



Intro to High-Performance Computing with GPUs

Openlab Summer Student Programme 2019

Ahmad Hesam

19/07/2019

About Me



- Bachelor's Applied Physics
- Master's Computer Engineering

Joined CERN as Openlab Summer Student

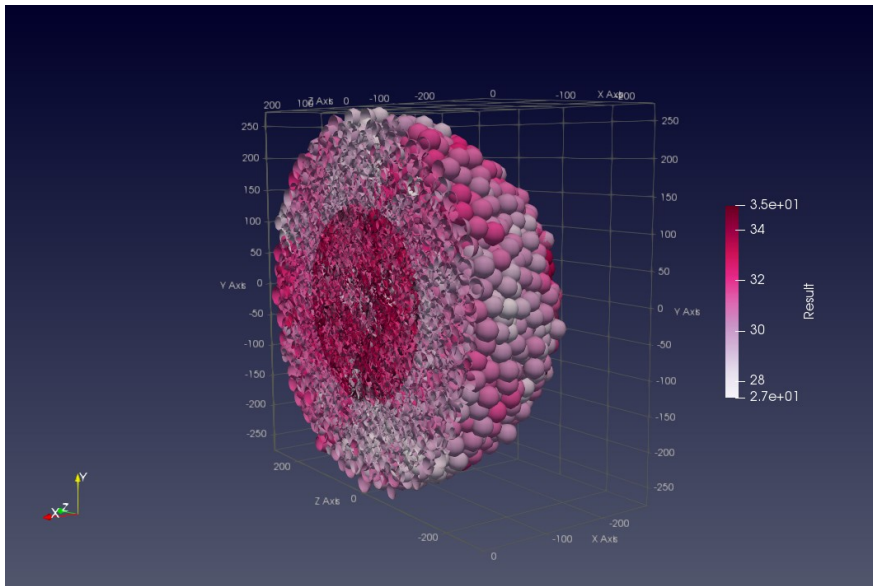
- PhD Computer Engineering (currently)

Openlab Summer Programme 2016



Summer Student Project

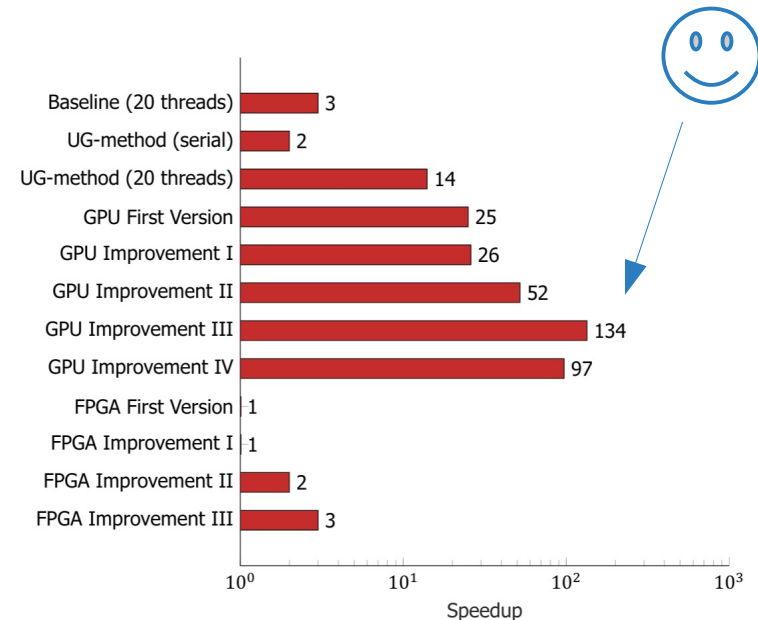
- Simulation platform for biomedical research (BioDynaMo)
- Integrated ROOT I/O for back-up & restore



Lightning talk winner :-D

Technical Studentship → Fellow

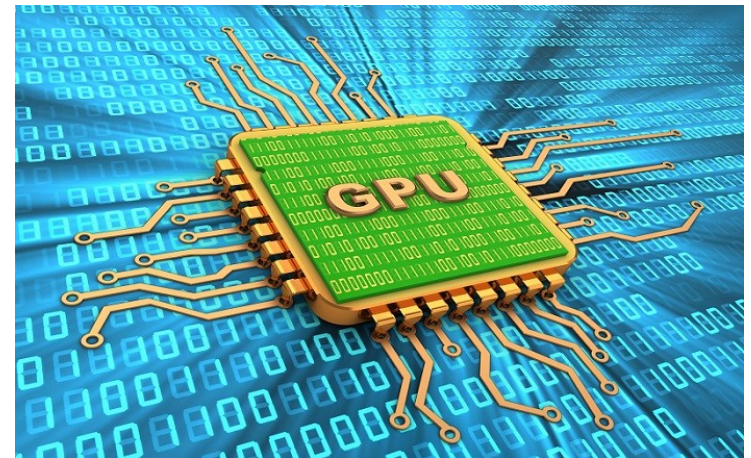
- Continued on the same project as a Technical Student
 - Implemented visualization
 - More ROOT
 - **Mainly: GPU & FPGA acceleration**
- Continued even further as a Fellow
 - Even more ROOT
 - **(Heterogeneous) distributed runtime**



