# Performance Portability Study of Linear Algebra Kernels in OpenCL

Karl Rupp[1,2], Philippe Tillet[1], Florian Rudolf[1], Josef Weinbub[1], Ansgar Jüngel[2], Tibor Grasser[1]

rupp@iue.tuwien.ac.at

@karlrupp

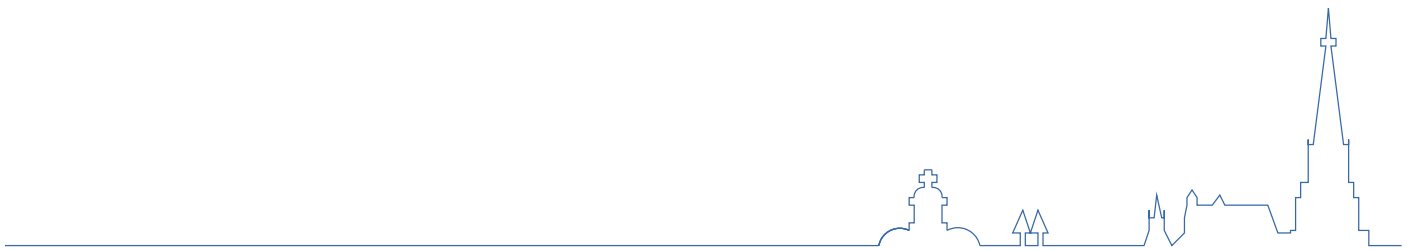
[1] Institute for Microelectronics, TU Wien, Austria
[2] Institute for Analysis and Scientific Computing, TU Wien, Austria

International Workshop on OpenCL 2014

May 13th, 2014

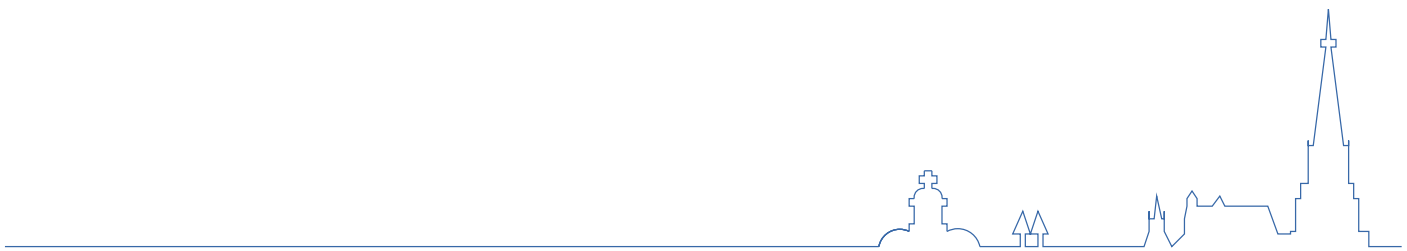
# Introduction

## OpenCL Code Library Expectations



## OpenCL Code Library Expectations

- Just work on my (user) hardware
- Run fast on my (user) hardware
- Integrate easily into my (user) code
- Mix with CUDA and OpenMP
- Available for free



# Introduction

## OpenCL Code Library Expectations

Just work on my (user) hardware

Run fast on my (user) hardware

Integrate easily into my (user) code

Mix with CUDA and OpenMP

Available for free

## Work Based on Developer Experiences


http://viennacl.sourceforge.net/

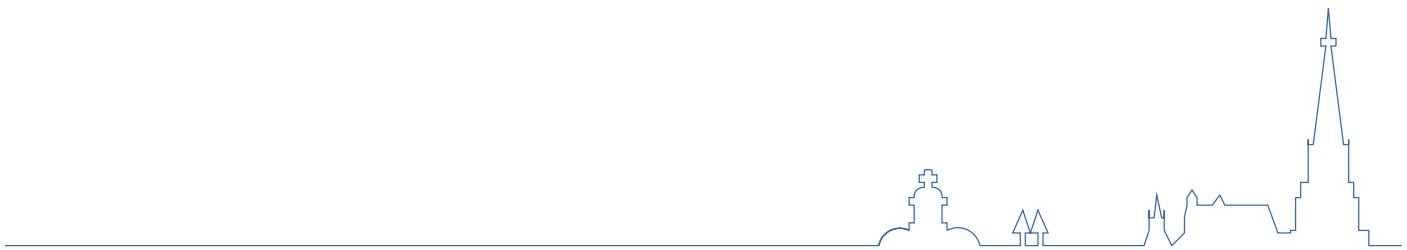
Free open-source C++ linear algebra library (Python? PyViennaCL!)

API similar to Boost.uBLAS

# Performance Portability

## How OpenCL Helps

- Unified hardware information queries
- Unified kernel language



# Performance Portability

## How OpenCL Helps

Unified hardware information queries

Unified kernel language

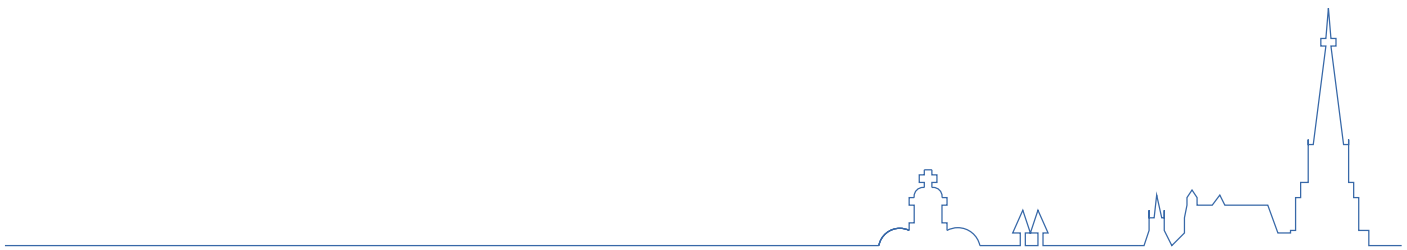
## Developer Tasks

OpenCL $\neq$ automatic performance portability

Parameterize each kernel

Performance models vs. (auto)tuning

Impractical to repeat for each kernel



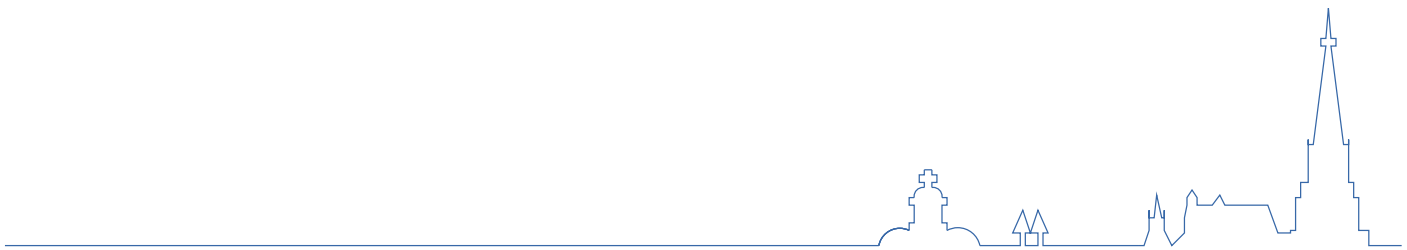
# Benchmark Setting

## Scope for Portability Study

Vector and matrix-vector operations (BLAS levels 1 and 2)

Limited by memory bandwidth

Matrix-matrix-multiplication (only briefly today)



# Benchmark Setting

## Scope for Portability Study

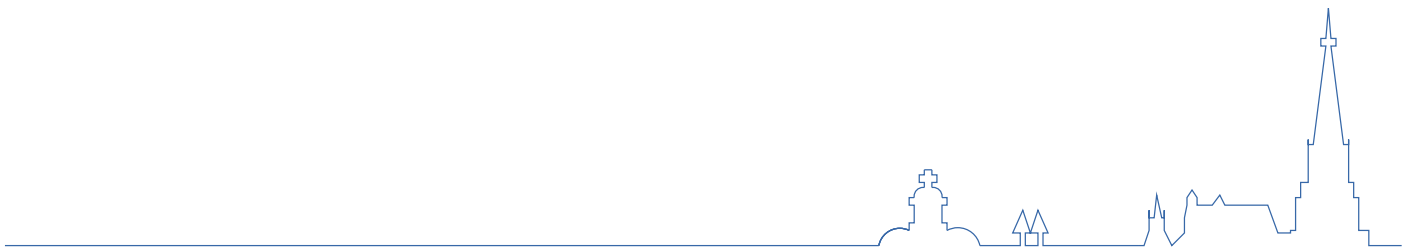
Vector and matrix-vector operations (BLAS levels 1 and 2)

Limited by memory bandwidth

Matrix-matrix-multiplication (only briefly today)

## Key Question (Memory-Bandwidth-Limited Kernels)

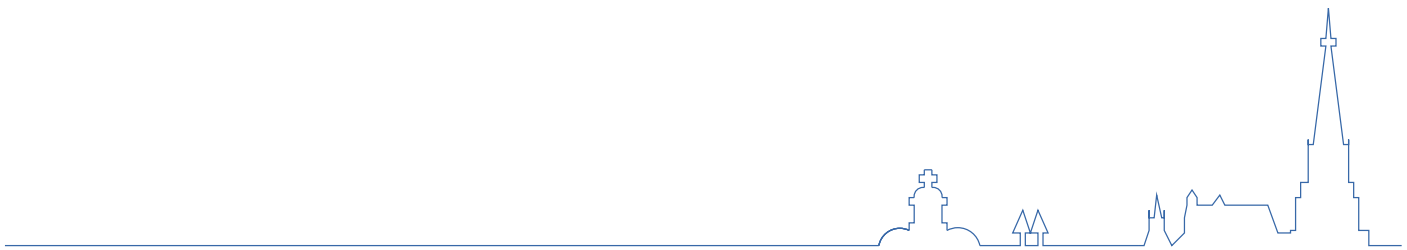
Good performance of complicated kernels
by optimizing the simplest kernel?



# Benchmark Setting

## Vector Assignment (Copy) Kernel

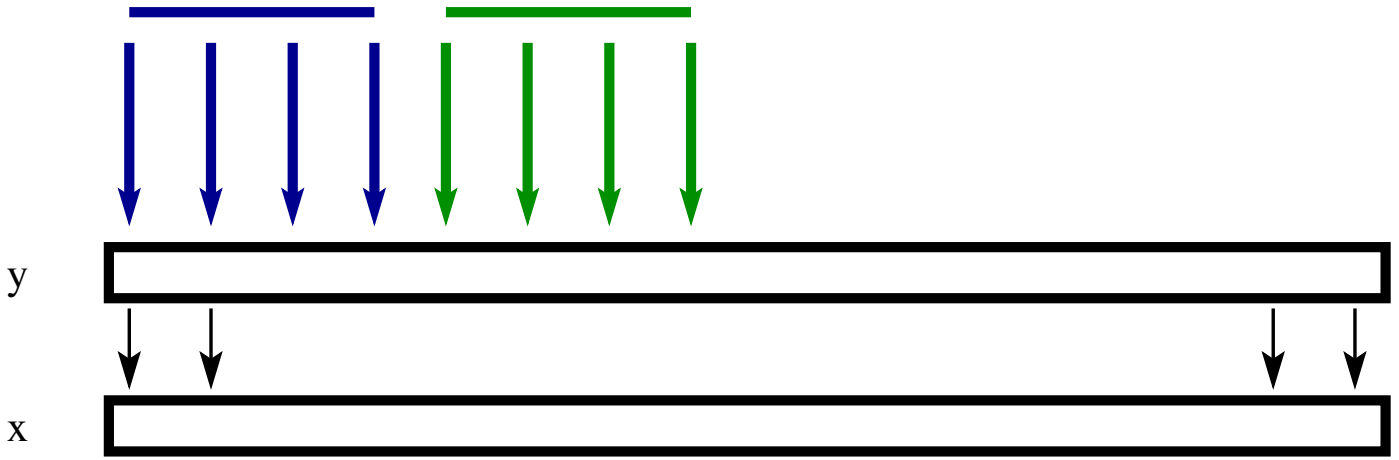
$x \Leftarrow y$ for (large) vectors $x$, $y$



# Benchmark Setting

## Vector Assignment (Copy) Kernel

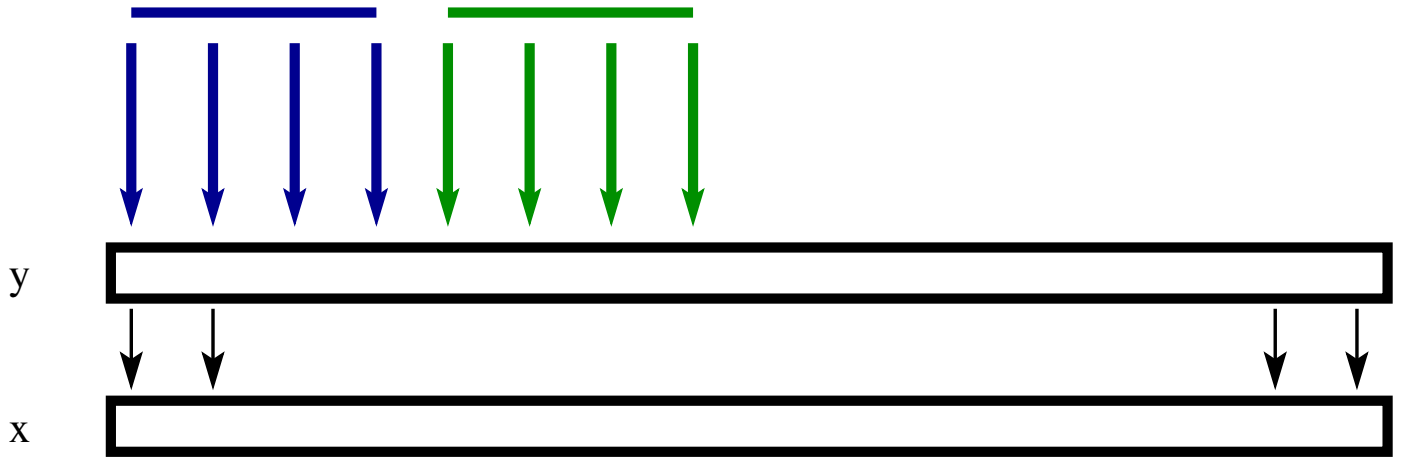
$x \Leftarrow y$ for (large) vectors $x$, $y$



# Benchmark Setting

## Vector Assignment (Copy) Kernel

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




## Parameters (1900 variations)

Local work size, global work size

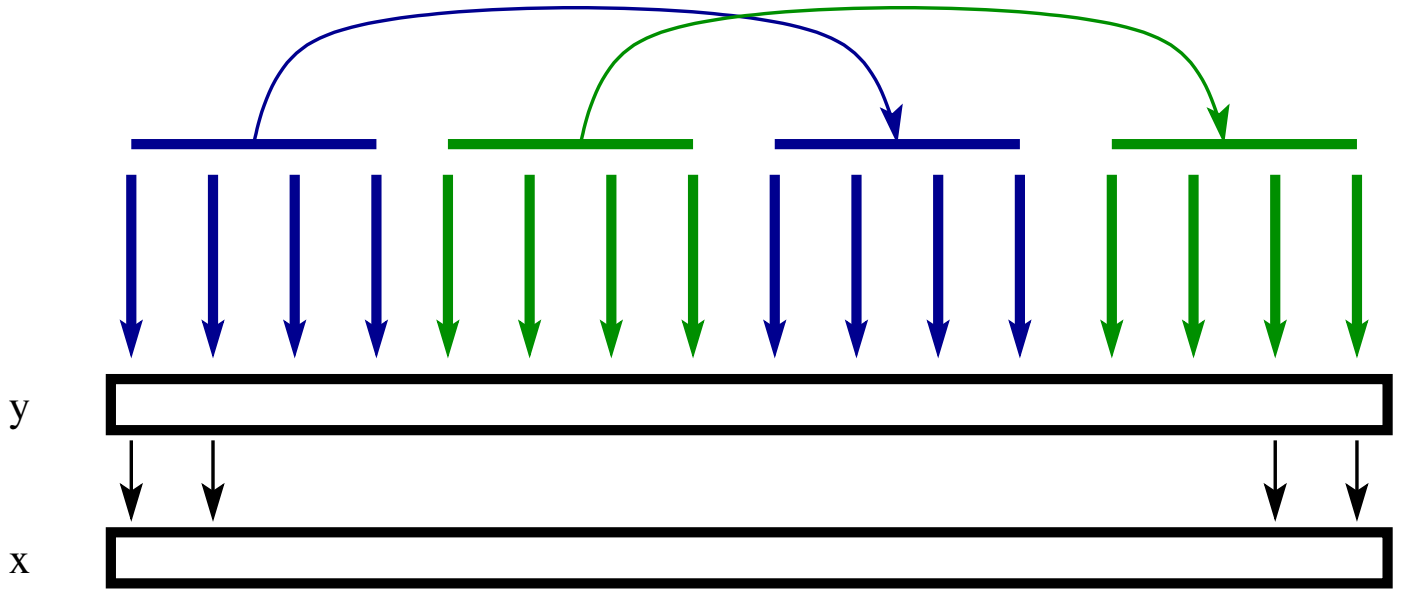
Vector types (float1, float2, ... , float16)

