recap

Feedback Time

- How was your experience?
- Log-log plots preferred for time vs. problem size
- 100k vector entries still **tiny** for a GPU
- Legal notice: I’m not a *Professor*

Exercise 2 Recap

Feedback Time

- How was your experience?
- Log-log plots preferred for time vs. problem size
- 100k vector entries still **tiny** for a GPU
- Legal notice: I'm not a *Professor*

Okay'ish Way to Get Timings

```
cudaDeviceSynchronize();
timer.reset();
    /* TIMED SECTION HERE */
cudaDeviceSynchronize();
double time_elapsed = timer.get();  // in seconds
```

Exercise 2 Recap

Better Way to Get Timings: Average

```
cudaDeviceSynchronize();
timer.reset();
for (int reps=0; reps < MAGIC_NUMBER; ++reps) {
    /* TIMED SECTION HERE */
}
cudaDeviceSynchronize();
double time_elapsed = timer.get() / MAGIC_NUMBER;
```

Exercise 2 Recap

Better Way to Get Timings: Average

```
cudaDeviceSynchronize();
timer.reset();
for (int reps=0; reps < MAGIC_NUMBER; ++reps) {
    /* TIMED SECTION HERE */
}
cudaDeviceSynchronize();
double time_elapsed = timer.get() / MAGIC_NUMBER;
```

Even Better Way to Get Timings: Median

```
cudaDeviceSynchronize();
timer.reset();
std::vector<double> timings;
for (int reps=0; reps < MAGIC_NUMBER; ++reps) {
    /* TIMED SECTION HERE */
  cudaDeviceSynchronize();
  timings.push(timer.get());
}
std::sort(timings.begin(), timings.end());
double time_elapsed = timings[MAGIC_NUMBER/2];
```

Performance Modeling

Latency

- Bottleneck in strong scaling limit
- Ultimate limit for time stepping

Latency - Sources

- Network latency (Ethernet $\sim 20\mu$s, Infiniband $\sim 5\mu$s)
- PCI-Express latency (Kernel launches, $\sim 10\mu$s)
- Thread synchronization (barriers, locks, $\sim 1 - 10\mu$s)
- Memory latency (~ 100ns)

Performance Modeling

Load Imbalance

- Total execution time determined by slowest thread
- Focus on making the slowest thread fast
- Easy for static data structures (e.g. dense matrices)
- Hard for dynamic data structures (e.g. sparse matrices)

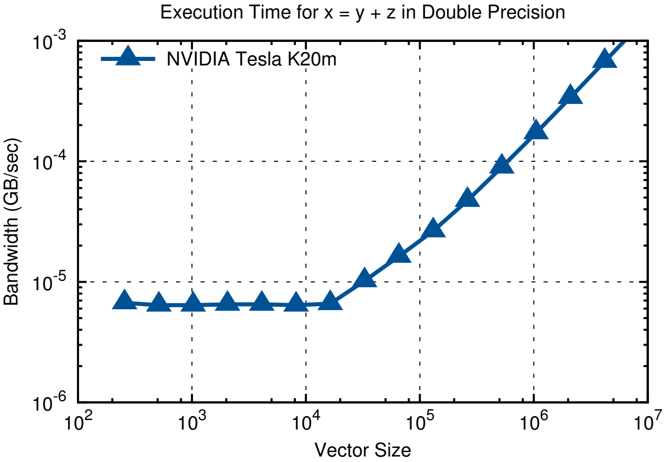
Amdahl's Law

- Total execution time T_{total} given by time spent in serial and parallel parts
- $T_{\text{total}} = T_{\text{serial}} + T_{\text{parallel}}/\#\text{processors}$
- Speed-up limited by serial portion of an algorithm