– More on BioDynaMo coming Monday! –

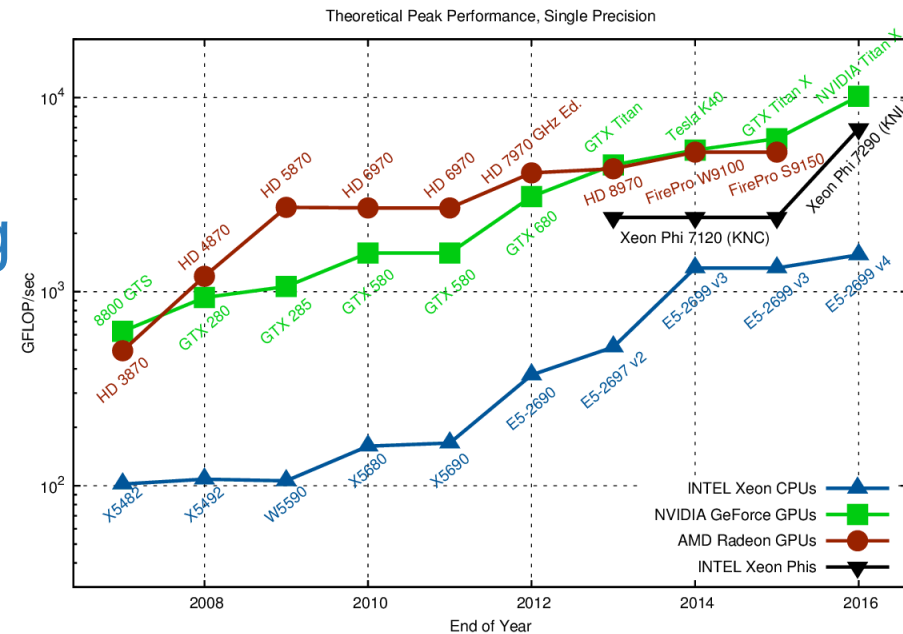
Today's Talk

- What are GPUs?
- Why do we need them (at CERN)?
- How do we use them?
- **Hands-on session**



What are GPUs?

- **Graphics Processing Units**
- Name from the 'old days' when only used for graphics processing
- Increasingly more powerful
 - **General-purpose** use cases were coming up
- Offloading computational intensive workloads to GPUs



<https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/>

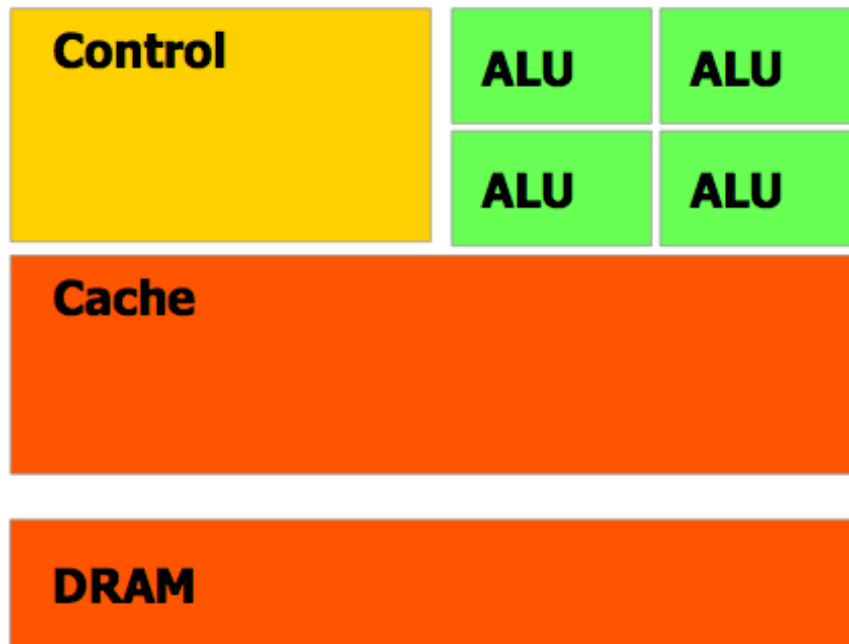
How do GPUs compare against regular old CPUs?

GPU vs CPU

A short, but convincing, demonstration...

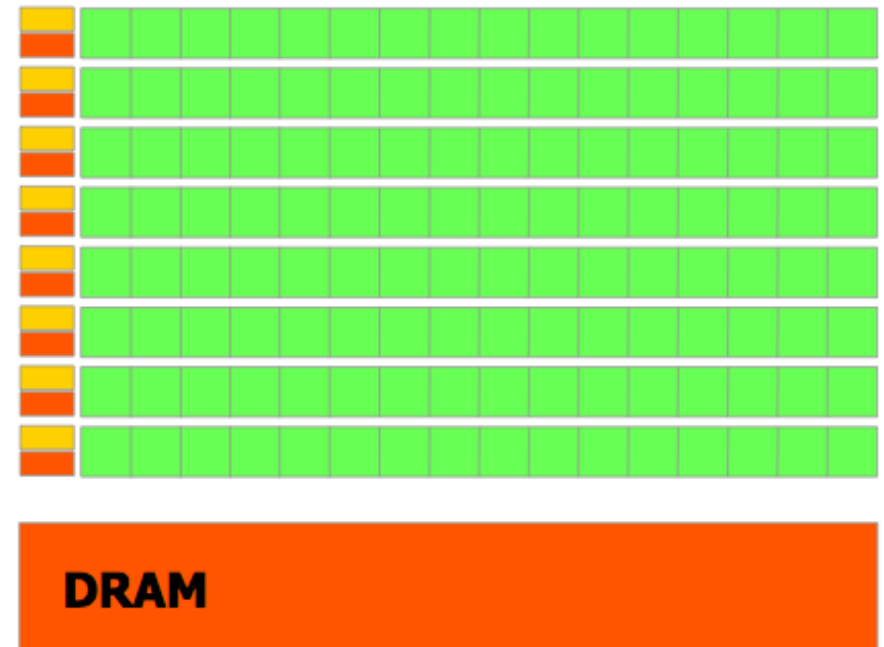
<https://www.youtube.com/watch?v=-P28LKWTzrI>

GPU vs CPU



CPU

- Out of order execution
- Superscalar architecture



GPU

- Single Instruction Multiple Threads (SIMT)
- Great for massively parallel computations

For certain workloads, GPUs can outperform a small CPU-only cluster!