Thread increment type

# Benchmark Setting

## Vector Assignment (Copy) Kernel

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



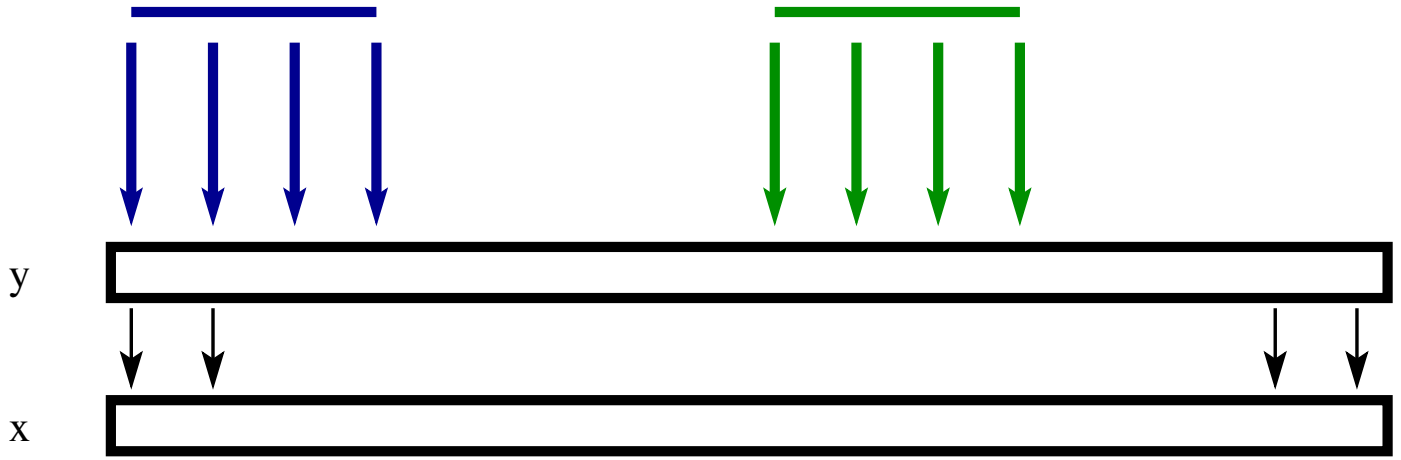

## Parameters (1900 variations)

```
for (size_t i = get_global_id(0); i < N;
                i+= get_global_size(0))
  x[i] = y[i];
```

# Benchmark Setting

## Vector Assignment (Copy) Kernel

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



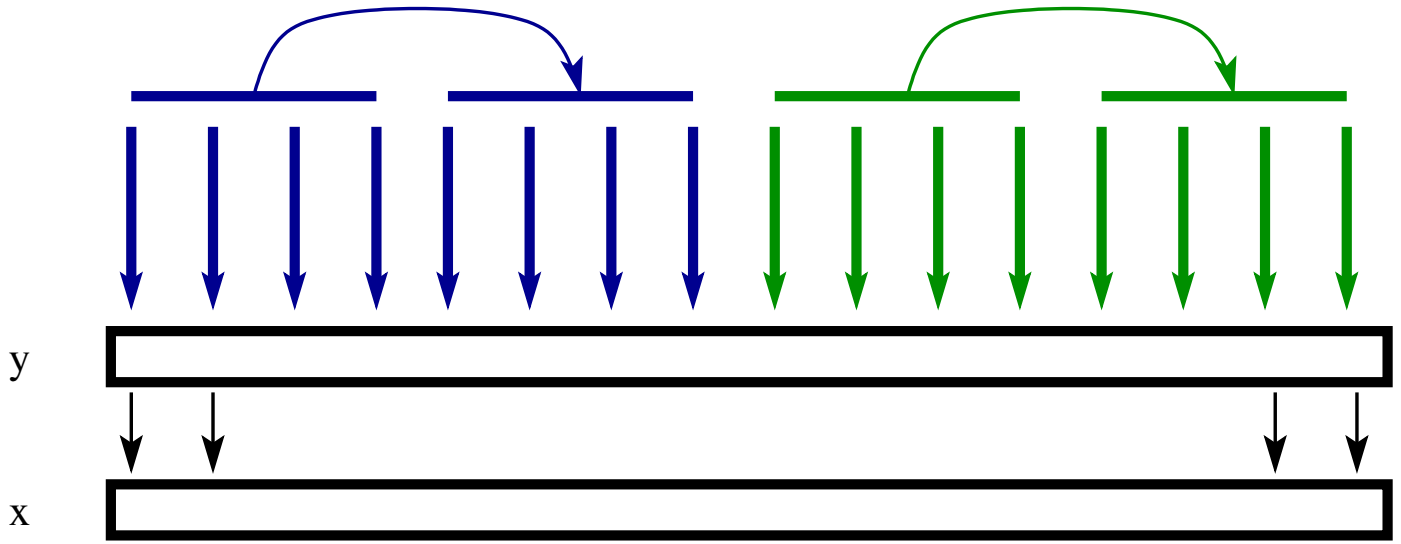

## Parameters (1900 variations)

```
for (size_t i = group_start + get_local_id(0);
                i < group_end;
  i+= get_local_size(0)) x[i] = y[i];
```

# Benchmark Setting

## Vector Assignment (Copy) Kernel

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




## Parameters (1900 variations)

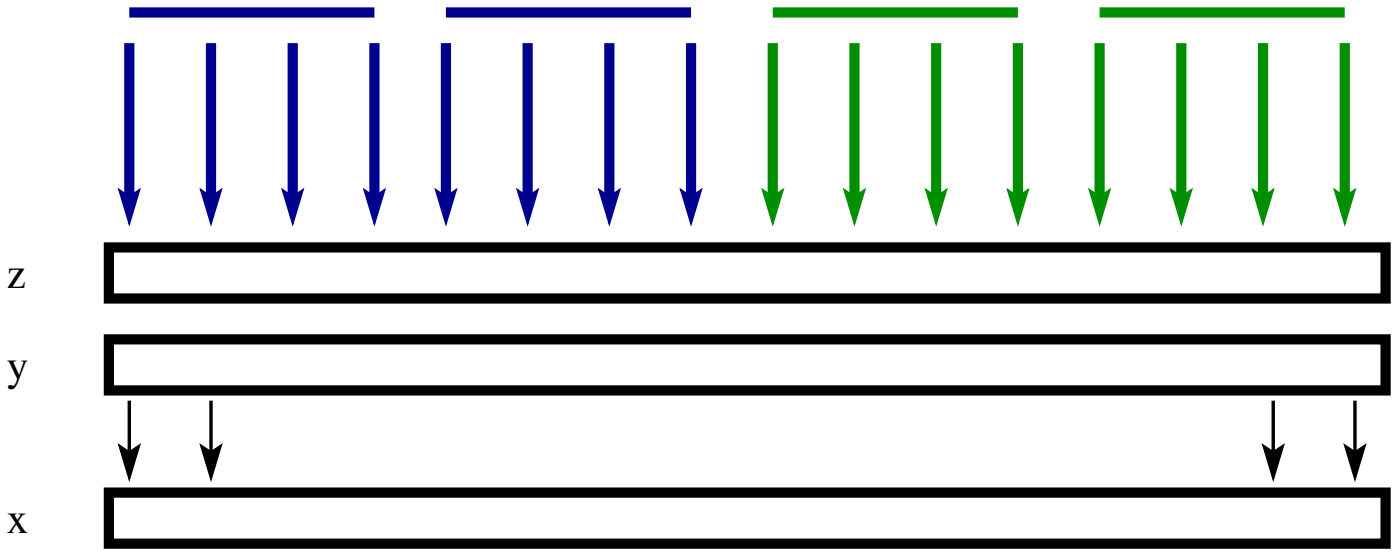
```
for (size_t i = group_start + get_local_id(0);
     i < group_end; i+= get_local_size(0))
  x[i] = y[i];
```