Performance Modeling: Vector Addition

Vector Addition

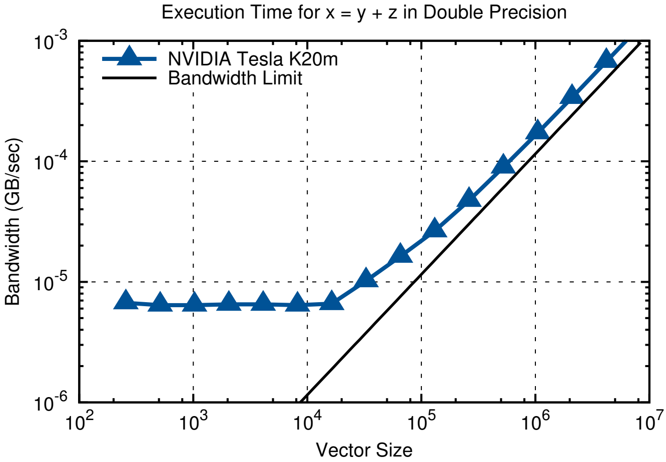
- $x = y + z$ with N elements each
- 1 FLOP per 24 byte in double precision
- Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



Performance Modeling: Vector Addition

Vector Addition

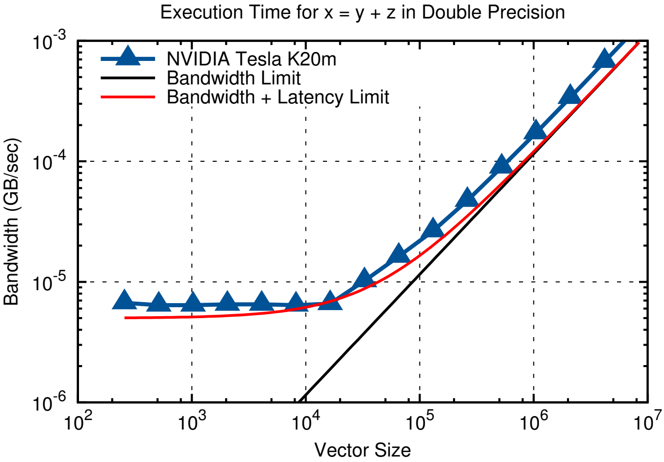
- $x = y + z$ with N elements each
- 1 FLOP per 24 byte in double precision
- Limited by memory bandwidth $\Rightarrow T_2(N) \stackrel{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



Performance Modeling: Vector Addition

Vector Addition

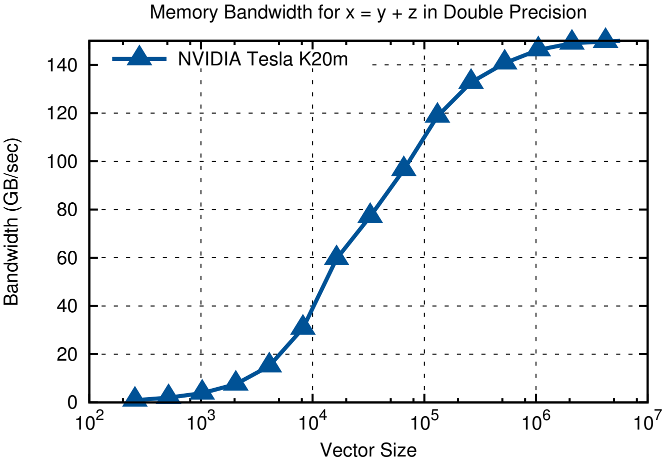
- $x = y + z$ with N elements each
- 1 FLOP per 24 byte in double precision
- Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



Performance Modeling: Vector Addition

Vector Addition

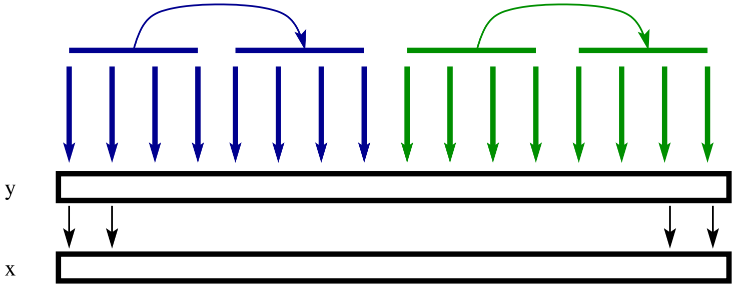
- $x = y + z$ with N elements each
- 1 FLOP per 24 byte in double precision
- Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