GPU vs CPU: Recent Experience

1x IBM SC821LC (login node)

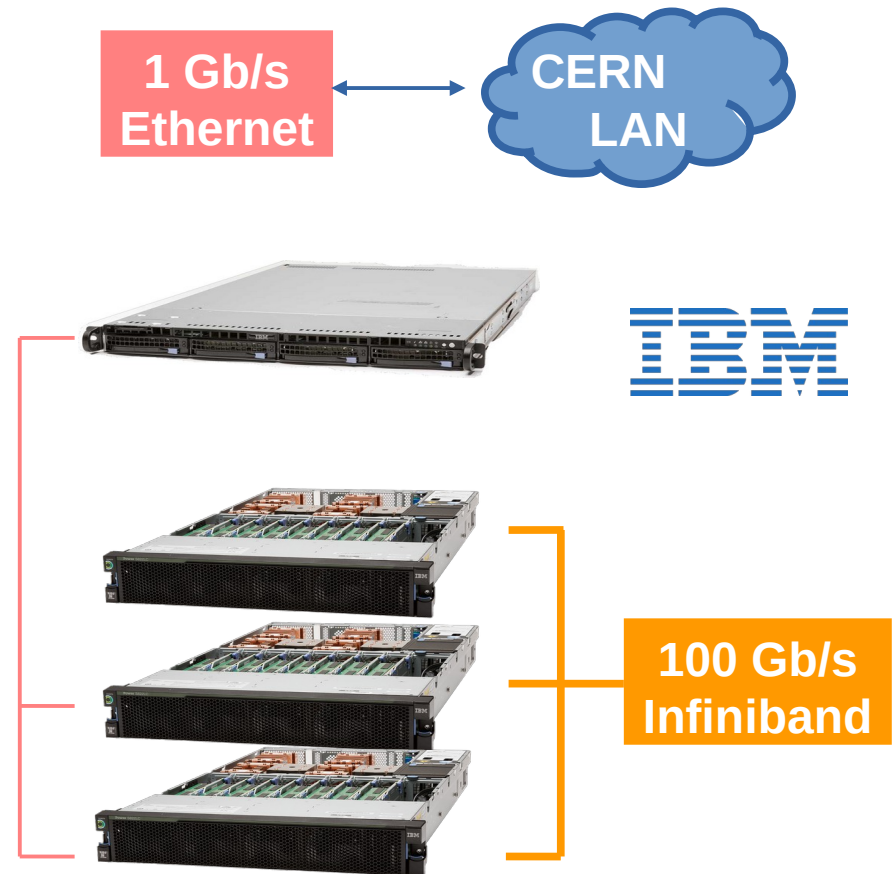
- 1x POWER8 socket (=8 cores)
- 64 GB DDR4

3x IBM SC822LC (worker nodes)

- 2x POWER8 sockets (=16 cores)
- 4x NVIDIA P100 GPUs
- CPU ↔ GPU NVLink
- 256GB DDR4

**Delivered roughly the same
performance as a 256-CPU cluster!
+
~10X more energy-efficient**

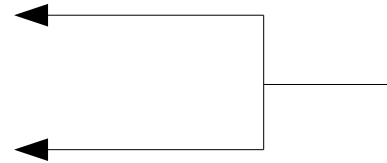
Use case: distributed training in deep learning



GPU Computing Basics

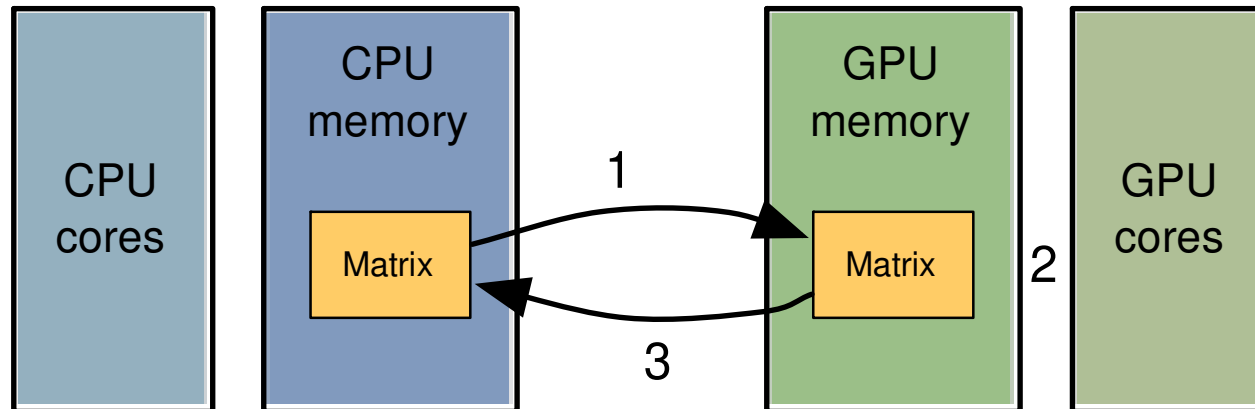
Computation is offloaded to the GPU in three steps:

1. CPU → GPU data transfer
2. GPU kernel execution
3. GPU → CPU data transfer



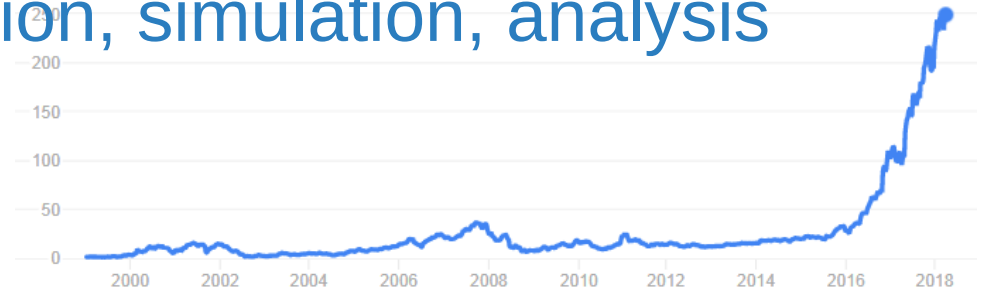
- Overhead
- Potential memory bottleneck

Why?