# Benchmark Setting

## Operations

Vector copy, vector addition, inner product

Matrix-vector product

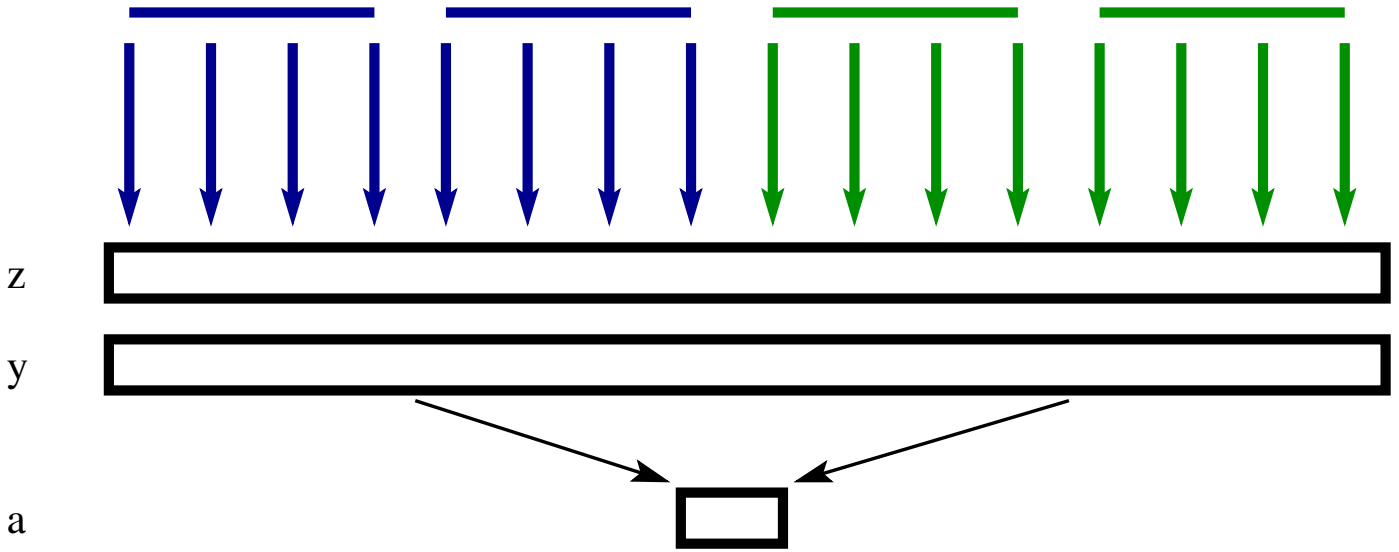


# Benchmark Setting

## Operations

Vector copy, vector addition, inner product

Matrix-vector product

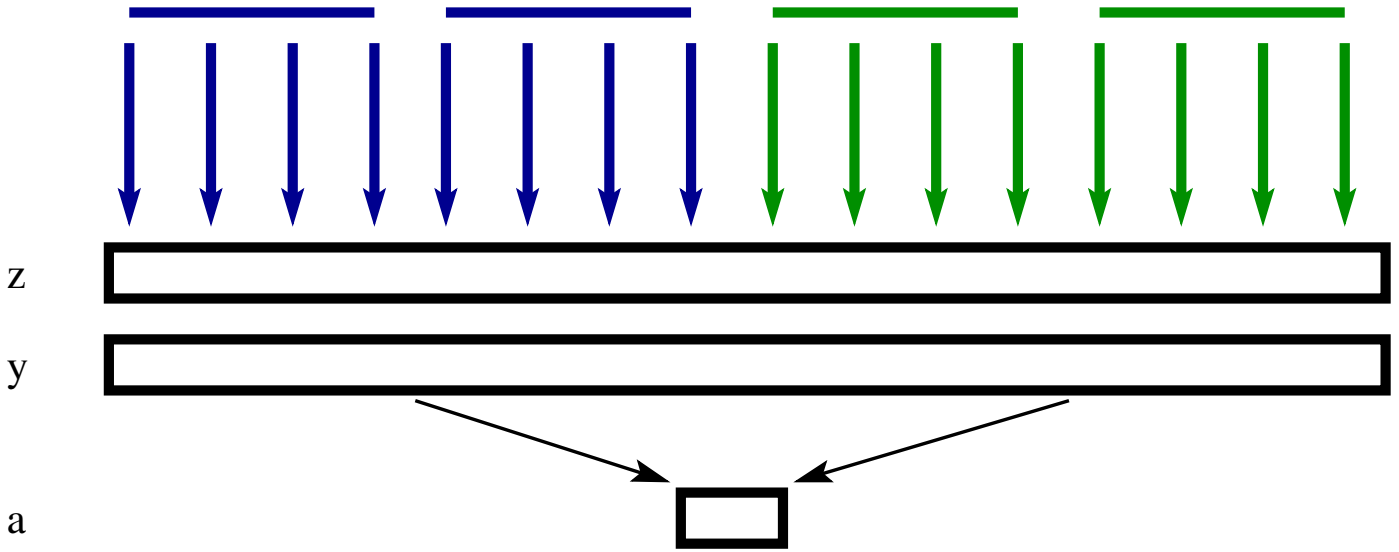


# Benchmark Setting

## Operations

Vector copy, vector addition, inner product

Matrix-vector product



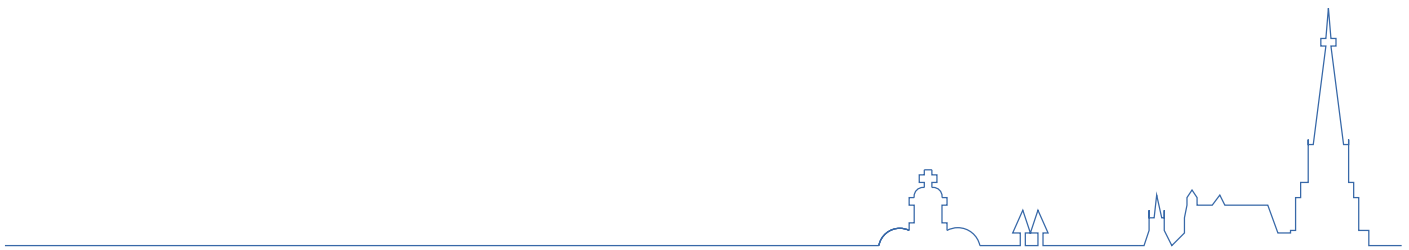

## Devices

AMD: A10-5800 APU, HD 5850 GPU

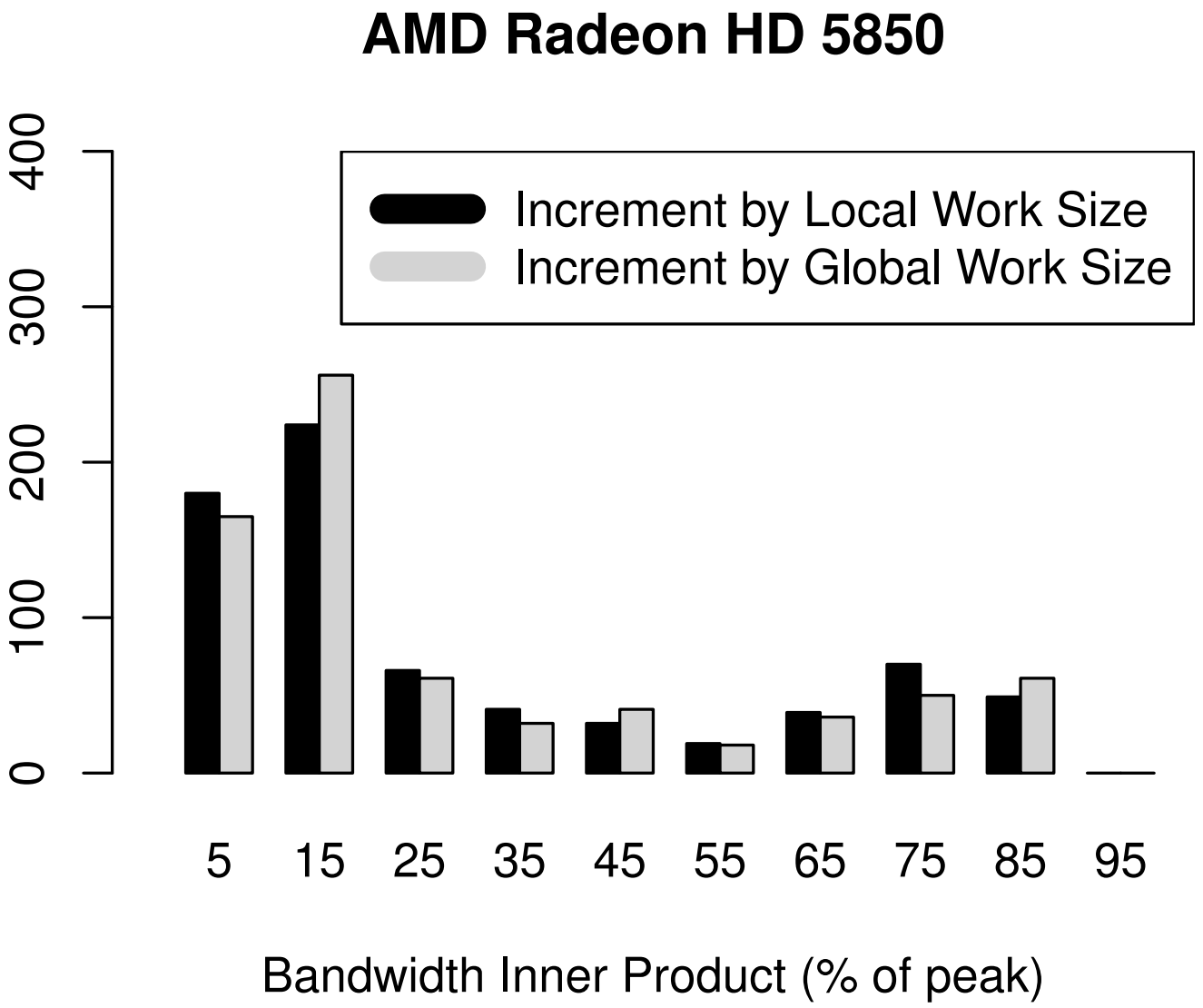
INTEL: Dual Socket Xeon E5-2670, Xeon Phi

NVIDIA: GTX 285, Tesla K20m

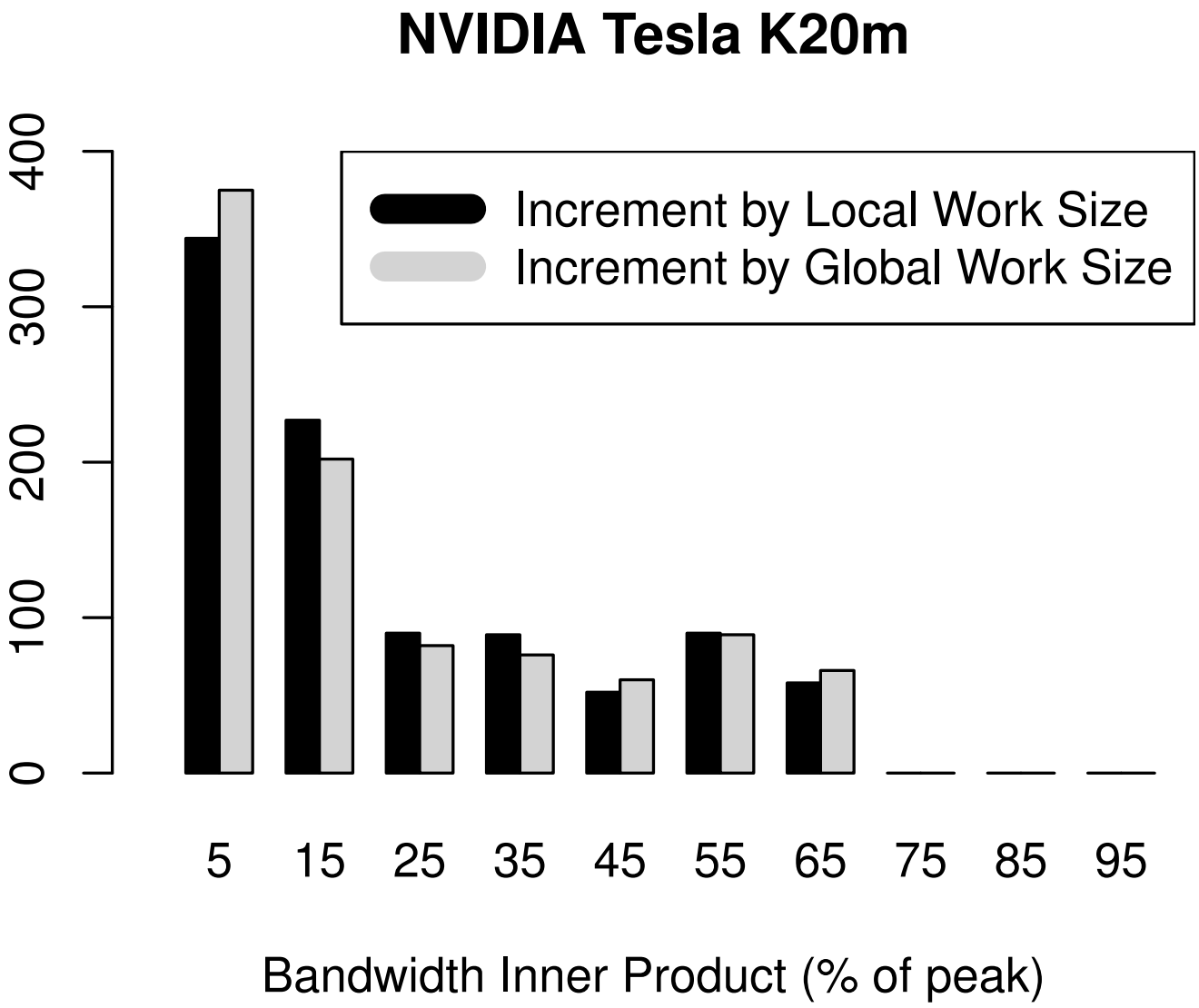
## Histograms



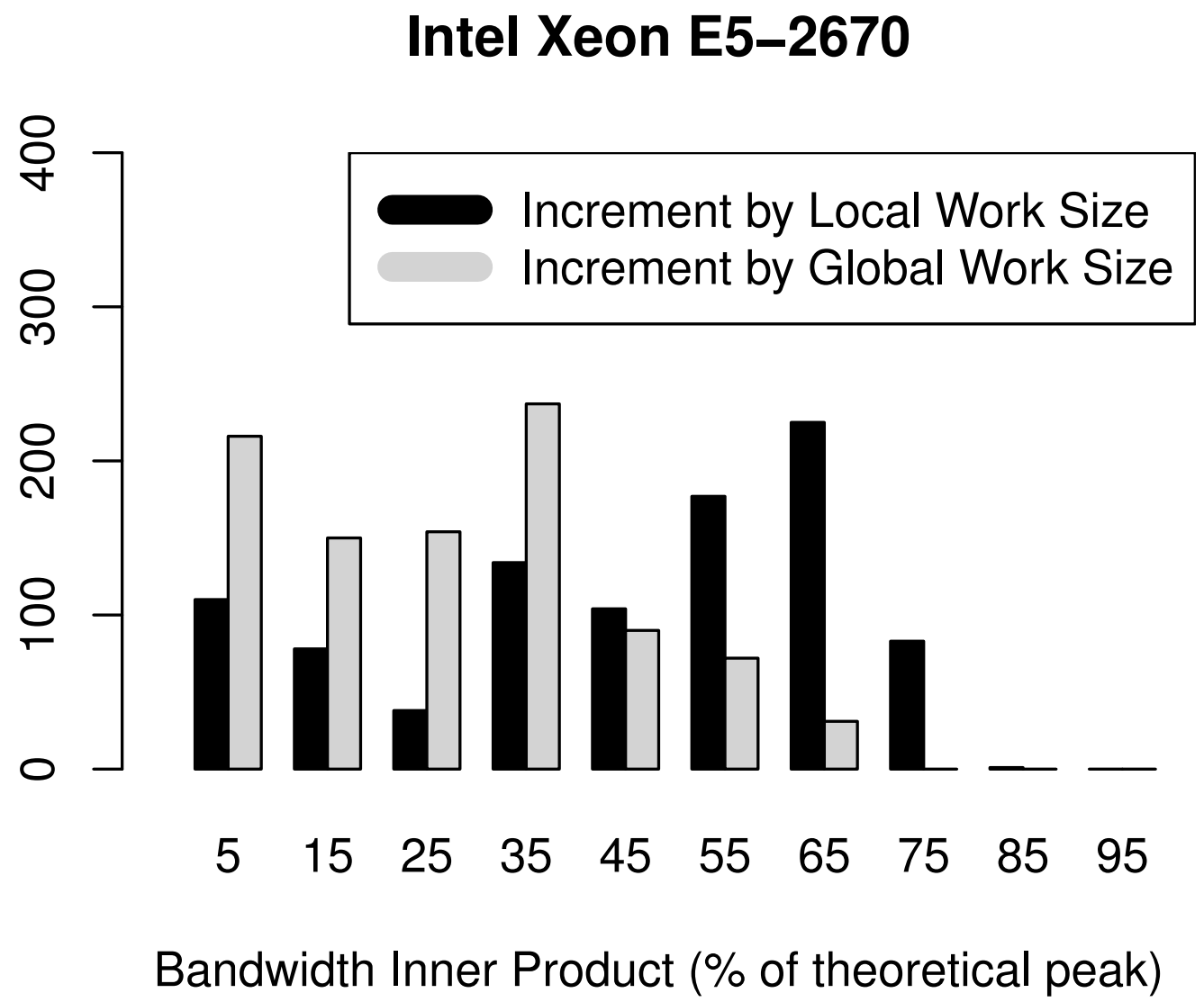
# Benchmark


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)

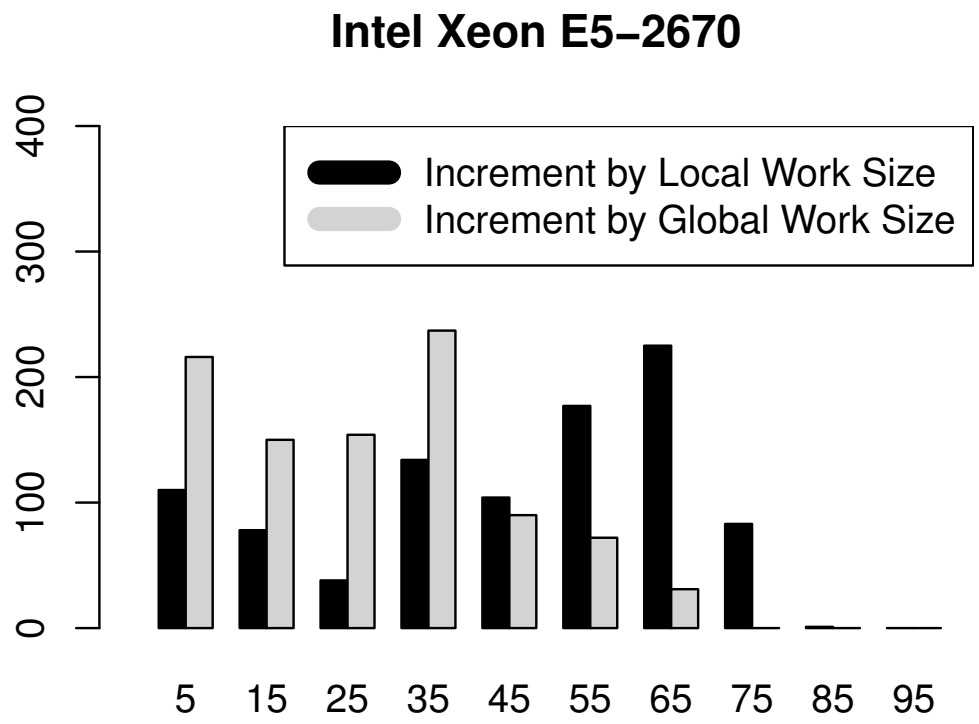
# Benchmark

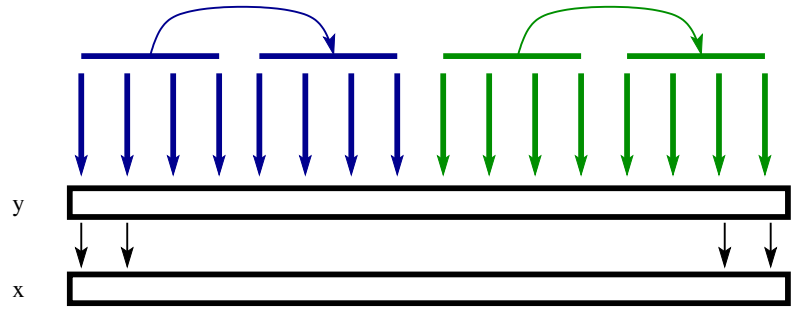

NVIDIA Tesla K20m
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of peak)

# Benchmark

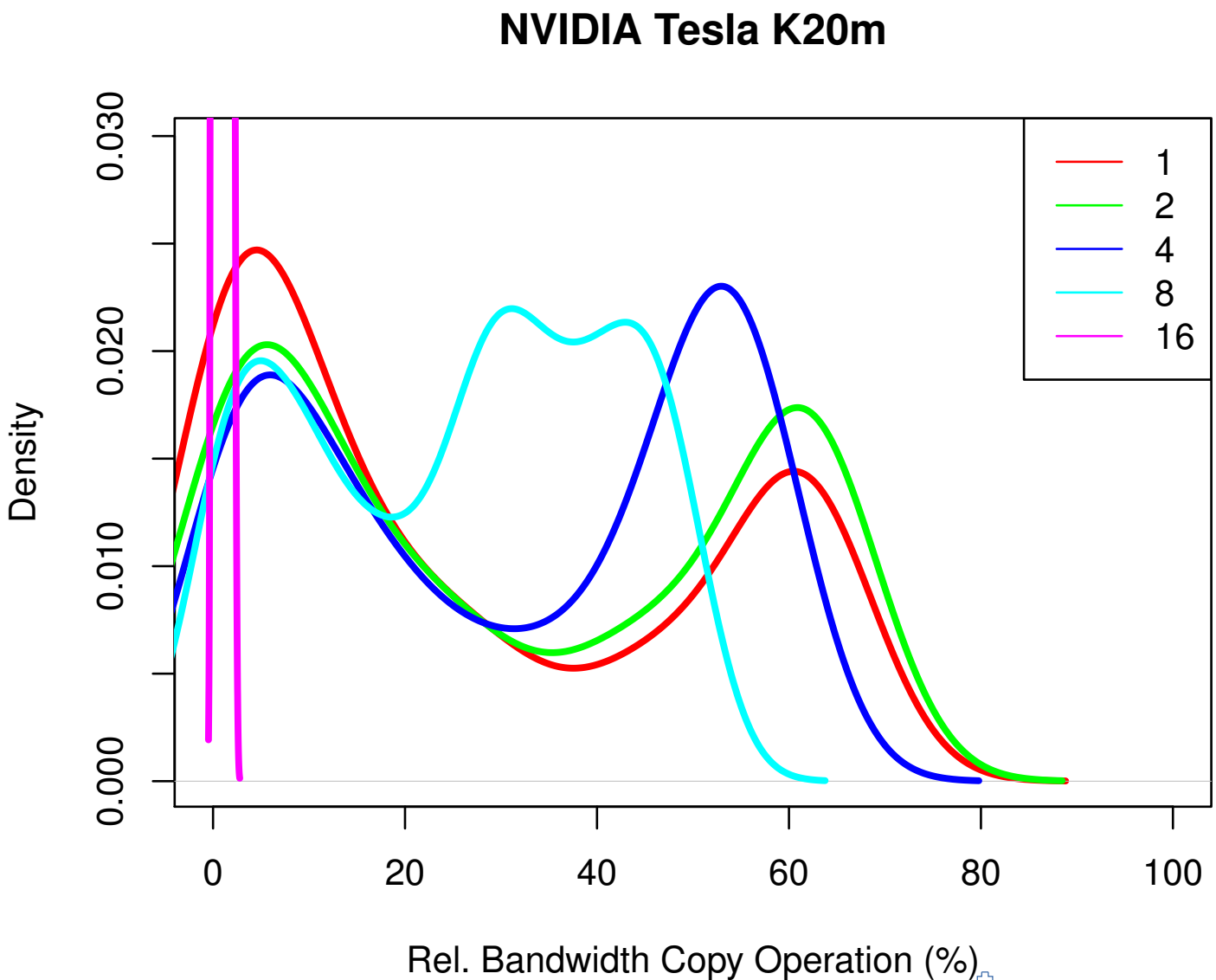

Intel Xeon E5–2670
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of theoretical peak)