Benchmark Setting

Vector Assignment (Copy) Kernel

- $x \Leftarrow y$ for (large) vectors x, y



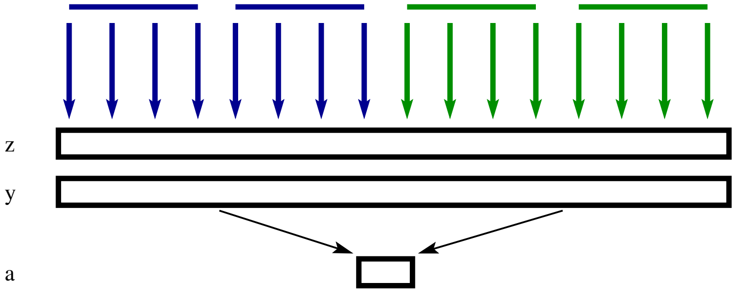
Parameters (1900 variations)

```
for (size_t i = group_start + threadIdx.x;
            i < group_end; i+= blockDim.x)
   x[i] = y[i];
```

Benchmark Setting

Operations

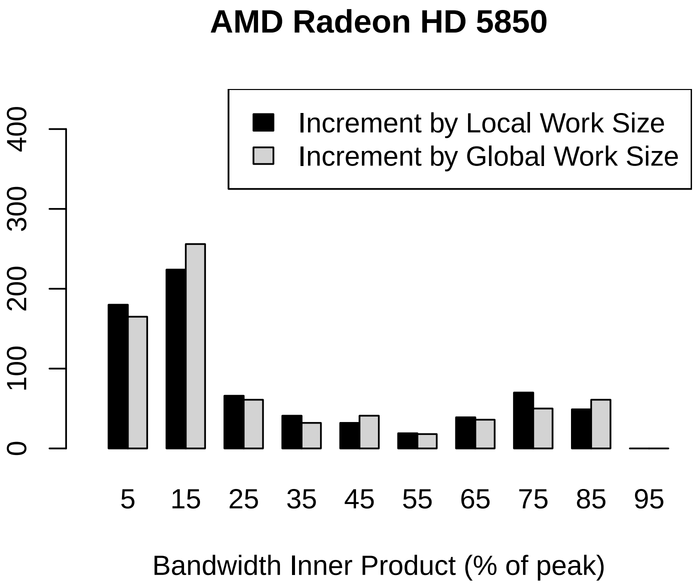
- Vector copy, vector addition, inner product
- Matrix-vector product



Devices

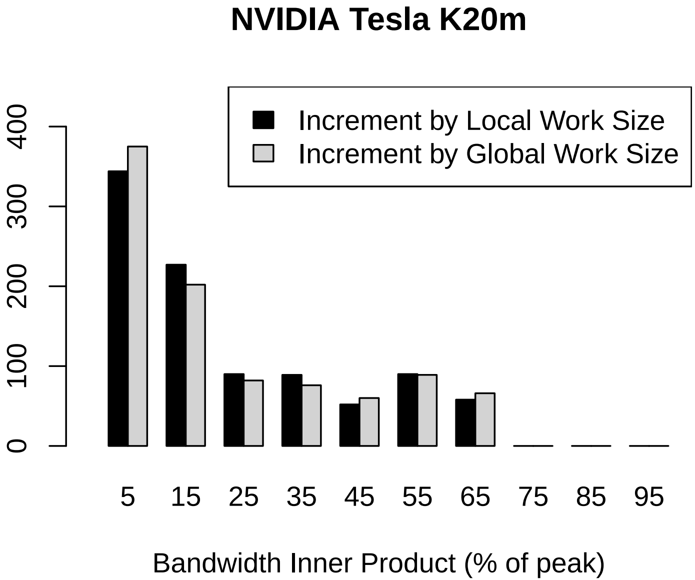
- AMD: HD 5850 GPU
- INTEL: Dual Socket Xeon E5-2670
- NVIDIA: GTX 285, Tesla K20m