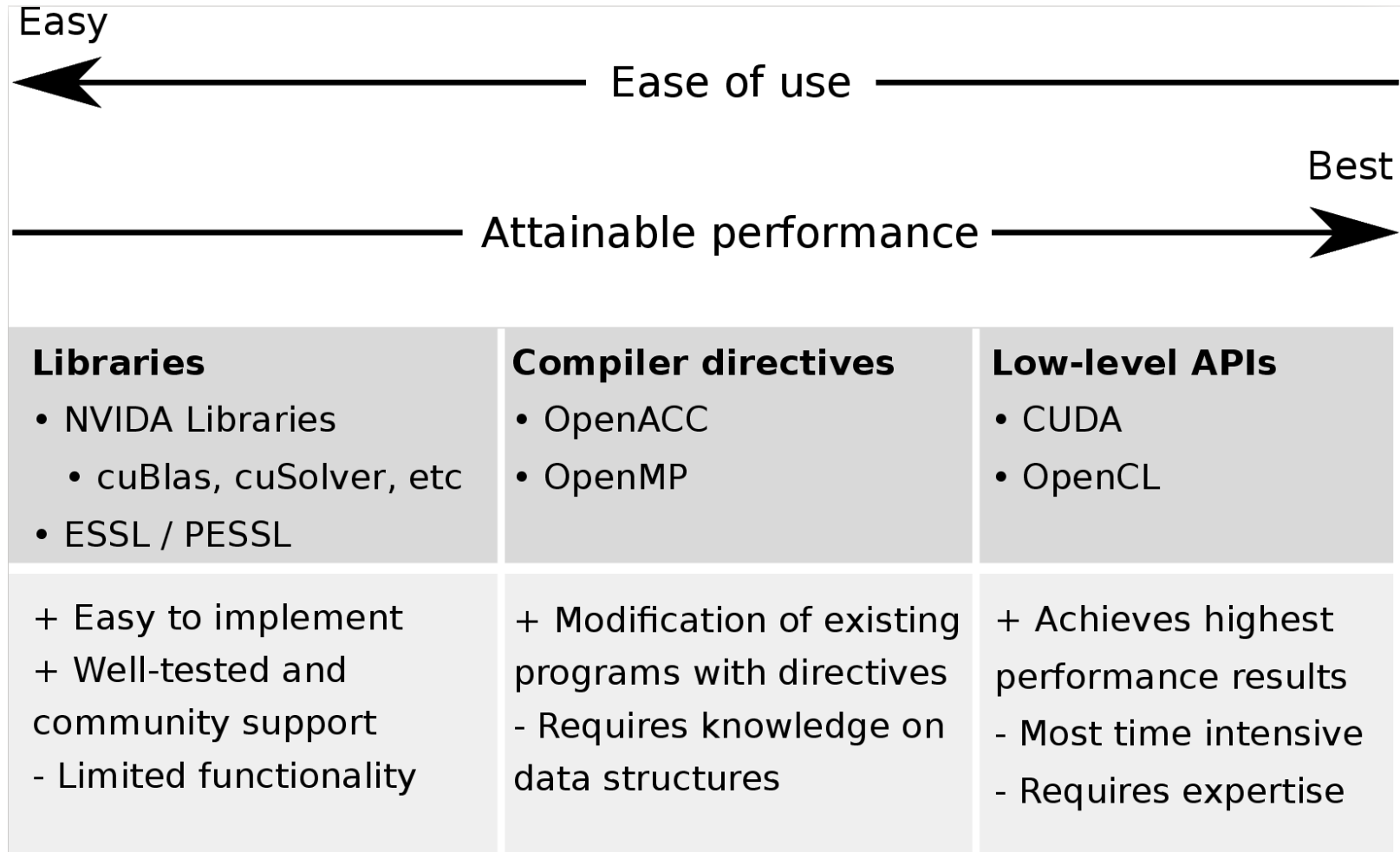
Why do we need GPUs?

- Reaching physics limits for CPUs
 - Multi-core era started nearly 2 decades ago
- Massively many-core era is the now (stock prices don't lie)
- Heterogeneous computing
- At CERN
 - Trigger, reconstruction, simulation, analysis
 - High Lumi



Nvidia's stock price has risen 1,900% over the past 5 years (data from 2018)

How do we use GPUs?



SAXPY

Single-Precision $A*X$ Plus Y

$$z = ax + y$$

x, y, z : vector

a : scalar

```
void saxpy(int n, float a, float * restrict x, float * restrict y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
```

Regular C implementation of saxpy

Using Libraries

Many popular frameworks with a GPU back-end rely on CUDA libraries:

- Deep Learning: Tensorflow, Keras, PyTorch
- Molecular Dynamics: NAMD, LAMMPS
- General Scientific Libraries: MATLAB, R

```
int N = 1<<20;
cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);
cublasShutdown();
```

Saxpy with CUBLAS library

Using Compiler Directives

Goal: insert directives into existing code base, without changing the code

+ Very simple in use

- Increasingly more difficult to use in more complex (or messier) codebases

```
void saxpy(int n, float a, float * restrict x, float * restrict y)
{
#pragma acc kernels
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
```