# Benchmark

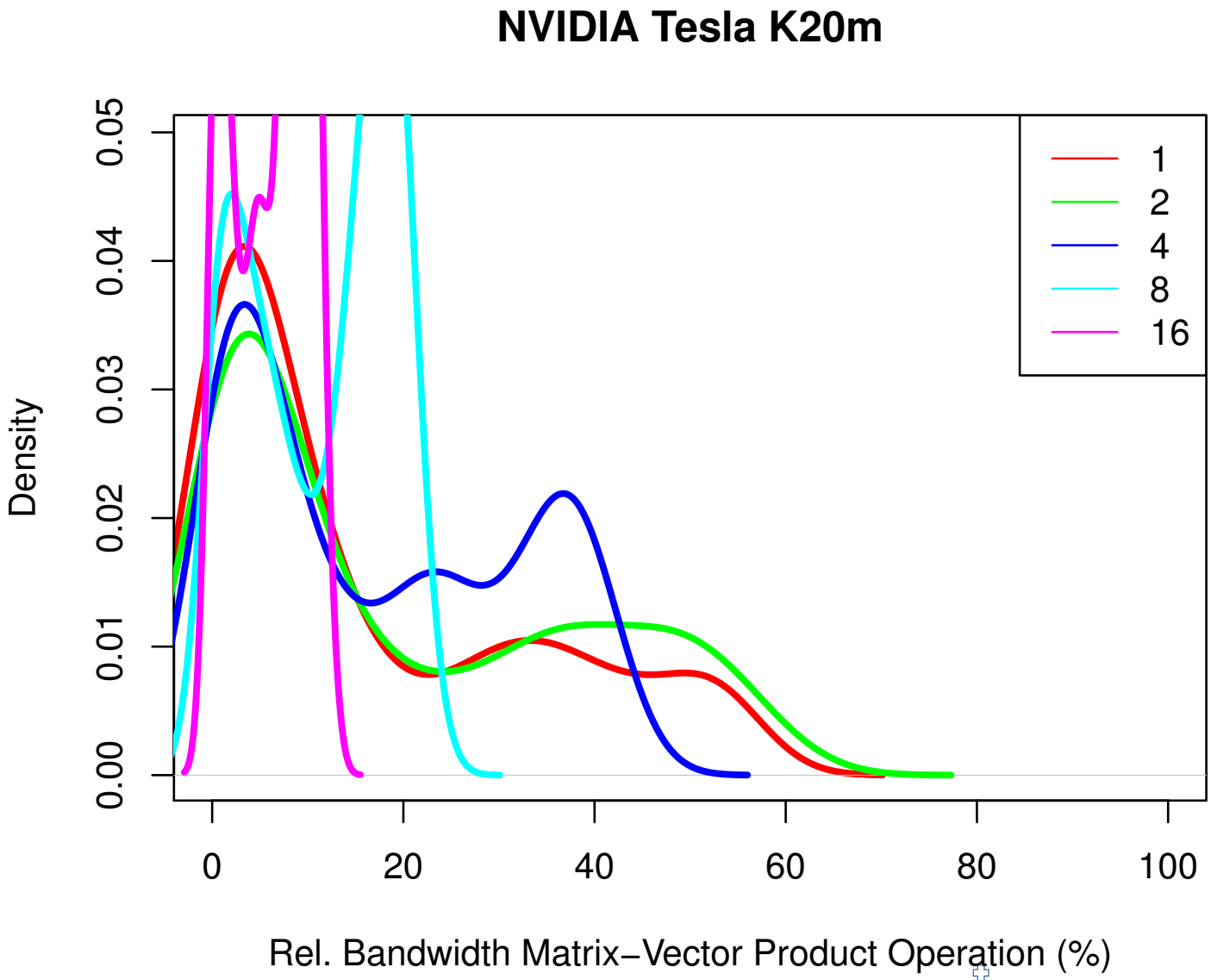

Intel Xeon E5–2670
Increment by Local Work Size
Increment by Global Work Size
0
100
200
300
400
5
15
25
35
45
55
65
75
85
95
Bandwidth Inner Product (% of theoretical peak)


y
x

# Benchmark


NVIDIA Tesla K20m
Density
0.000
0.010
0.020
0.030
0
20
40
60
80
100
Rel. Bandwidth Copy Operation (%)
1
2
4
8
16

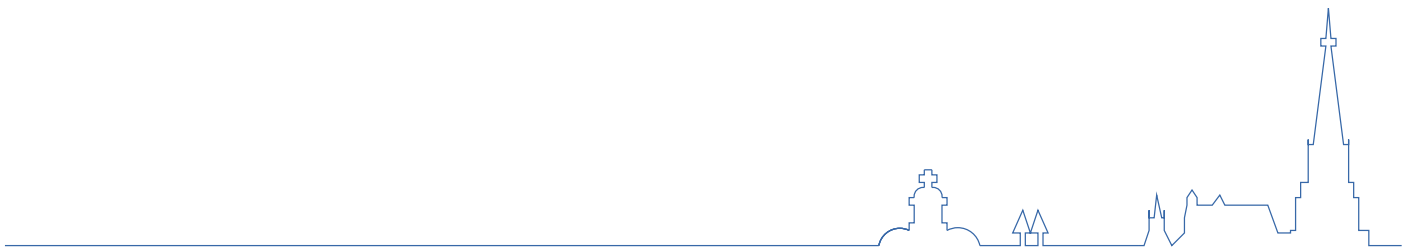
# Benchmark


NVIDIA Tesla K20m
Density
0.00
0.01
0.02
0.03
0.04
0.05
0
20
40
60
80
100
Rel. Bandwidth Matrix−Vector Product Operation (%)
1
2
4
8
16