Benchmark



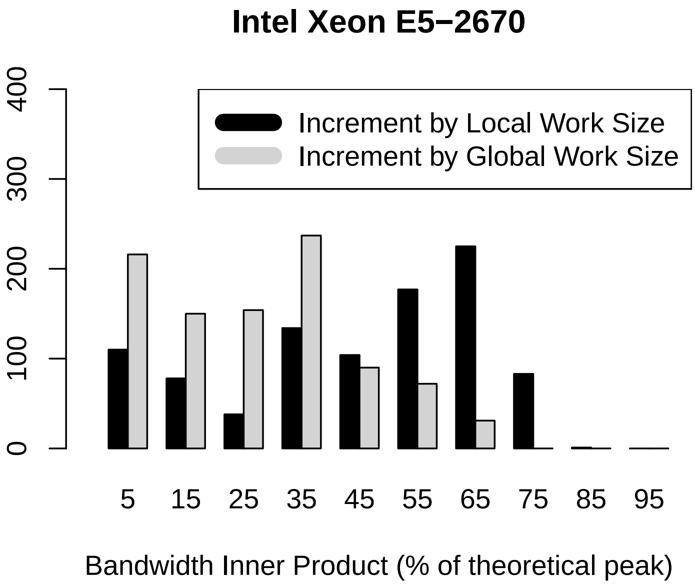
AMD Radeon HD 5850
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of peak)

Benchmark



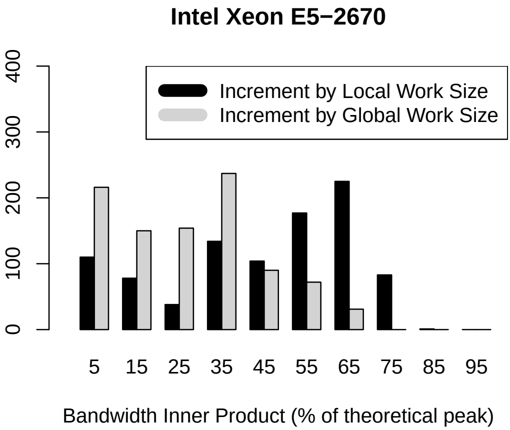
NVIDIA Tesla K20m
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of peak)

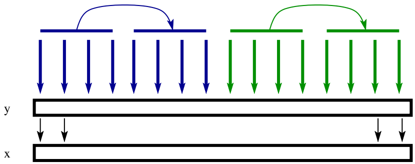
Benchmark



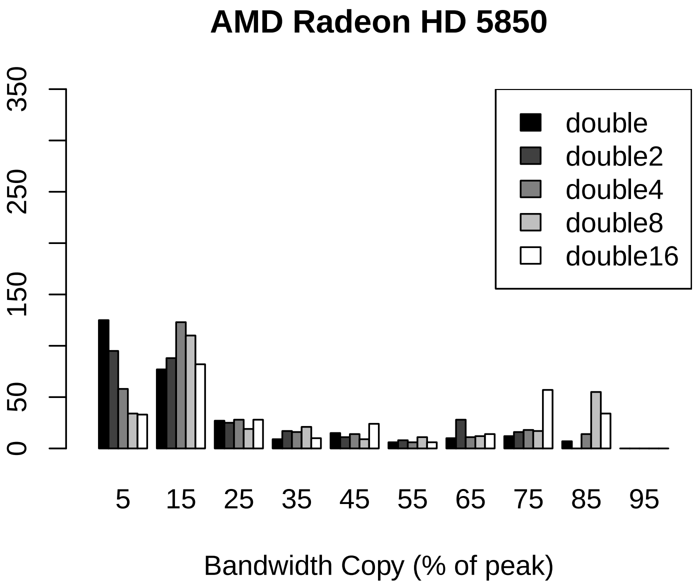
Intel Xeon E5−2670
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of theoretical peak)