Saxpy with OpenACC

Using Low-Level APIs

```
__global__
void saxpy(int n, float a, float * restrict x, float * restrict y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

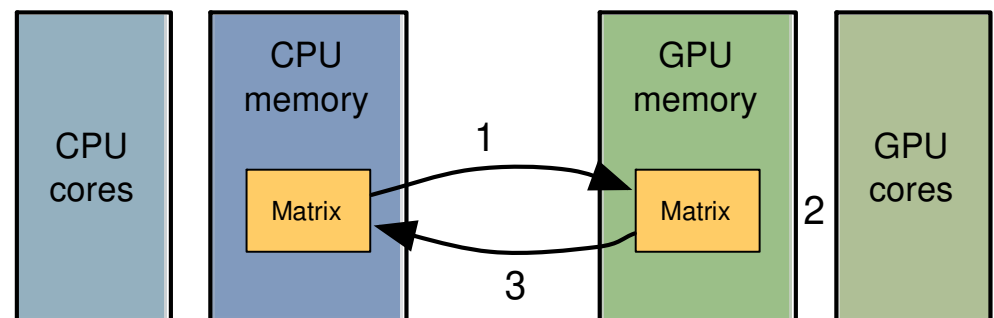
...
int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

- Steepest learning curve of all methods
- Often most rewarding

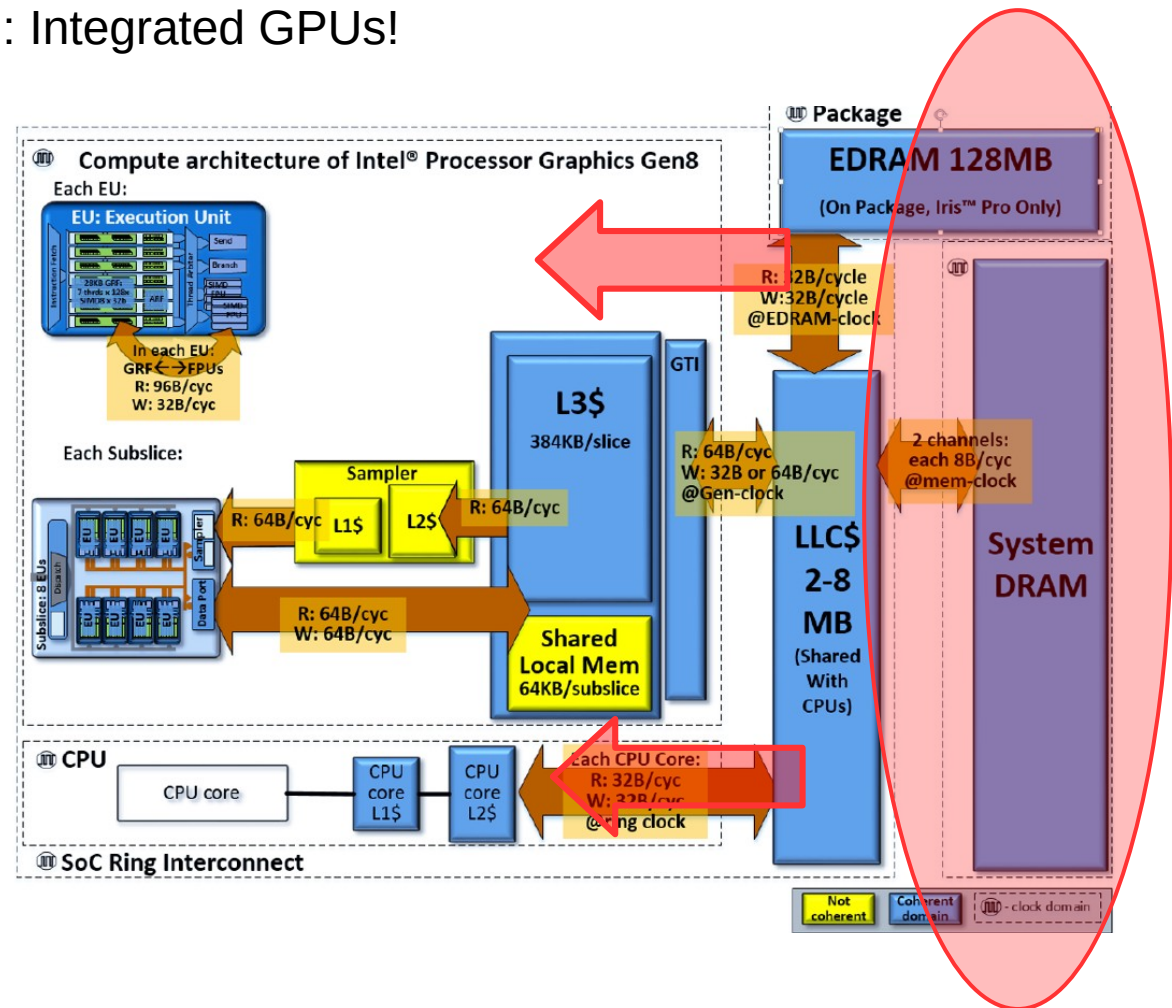
Saxpy with CUDA