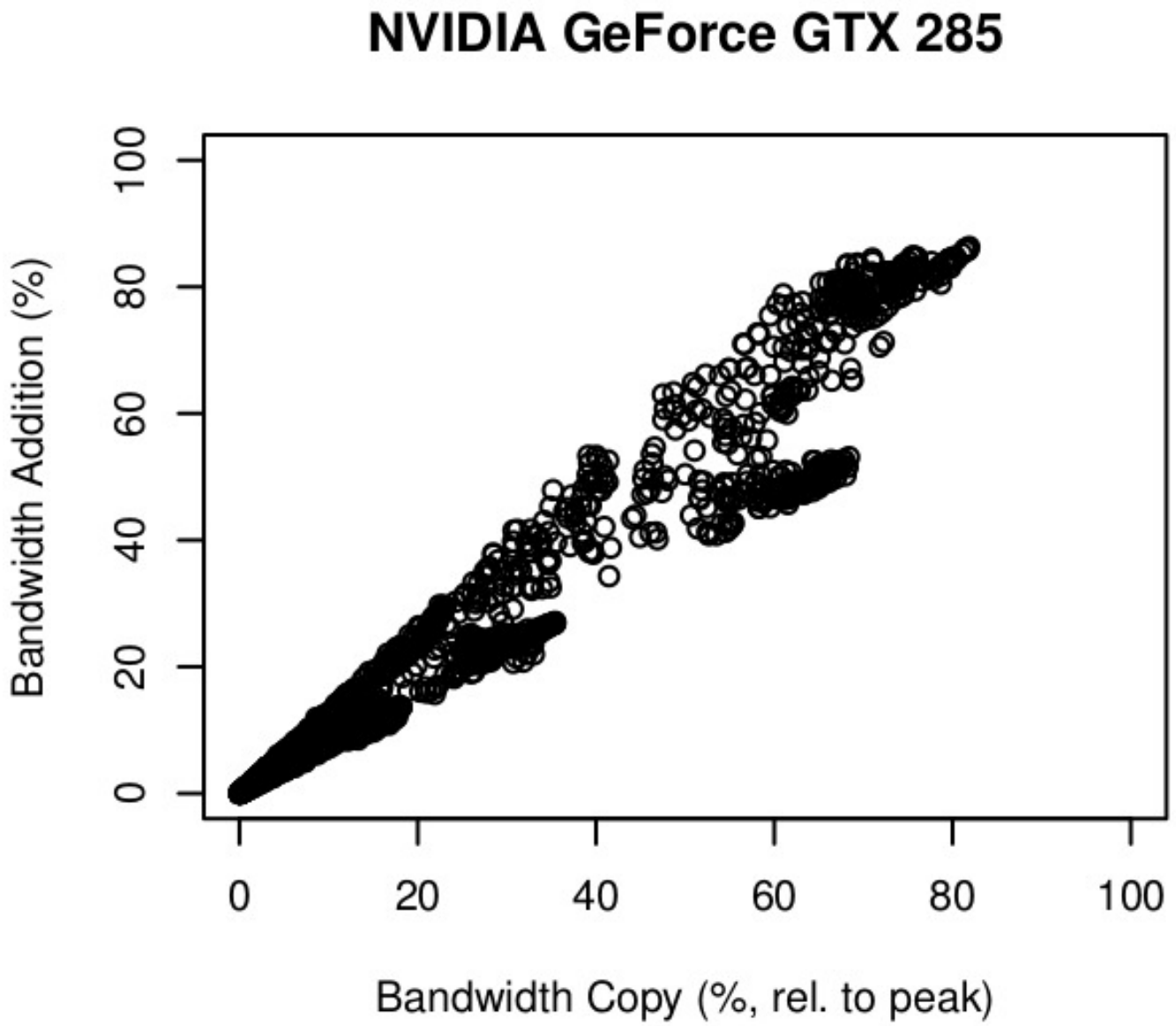
# Benchmark

**[Addition|Inner Product|Matrix-Vector] vs. Copy Kernel**

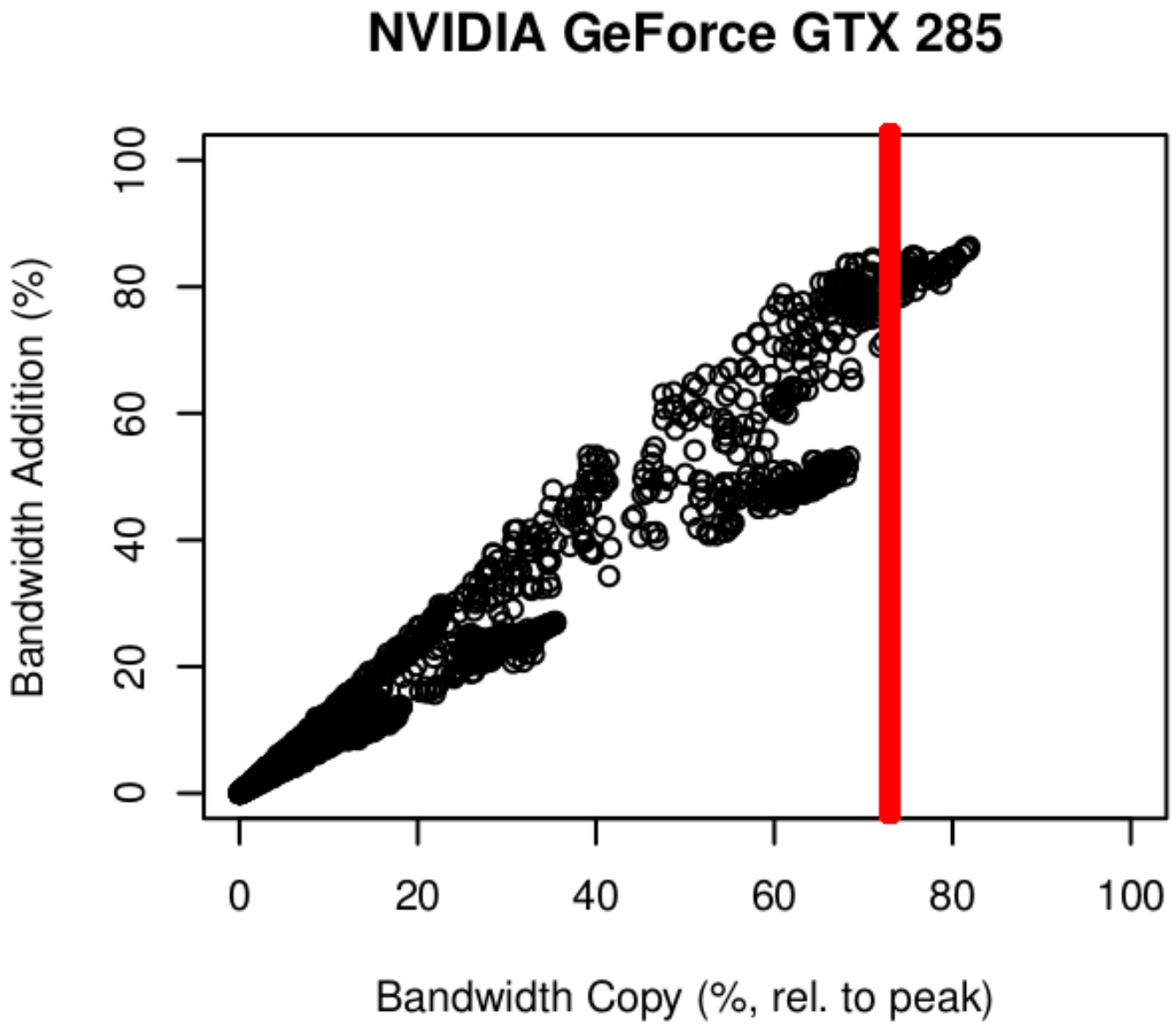
Same Device



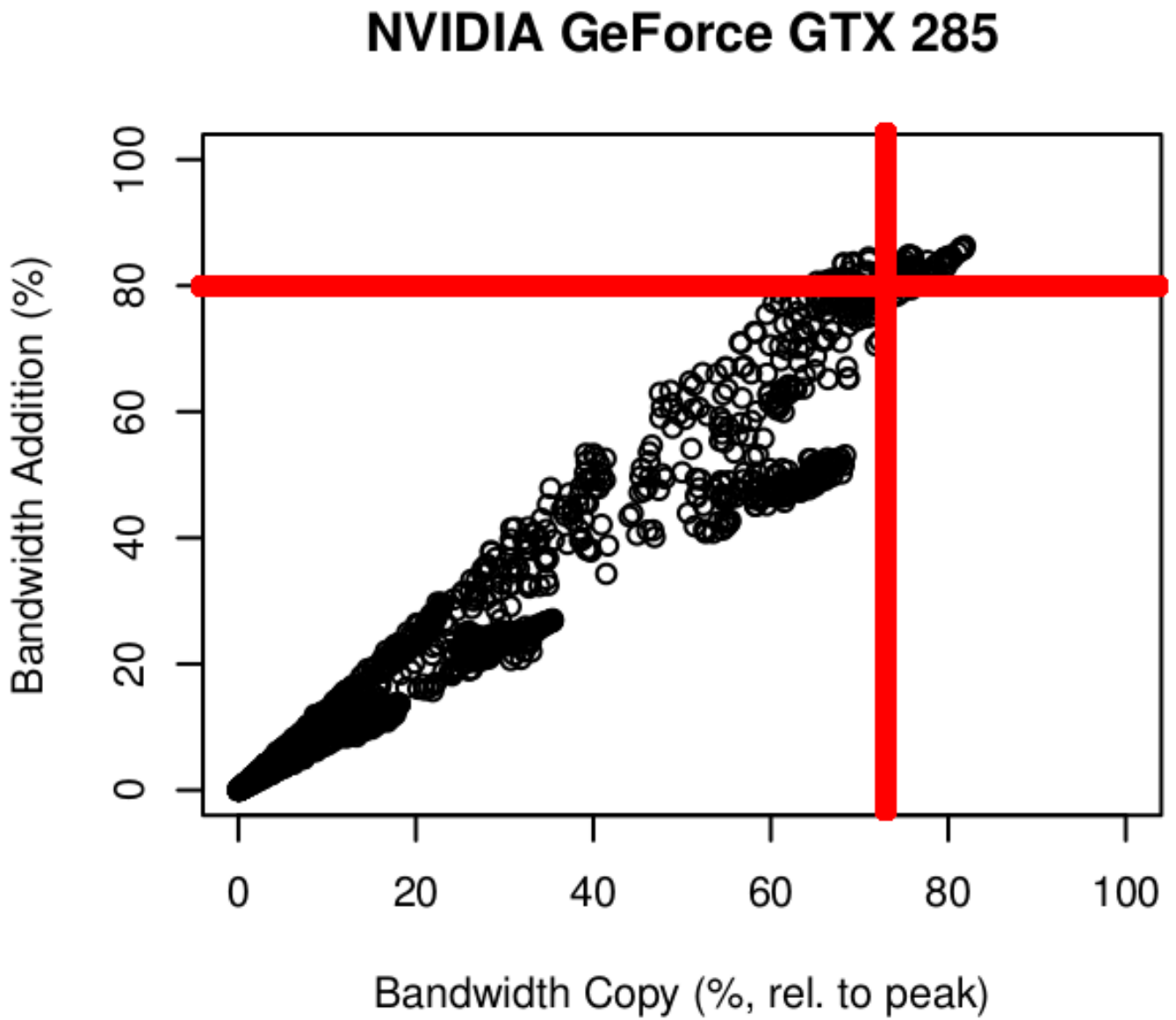
# Benchmark


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

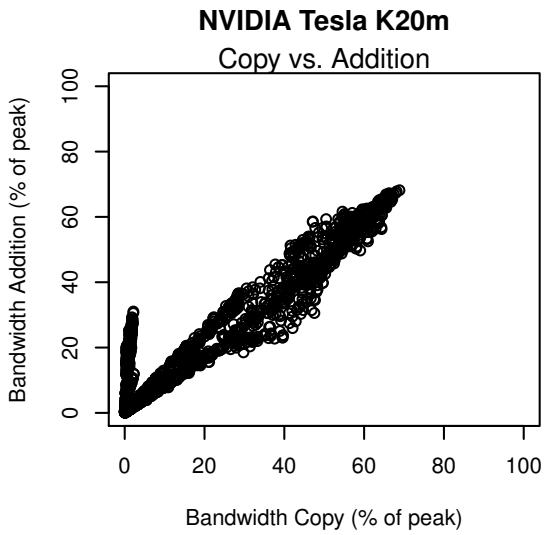
# Benchmark


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

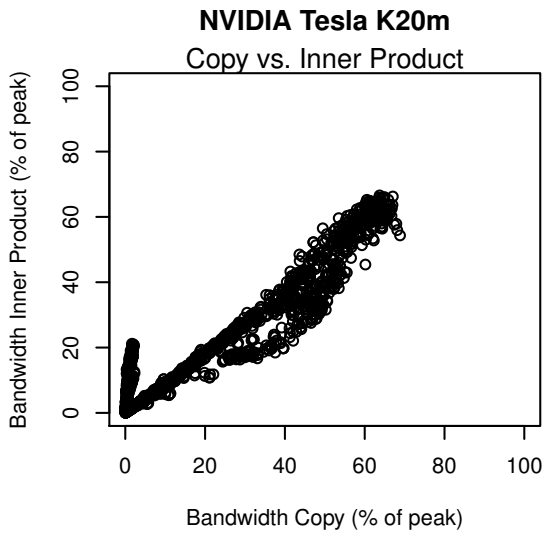
# Benchmark

## NVIDIA Tesla K20m

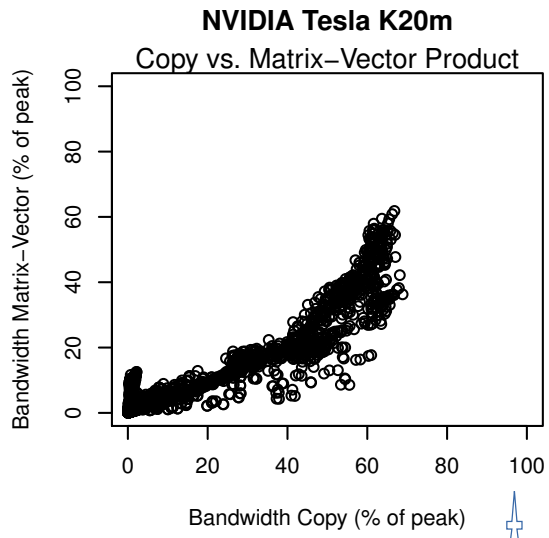
### Addition




### Inner Product



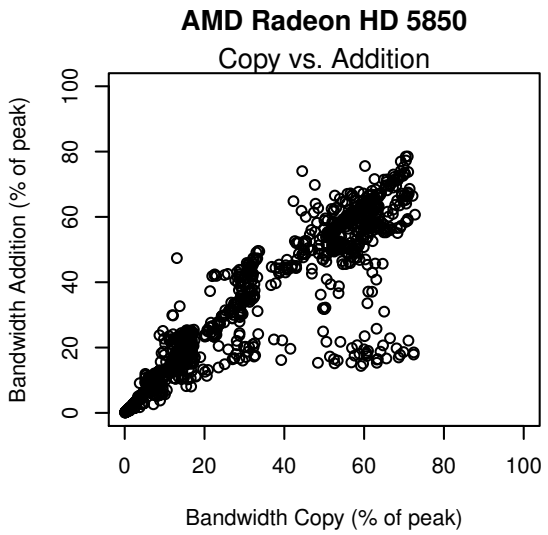

### Mat-Vec Product



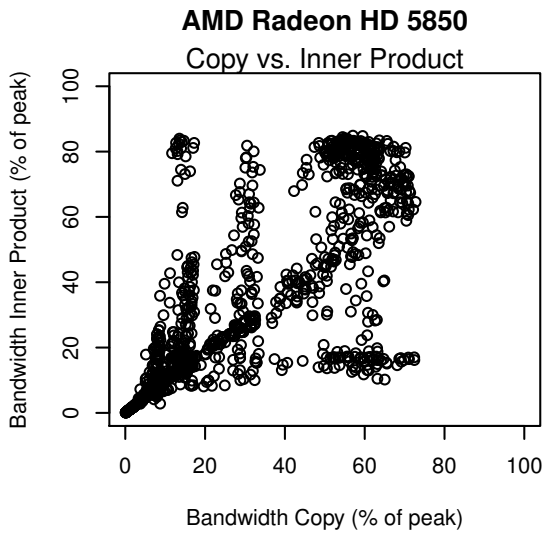
# Benchmark

## AMD Radeon HD 5850

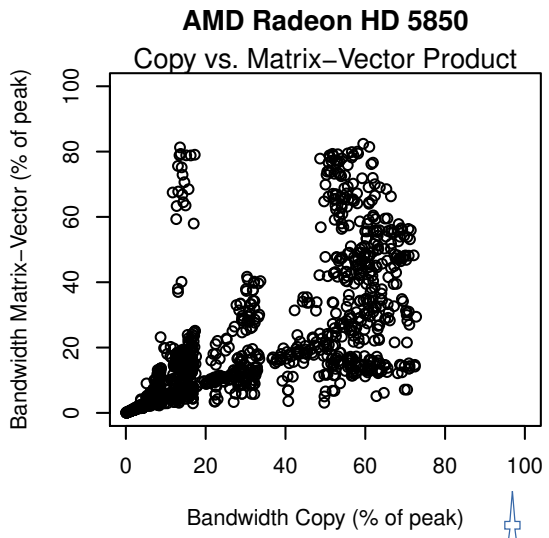
### Addition




### Inner Product



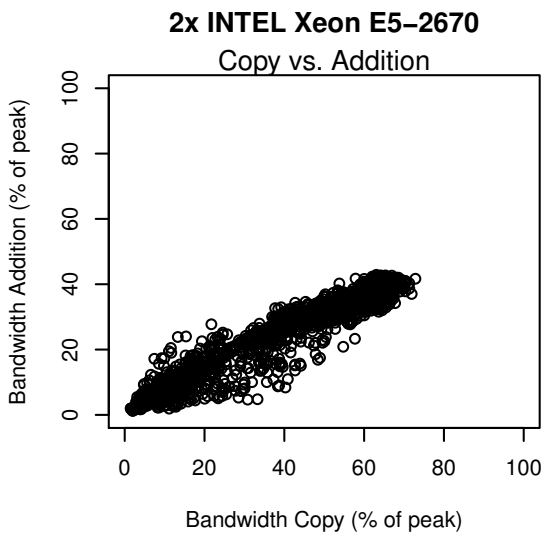

### Mat-Vec Product



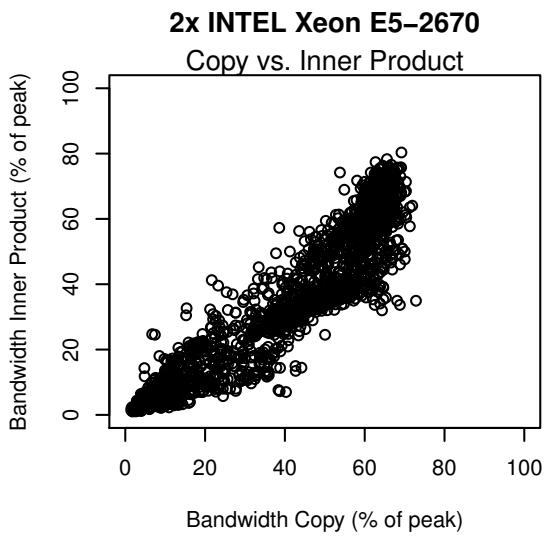
# Benchmark

## INTEL Dual Xeon E5-2670

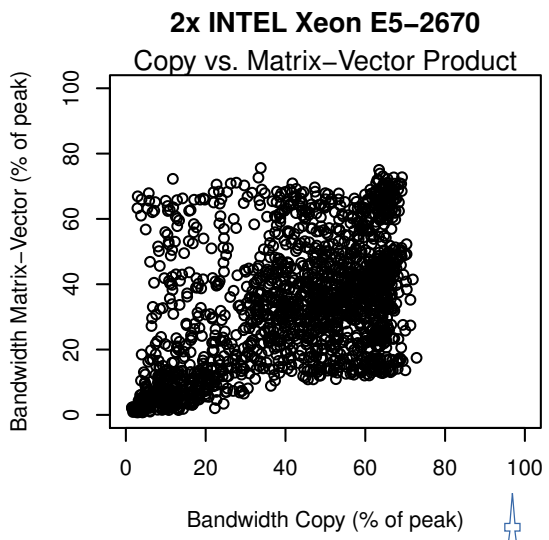
### Addition




### Inner Product



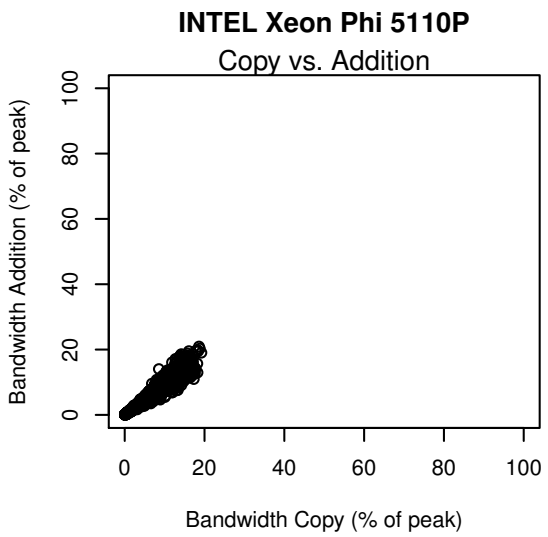

### Mat-Vec Product



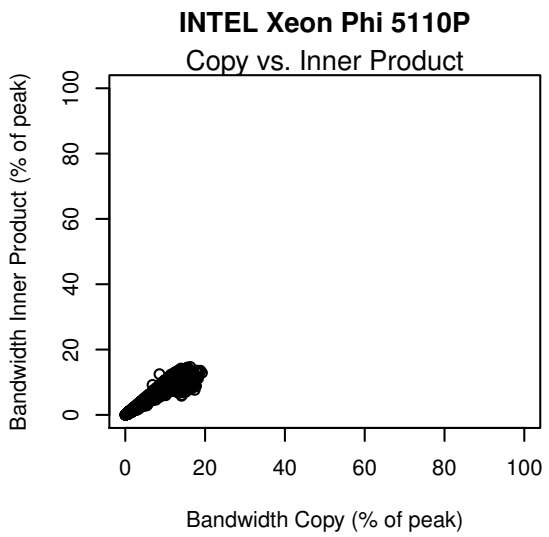
# Benchmark

## INTEL Xeon Phi

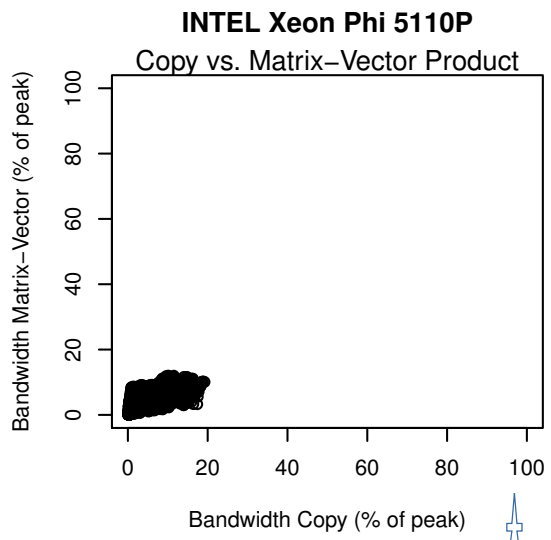
### Addition




### Inner Product



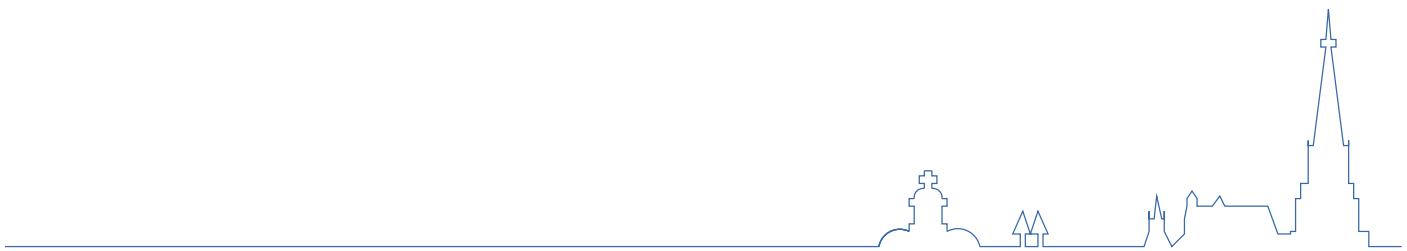

### Mat-Vec Product



# Benchmark

Conclusio:

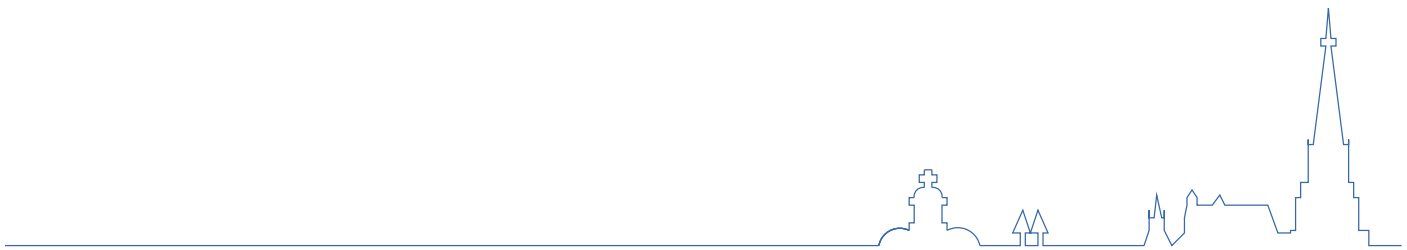
Focus on fastest configurations for copy-kernel sufficient



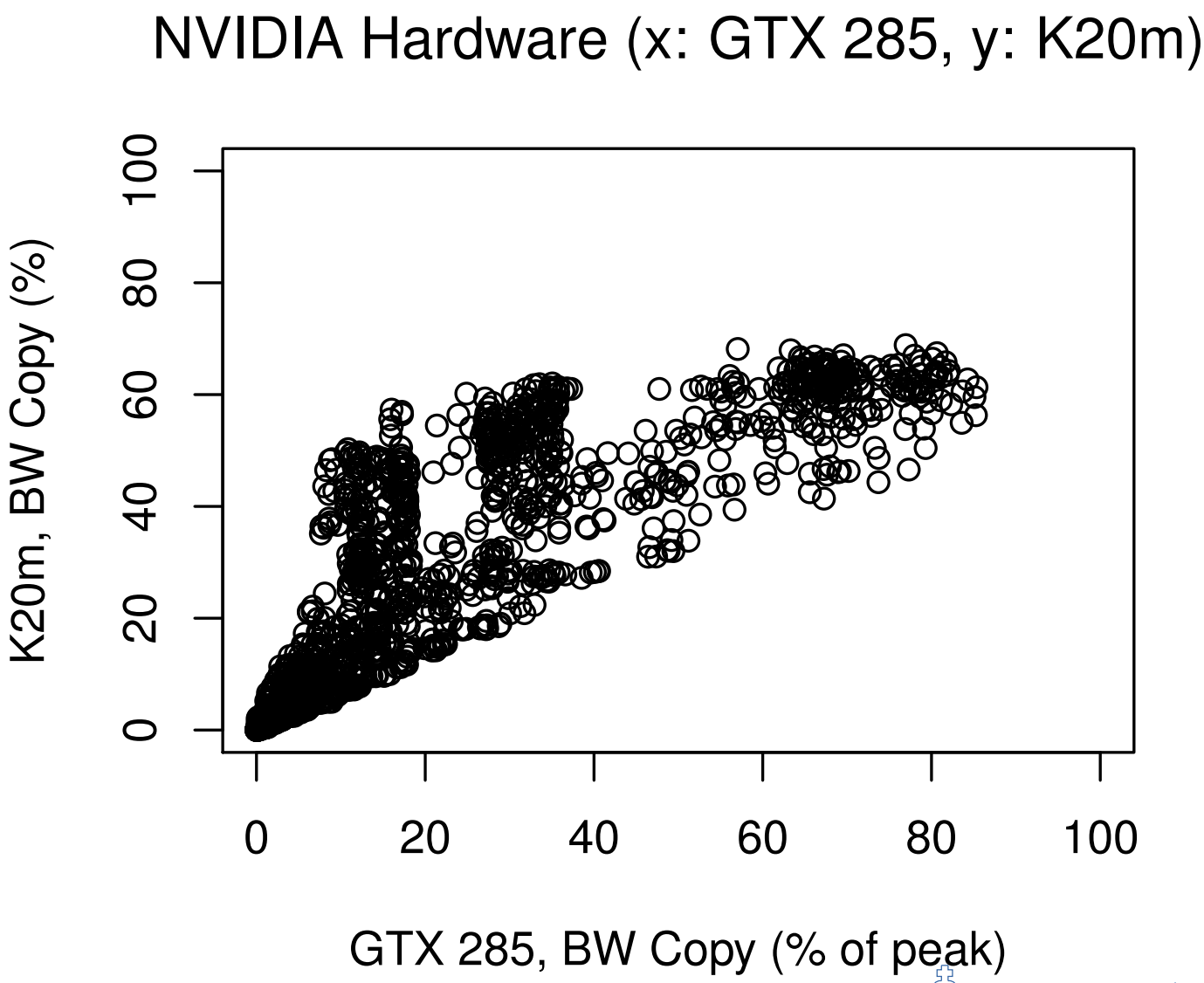
# Benchmark

**[Copy|Addition|Inner Product|Matrix-Vector] vs. Copy Kernel**

Different Device, Same Vendor

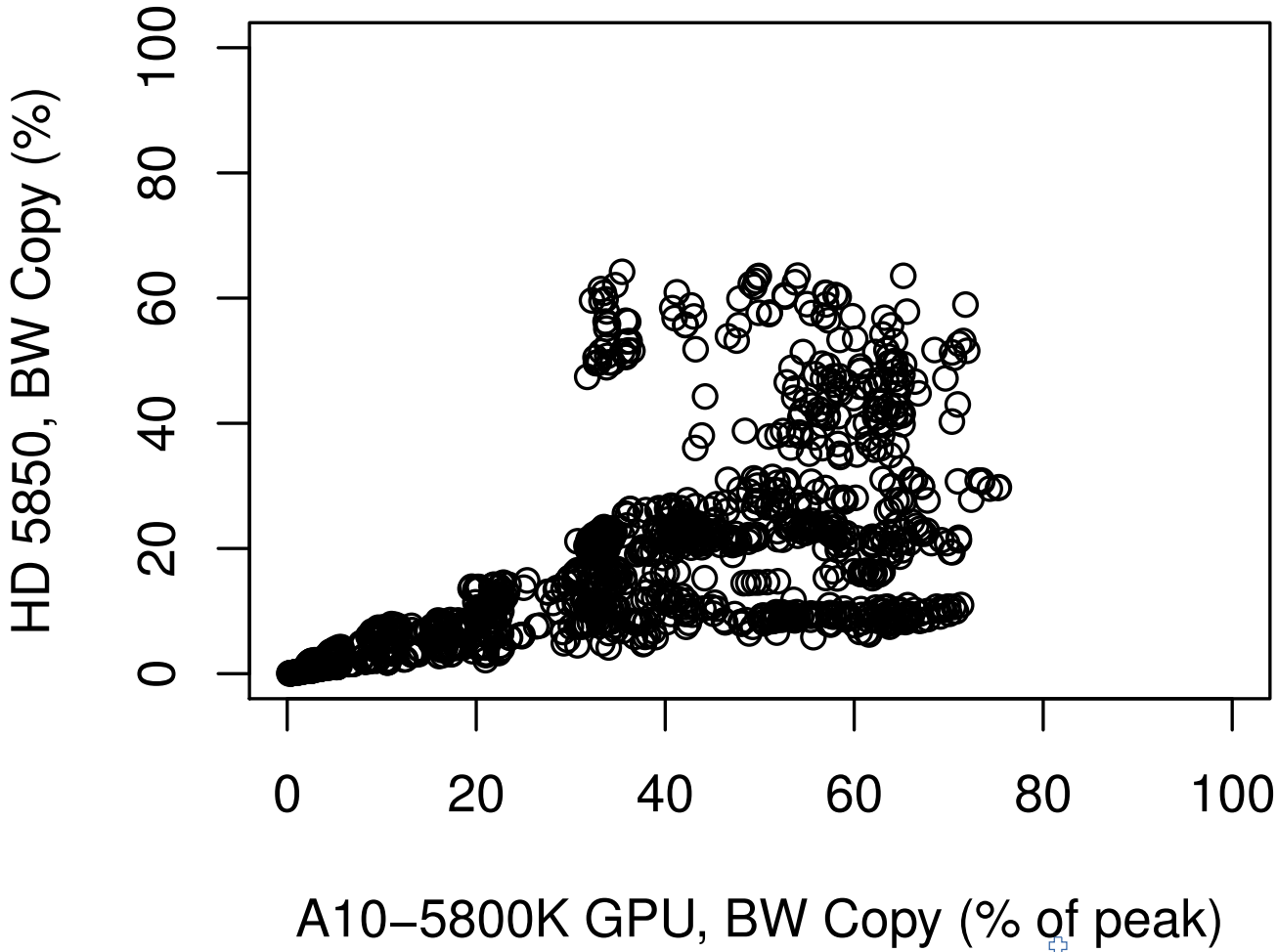


# Benchmark

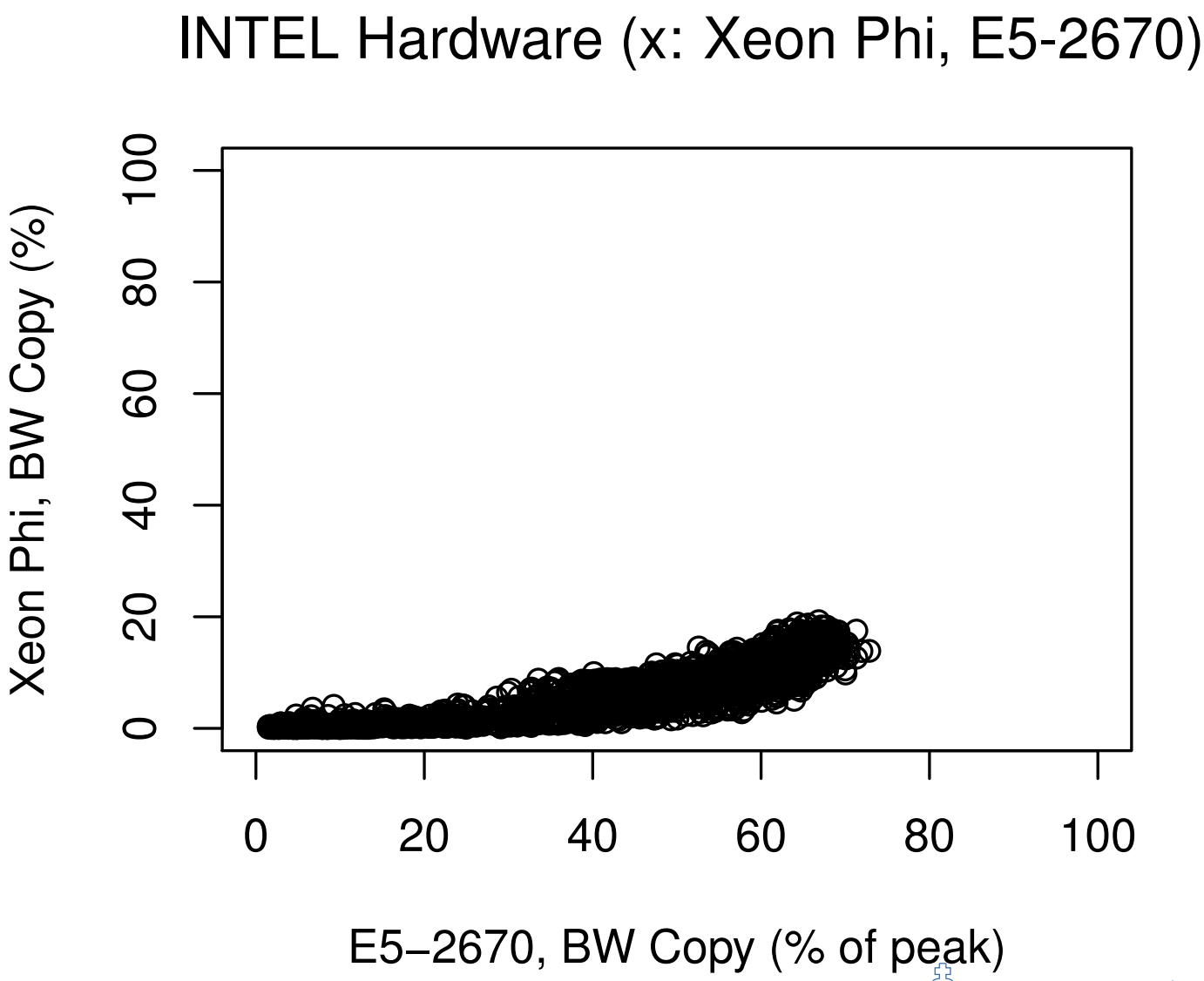

NVIDIA Hardware (x: GTX 285, y: K20m)
K20m, BW Copy (%)
0
20
40
60
80
100
0
20
40
60
80
100
GTX 285, BW Copy (% of peak)

# Benchmark

AMD Hardware (x: A10-5800K GPU, y: HD 5850)



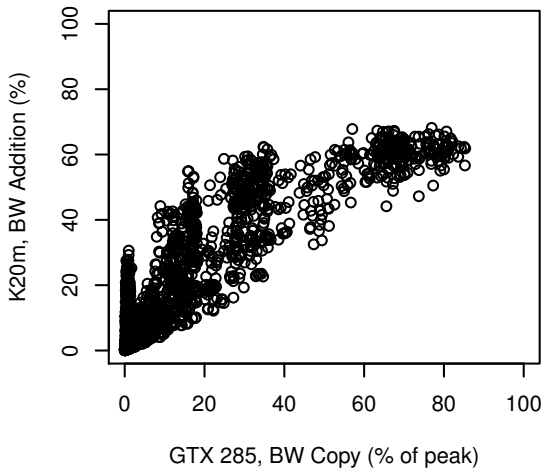
# Benchmark


INTEL Hardware (x: Xeon Phi, E5-2670)
Xeon Phi, BW Copy (%)
0
20
40
60
80
100
0
20
40
60
80
100
E5–2670, BW Copy (% of peak)