Benchmark

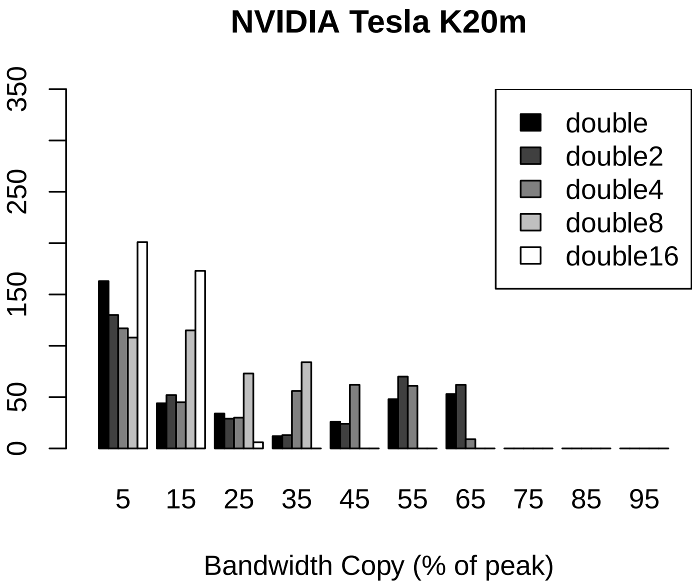

Intel Xeon E5−2670
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of theoretical peak)


y
x

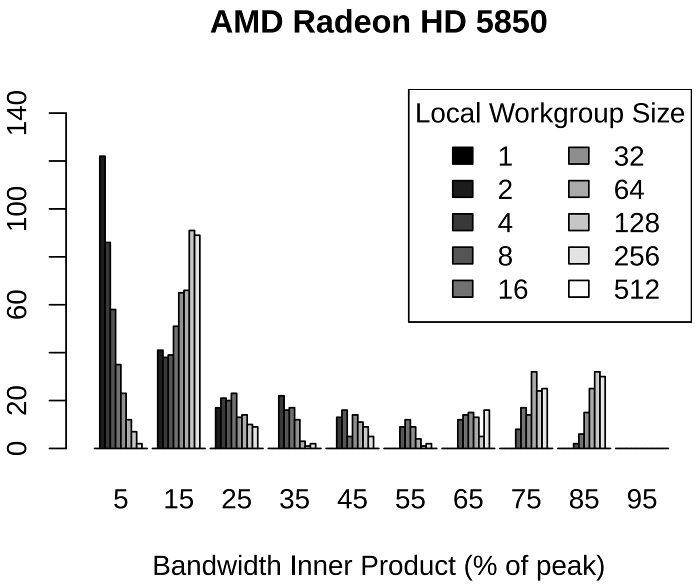
Benchmark


AMD Radeon HD 5850
0
50
150
250
350
double
double2
double4
double8
double16
5
15
25
35
45
55
65
75
85
95
Bandwidth Copy (% of peak)

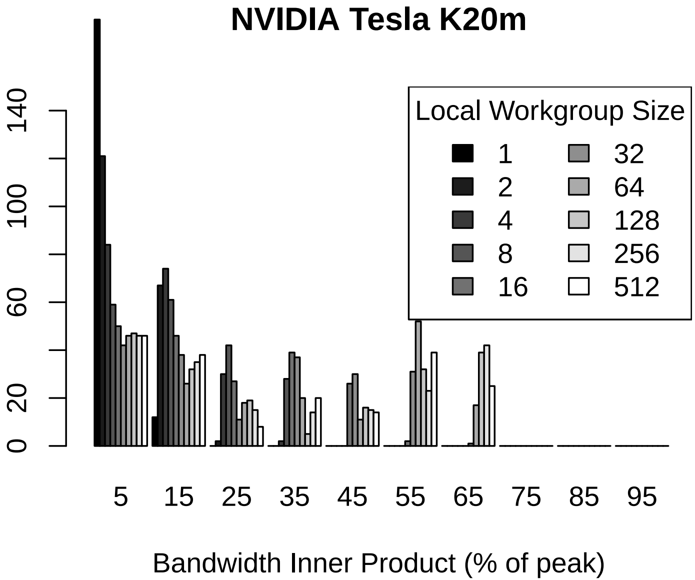
Benchmark


NVIDIA Tesla K20m
0
50
150
250
350
double
double2
double4
double8
double16
5
15
25
35
45
55
65
75
85
95
Bandwidth Copy (% of peak)