Intel Integrated GPU

Intel Graphics Technology (GT) : Integrated GPUs!

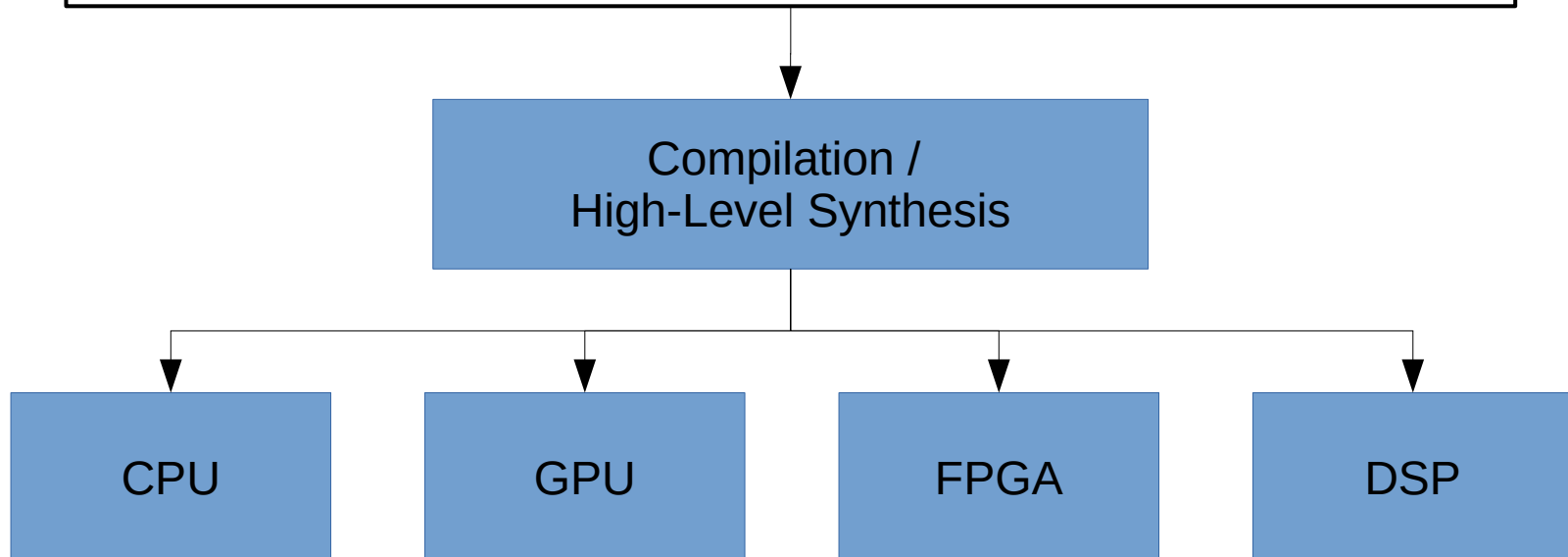
- CPU and GPU on same die
- Share same DRAM
- Often less powerful than dedicated GPUs
- Supports OpenCL for programming the GPU



OpenCL: Open Computing Language

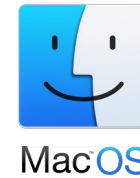
- Framework to program on heterogeneous platform (CPU, GPU, DSP, FPGA...)