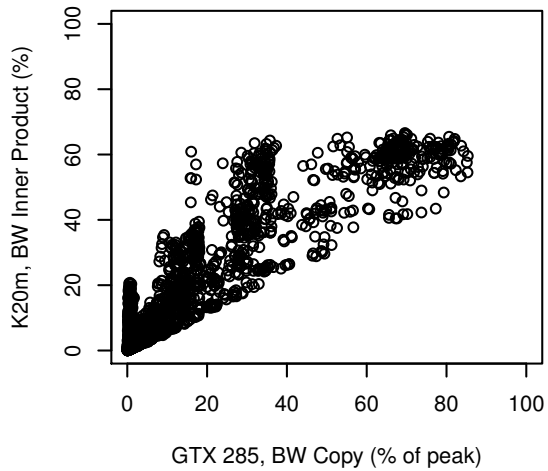
# Benchmark

## NVIDIA Hardware (x: GTX 285, y: K20m)

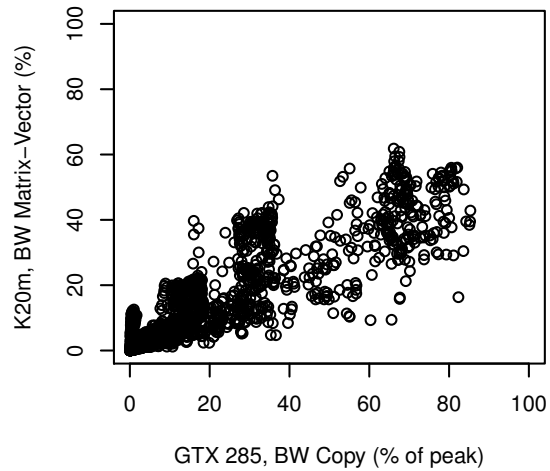
### Addition




### Inner Product



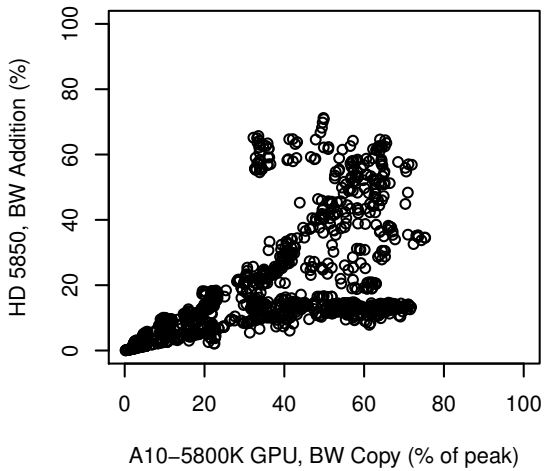

### Mat-Vec Product



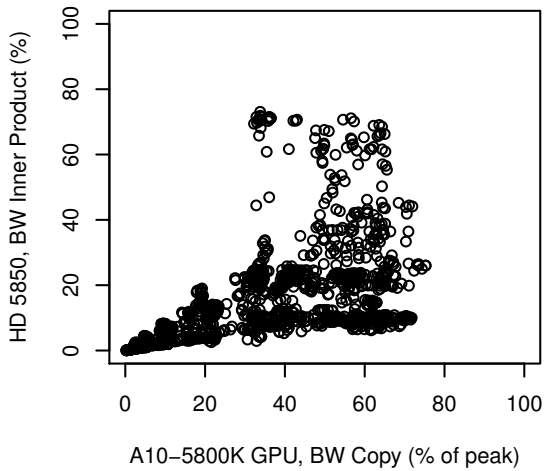
# Benchmark

## AMD Hardware (x: A10-5800K GPU, y: HD 5850)

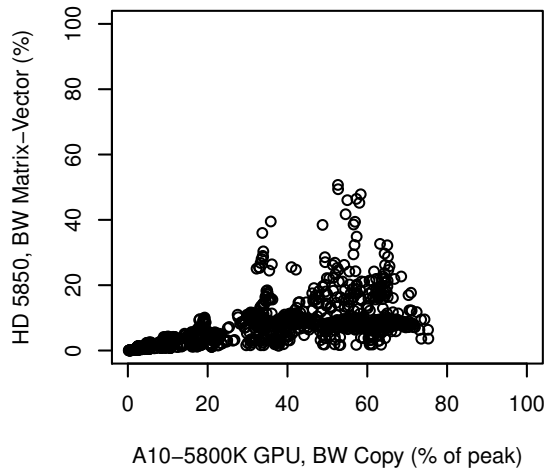
### Addition




### Inner Product



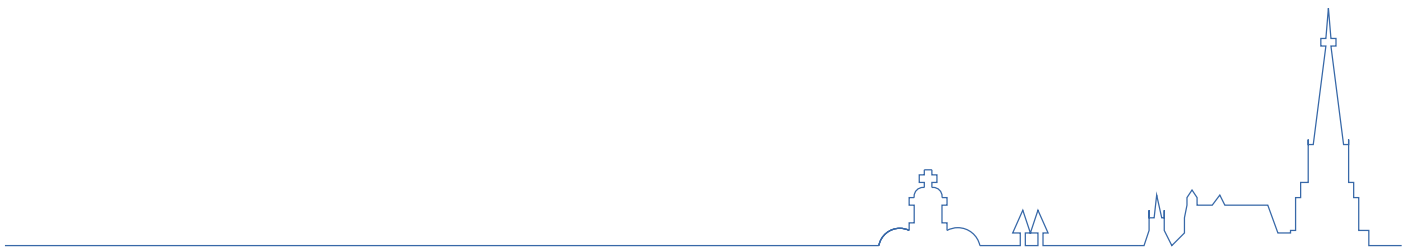

### Mat-Vec Product



# Benchmark

Conclusio:

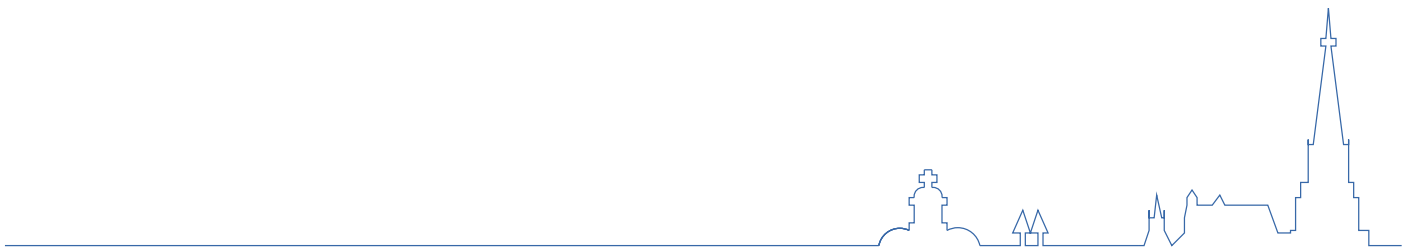
Certain Performance Portability per Vendor



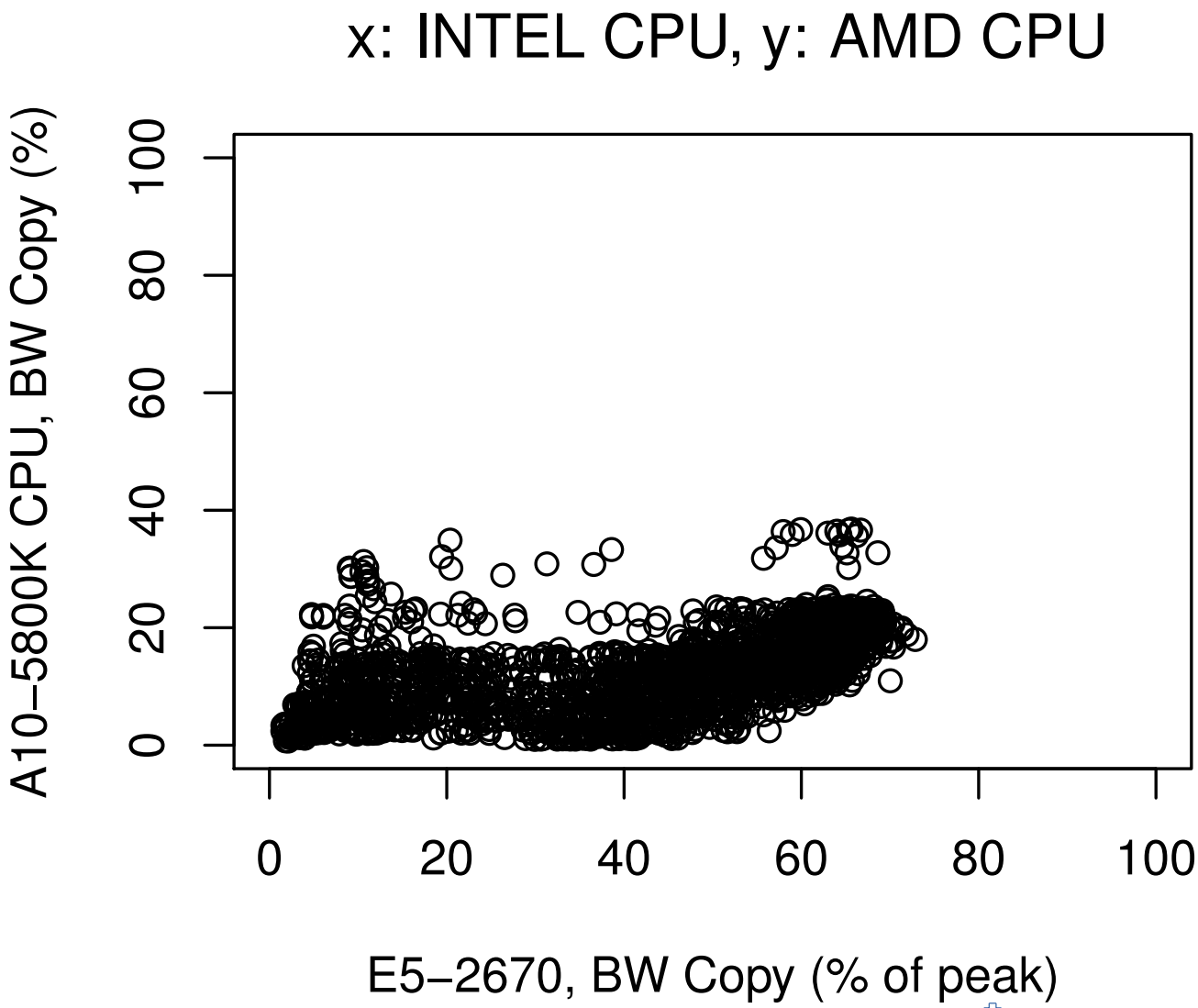
# Benchmark

**[Copy|Addition|Inner Product|Matrix-Vector] vs. Copy Kernel**

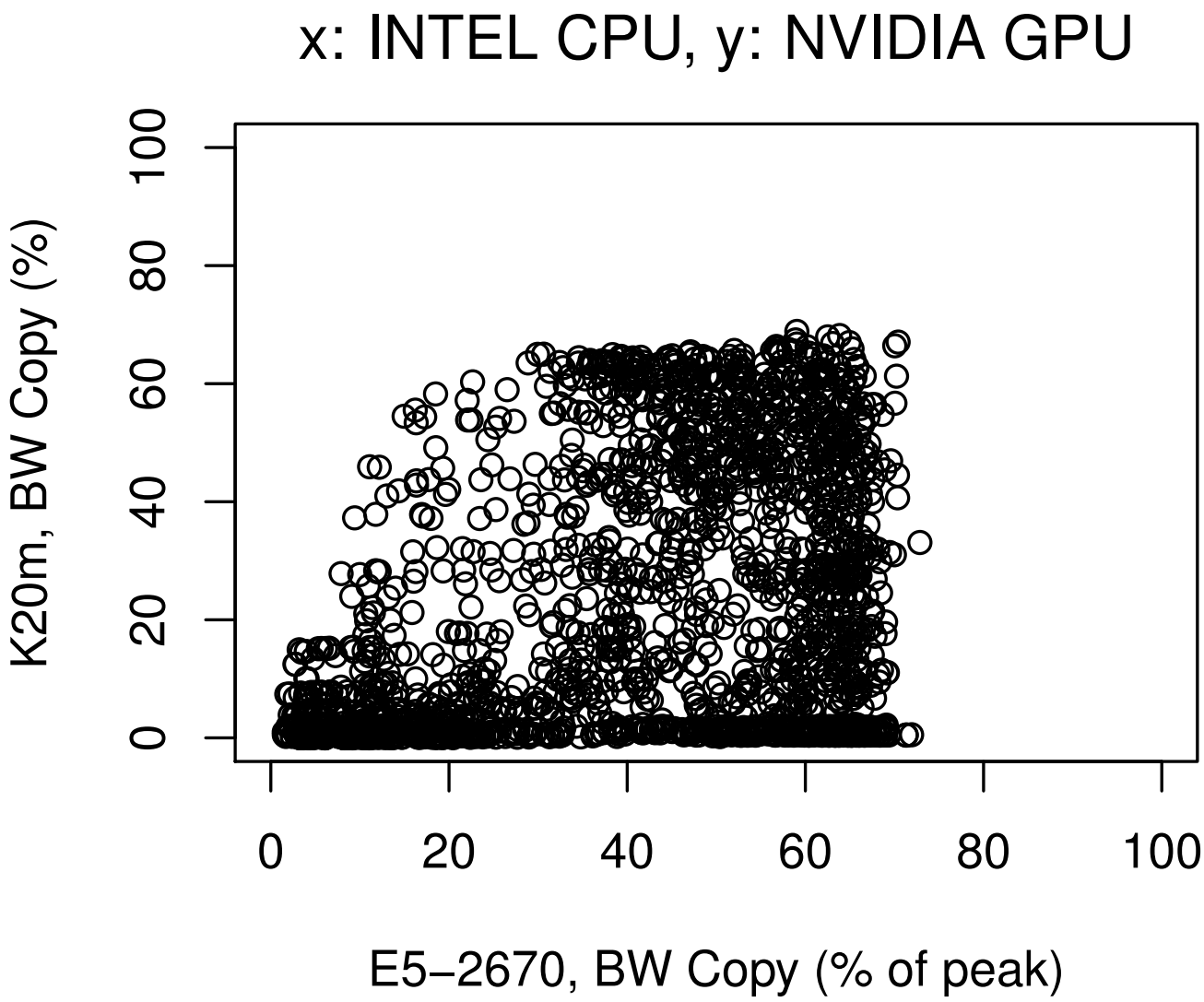
Different Device, Different Vendor



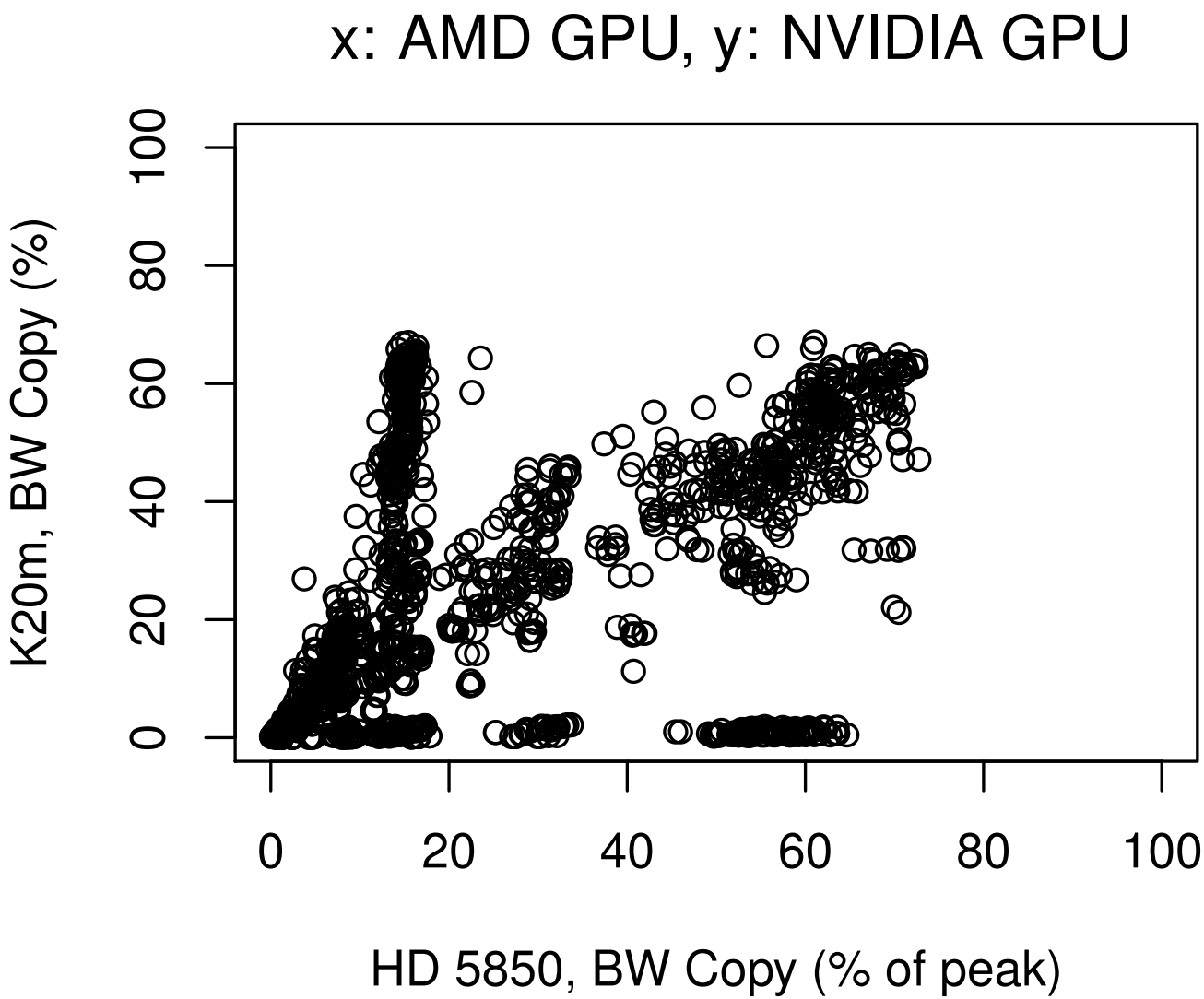
# Benchmark


x: INTEL CPU, y: AMD CPU
A10–5800K CPU, BW Copy (%)
0
20
40
60
80
100
E5–2670, BW Copy (% of peak)

# Benchmark


x: INTEL CPU, y: NVIDIA GPU
K20m, BW Copy (%)
0
20
40
60
80
100
0
20
40
60
80
100
E5−2670, BW Copy (% of peak)