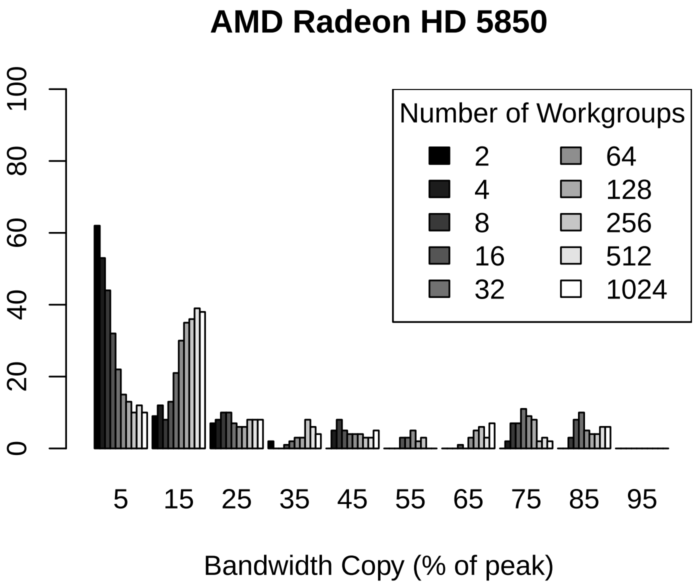
Benchmark


AMD Radeon HD 5850
Local Workgroup Size
1
2
4
8
16
32
64
128
256
512
0
20
60
100
140
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of peak)

Benchmark

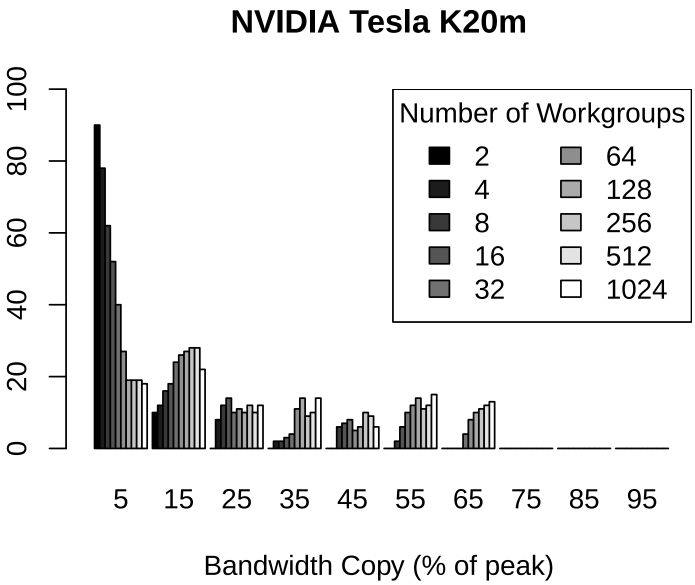

NVIDIA Tesla K20m
Local Workgroup Size
1
2
4
8
16
32
64
128
256
512
0
20
60
100
140
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of peak)