```
__kernel void saxpy(float a, __global float* X,  
                  __global float* Y) {  
    const int i = get_global_id(0);  
    Y[i] += a * X[i];  
}
```



Operating System Survey

Quite a spread usage of operating systems
~75% managed to get it to work



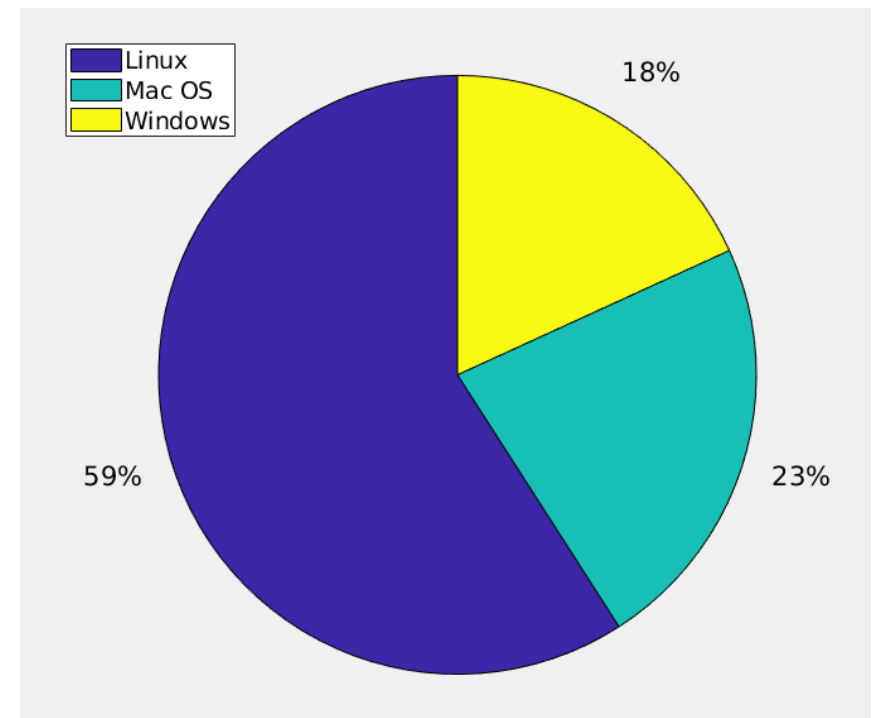
Why does the OS even matter?

Mac OS: the only OS that comes with OpenCL pre-installed

Linux: does not come with OCL pre-installed

Windows:

- Virtual environment cuts off access to (integrated) GPU
- Upcoming support for native Linux subsystem
- Possible to run natively, but not tested!



OSs of Summer Students 2019

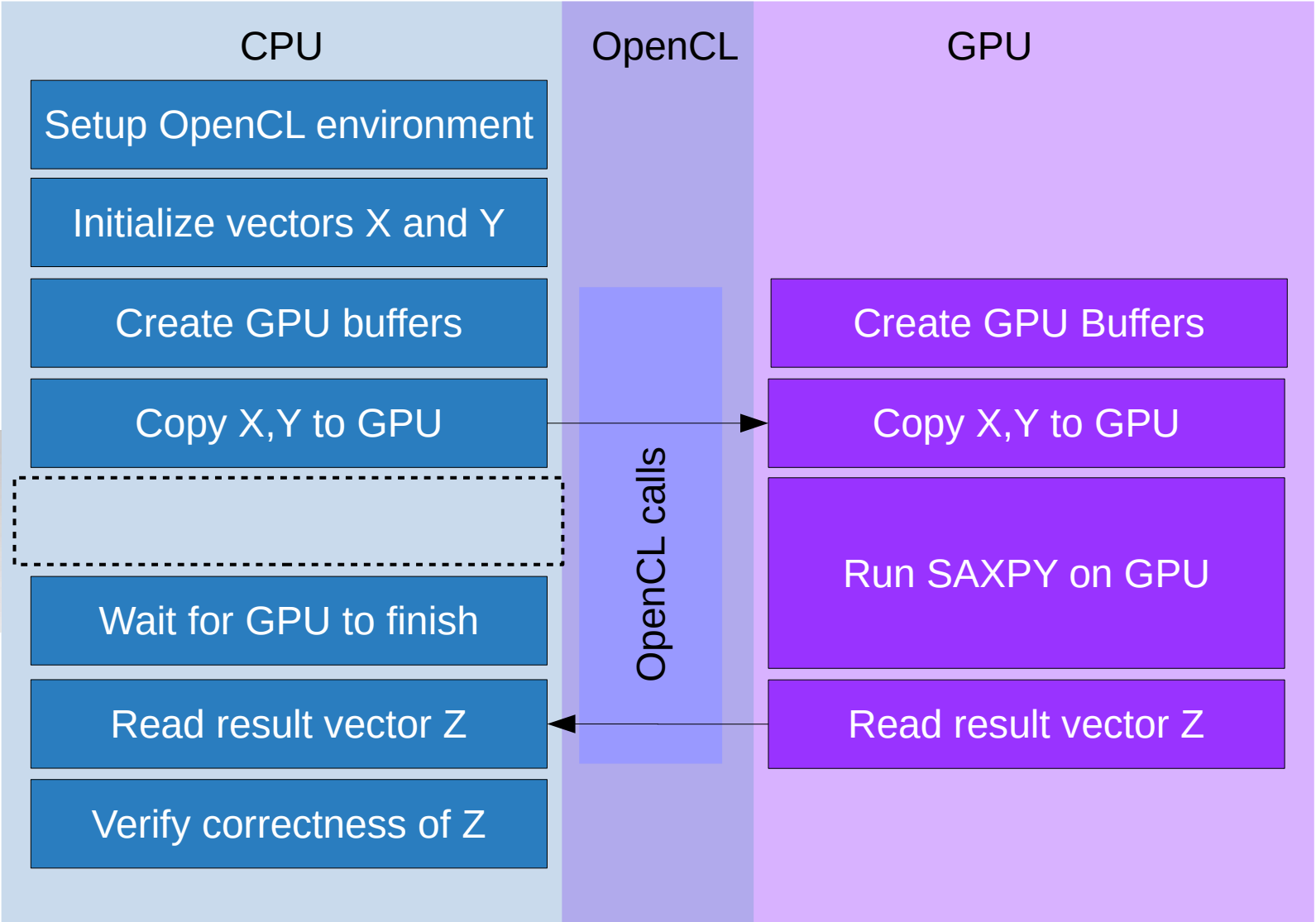
Saxpy on GPU

...with OpenCL

$$z = ax + y$$

x,y,z: vector
a : scalar

time
↓

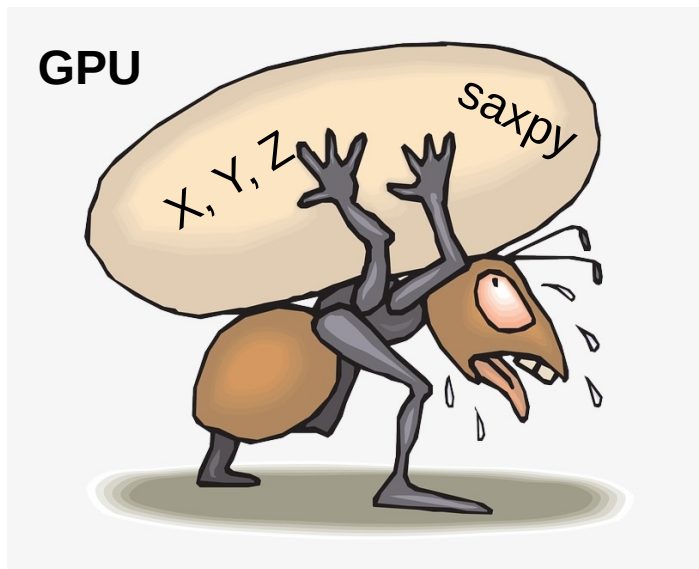


↓
How to capitalize on that?

How to improve?