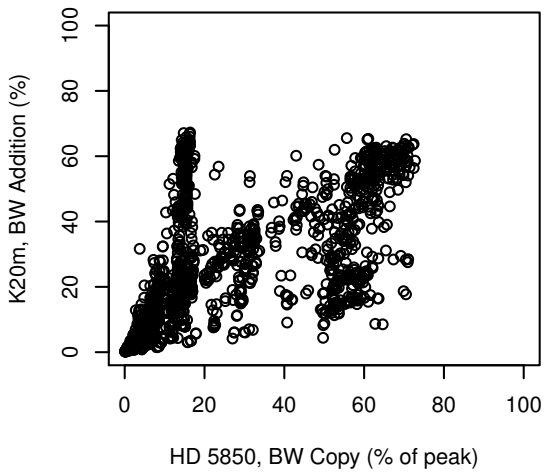
# Benchmark


x: AMD GPU, y: NVIDIA GPU
K20m, BW Copy (%)
0
20
40
60
80
100
0
20
40
60
80
100
HD 5850, BW Copy (% of peak)

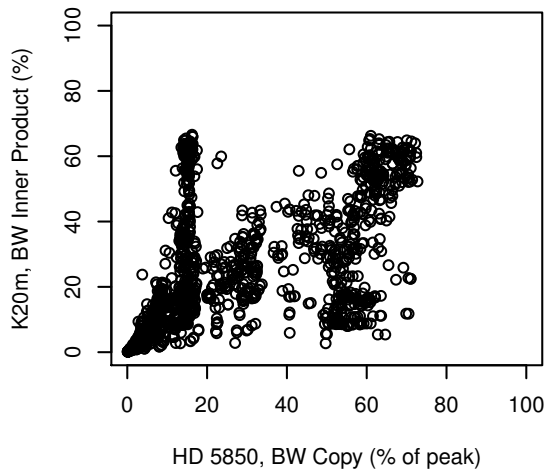
# Benchmark

x: AMD HD 5850, y: NVIDIA K20m

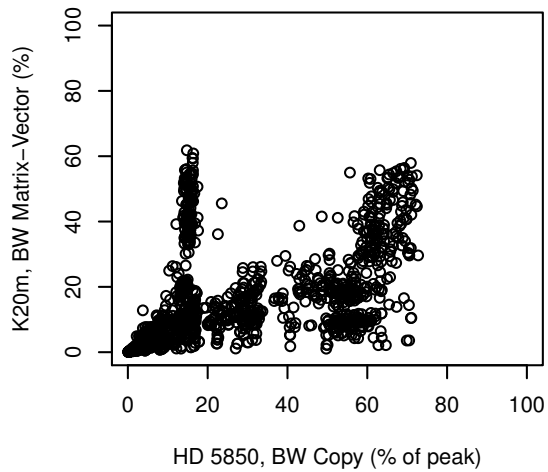
Addition




Inner Product



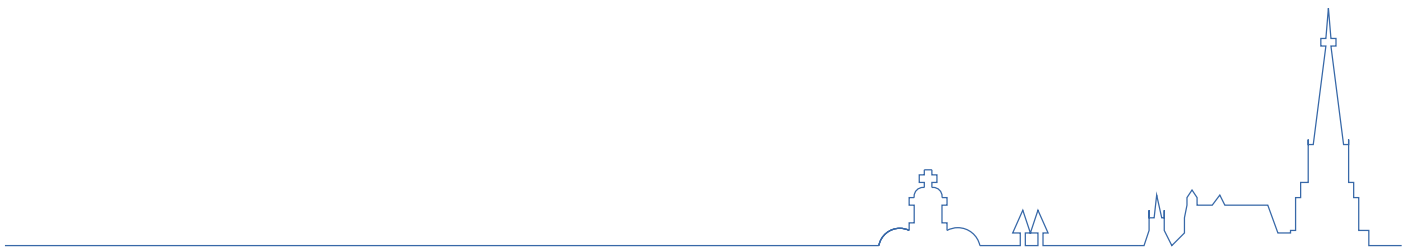

Mat-Vec Product



# Benchmark

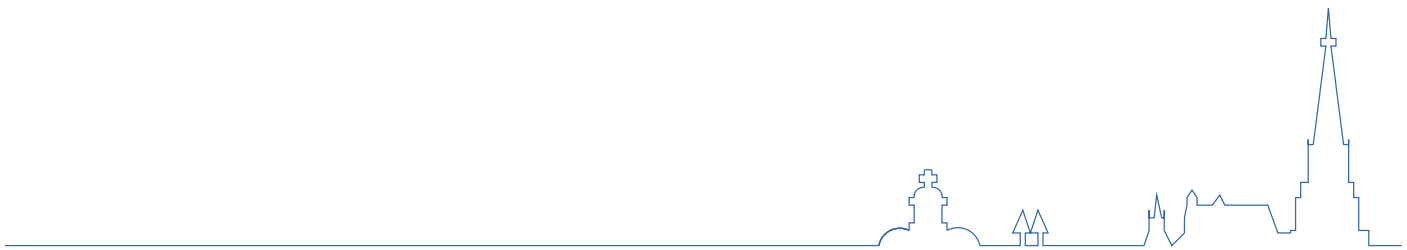
Conclusio:

Fast Configurations Across Vendors Exist

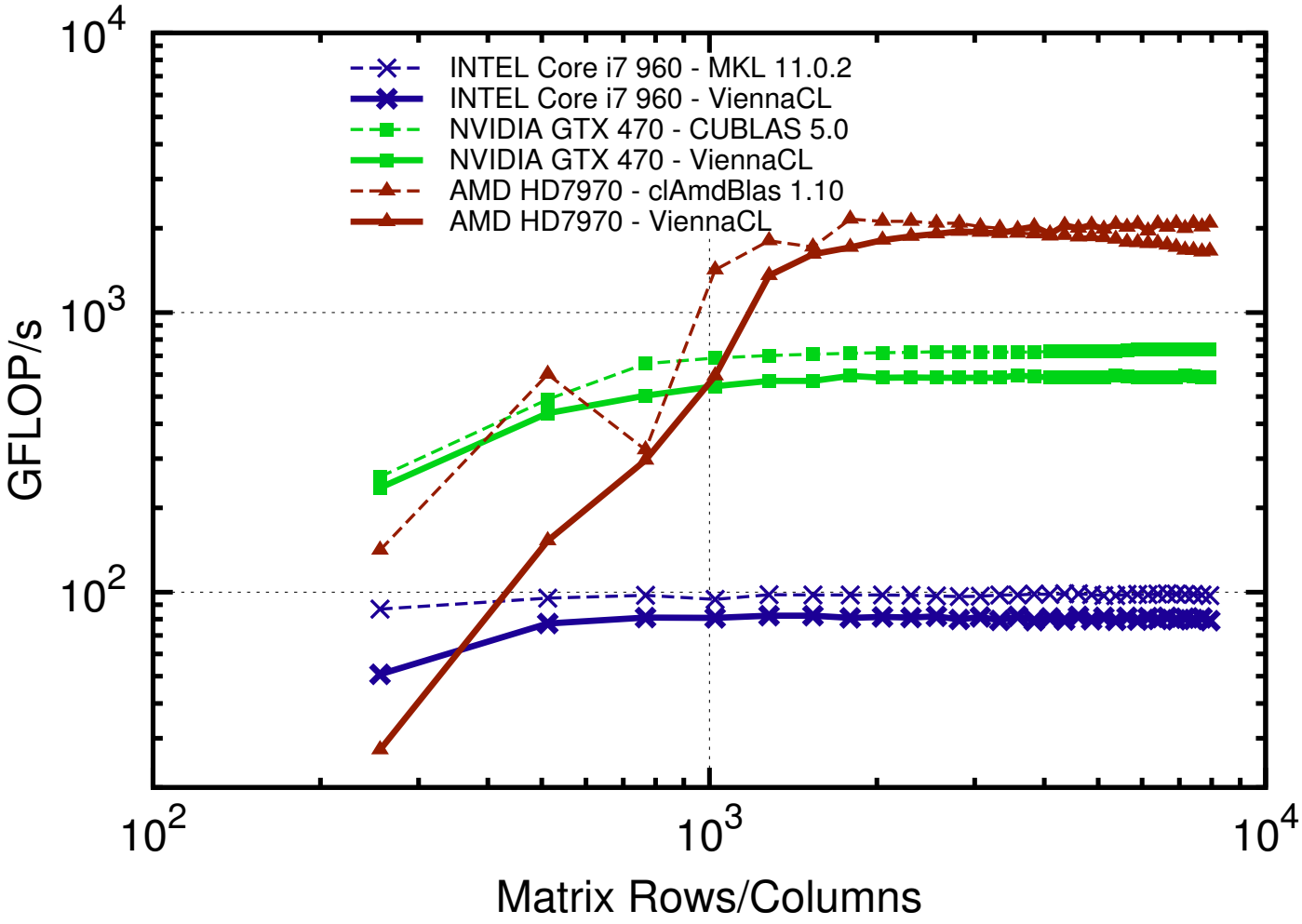


# Matrix-Matrix Multiplication

Single Precision

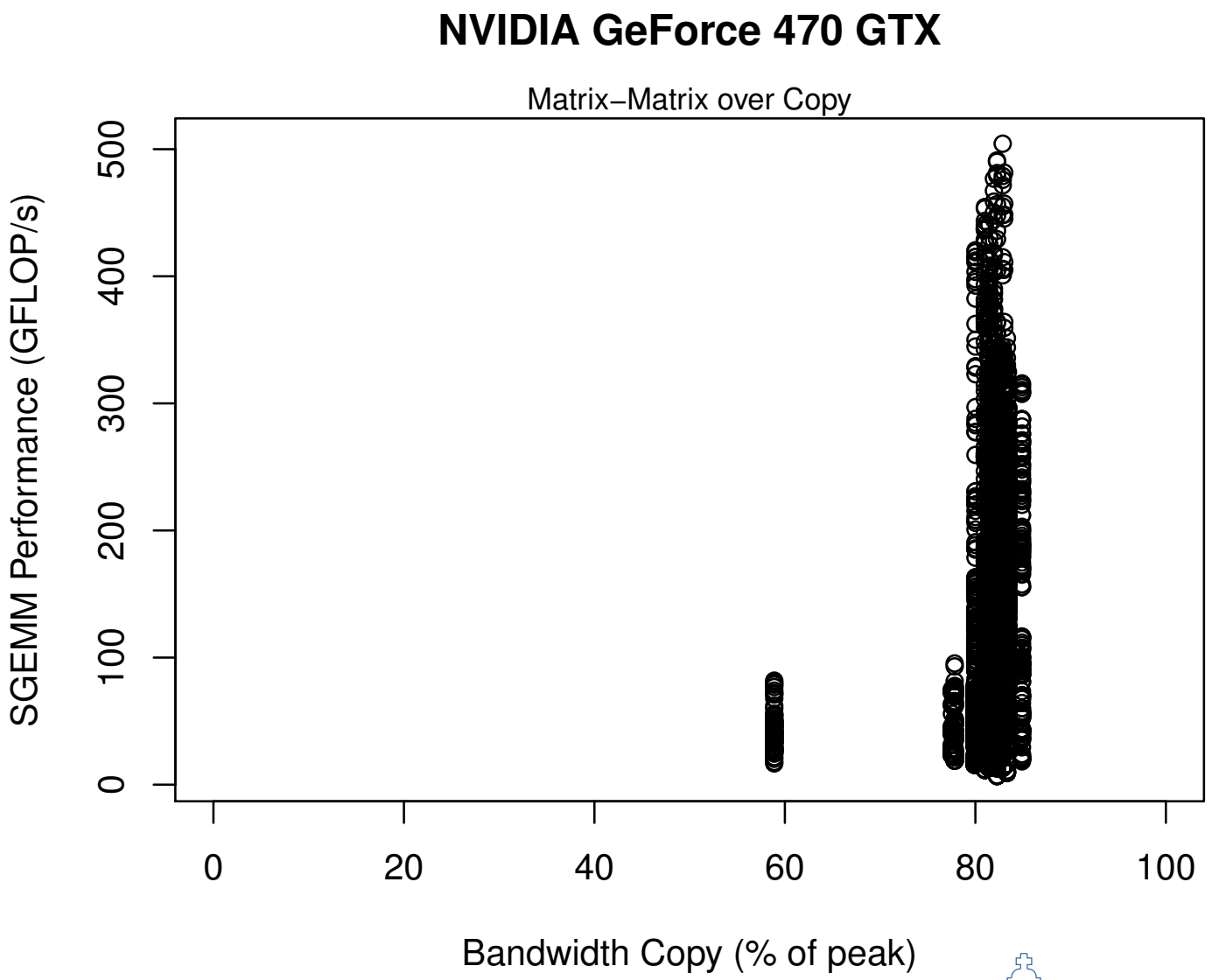


# Benchmark




Ph. Tillet *et al.*: HotPar'13

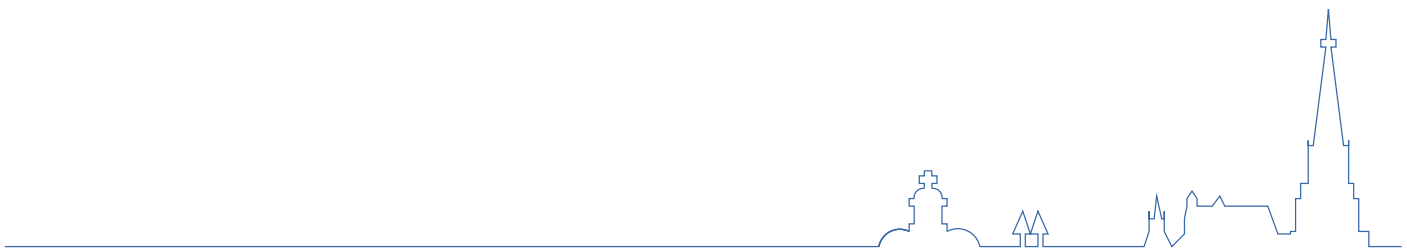
# Benchmark


NVIDIA GeForce 470 GTX
Matrix−Matrix over Copy
SGEMM Performance (GFLOP/s)
0
100
200
300
400
500
0
20
40
60
80
100
Bandwidth Copy (% of peak)

# Conclusion

## Copy Kernel

- Prototype for complicated memory-bandwidth-limited kernels
- Strong performance correlation
- Reduces search space for auto-tuning



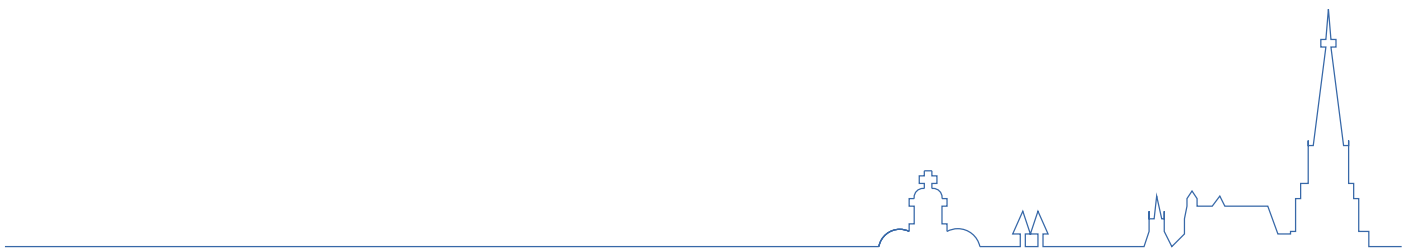
# Conclusion

## Copy Kernel

- Prototype for complicated memory-bandwidth-limited kernels
- Strong performance correlation
- Reduces search space for auto-tuning

## Implications

- Tune primarily with memory transfers in mind
- Prefer regular memory access patterns
- Use *appropriate* vector data types



# Conclusion

## Copy Kernel

- Prototype for complicated memory-bandwidth-limited kernels
- Strong performance correlation
- Reduces search space for auto-tuning

## Implications

- Tune primarily with memory transfers in mind
- Prefer regular memory access patterns
- Use *appropriate* vector data types

## Next Steps

- Build a device parameter database
- Provide tools for crowd-tuning