Benchmark

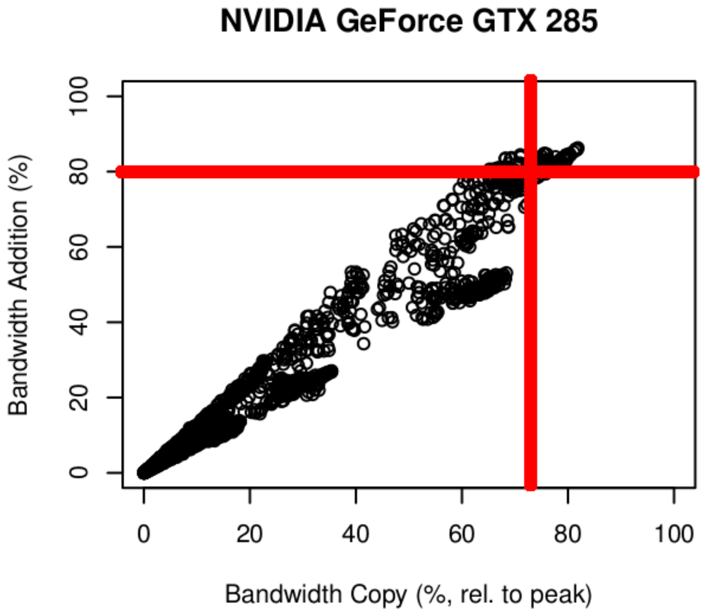


AMD Radeon HD 5850
Number of Workgroups
2
4
8
16
32
64
128
256
512
1024
0
20
40
60
80
100
5
15
25
35
45
55
65
75
85
95
Bandwidth Copy (% of peak)

Benchmark


NVIDIA Tesla K20m
Number of Workgroups
2
4
8
16
32
64
128
256
512
1024
0
20
40
60
80
100
5
15
25
35
45
55
65
75
85
95
Bandwidth Copy (% of peak)

Benchmark



NVIDIA GeForce GTX 285
Bandwidth Addition (%)
0
20
40
60
80
100
0
20
40
60
80
100
Bandwidth Copy (%, rel. to peak)

Benchmark

Conclusio:

Focus on fastest configurations for copy-kernel often sufficient

Sparse Matrices - Intro

Sparse Matrices

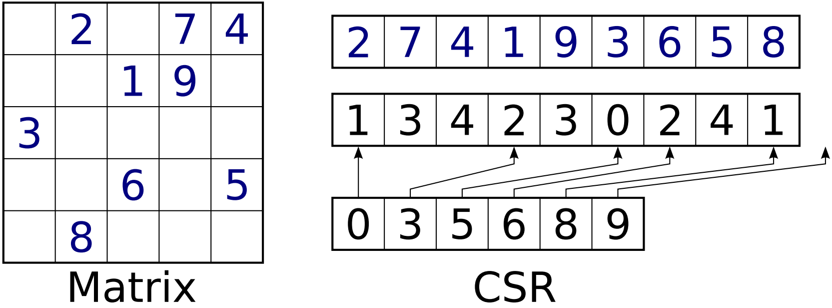
- Ubiquituous for: graph algorithms, numerical solution of PDEs
- Finite differences, finite elements, finite volumes, etc.

Algebraic Multigrid

- Asymptotically optimal solver
- Computation of coarse grid operator $\mathbf{A}^{\text{coarse}} = \mathbf{R}\mathbf{A}^{\text{fine}}\mathbf{P}$ expensive

Sparse Matrices - Intro

Compressed Sparse Row Format



Three Arrays

- Nonzero Values
- Column Indices
- Offset array for each row (typically size N+1)

Sparse Matrices - Intro

Typical Kernel for $y = Ax$

```
__global void csr_matvec(int N,
  int *rowoffsets, int *colindices, double *values, //CSR arrays
  double const *x, double *y) {

 for (int row = blockDim.x * blockIdx.x + threadIdx.x;
          row < N;
          row += gridDim.x * blockDim.x) {
   double val = 0;
   for (int jj = rowoffsets[i]; jj < rowoffsets[i+1]; ++jj) {
      val += values[jj] * x[colindices[jj]];
   }
   y[row] = val;
 }
}
```

- One thread per row
- Good starting point, but not the fastest option

Exercises

Environment

- `https://gtx1080.360252.org/2020/ex3/`
- (Might receive visual updates and additional hints over the next days)
- Due: Tuesday, November 10, 2020 at 23:59pm

Hints and Suggestions

- Consider version control for locally developed code
- Please let me know of any bugs or issues
- Example codes and code skeletons provided at URL above