```
__kernel void saxpy(float a, __global float* X,  
                  __global float* Y) {  
    const int i = get_global_id(0);  
    Y[i] += a * X[i];  
}
```



CPU



On My Machine

```
ahmad@ahmad:~/saxpy-benchmark/src$ ./saxpy_ocl1
Platform "Intel(R) OpenCL". Devices:
- [gpu ] Intel(R) Corporation: Intel(R) HD Graphics
  (Max compute units: 23, max work group size: 256)
- [cpu ] Intel(R) Corporation: Intel(R) Core(TM) i7-7
  (Max compute units: 4, max work group size: 8192)

Using Intel(R) Corporation Intel(R) HD Graphics
GPU execution time is: 61.759 ms
Errors: 0
```

```
ahmad@ahmad:~/saxpy-benchmark/src$ ./saxpy_cpu
N: 67108864
CPU execution time = 57.8496 ms
Errors: 0
```

On my machine GPU is about as fast as my CPU for saxpy

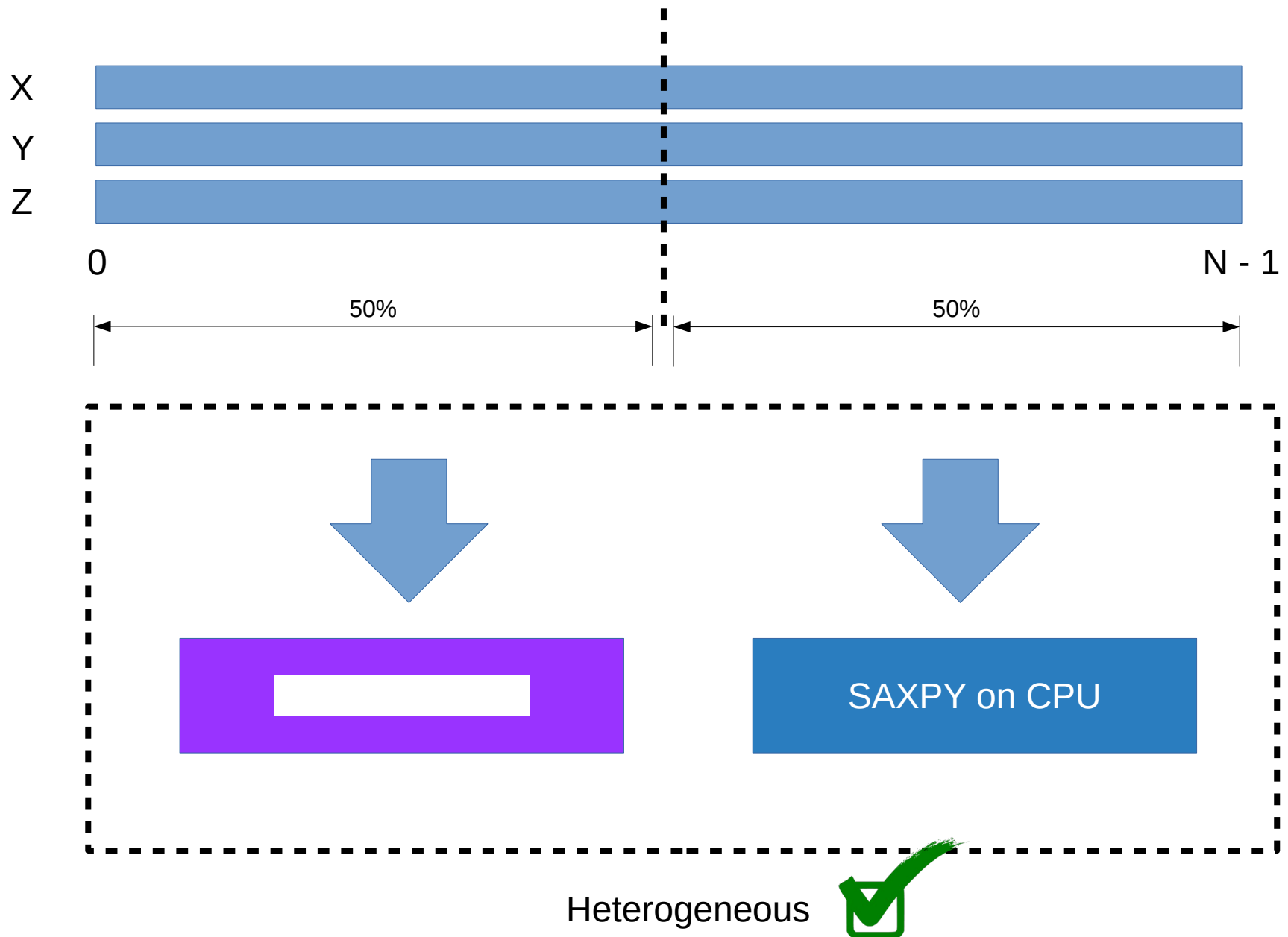
On Another Machine

```
ahhesam@tlab-gpu-gtx1080ti-09:~/saxpy-benchmark/src$ ./saxpy_ocl1
Platform "NVIDIA CUDA". Devices:
- [gpu ] NVIDIA Corporation: GeForce GTX 1080 Ti
  (Max compute units: 28, max work group size: 1024)

Using NVIDIA Corporation GeForce GTX 1080 Ti
GPU execution time = 2.238 ms
Errors: 0
ahhesam@tlab-gpu-gtx1080ti-09:~/saxpy-benchmark/src$ ./saxpy_cpu
N: 67108864
CPU execution time = 56.7027 ms
Errors: 0
```

On another machine the GPU is much faster than the CPU for saxpy

Heterogeneous Execution



I want you to observe:

```
ahmad@ahmad:~/saxpy-benchmark/src$ ./saxpy ocl1
Platform "Intel(R) OpenCL". Devices:
- [gpu ] Intel(R) Corporation: Intel(R) HD Graphics
  (Max compute units: 23, max work group size: 256)
- [cpu ] Intel(R) Corporation: Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz
  (Max compute units: 4, max work group size: 8192)

Using Intel(R) Corporation Intel(R) HD Graphics
GPU execution time = 121.896 ms
Errors: 0

ahmad@ahmad:~/saxpy-benchmark/src$ ./saxpy ocl1 hg
Platform "Intel(R) OpenCL". Devices:
- [gpu ] Intel(R) Corporation: Intel(R) HD Graphics
  (Max compute units: 23, max work group size: 256)
- [cpu ] Intel(R) Corporation: Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz
  (Max compute units: 4, max work group size: 8192)

Using Intel(R) Corporation Intel(R) HD Graphics
CPU execution time = 59.187 ms
GPU execution time = 63.403 ms
Errors: 0
```

GPU-only runtime

heterogeneous runtime

And no, this is not a solution:

```
cout << "CPU execution time = 59.187 ms" << endl;
cout << "GPU execution time = 63.403 ms" << endl;

;-)
```

Hands-On Session

Current setup

OS X: run natively

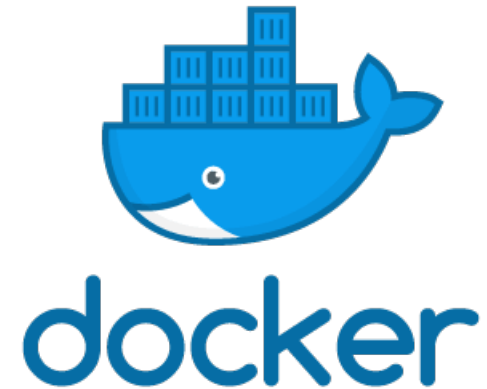
Linux: run in Docker container (useful commands in the course materials!)

Windows: look for someone with Linux / OSX or run natively if you dare...

NB: Make sure the source files are up to date!

Linux: `docker pull ahesam/intro-gpu`

OS X : `git pull`



Code available at (already in docker container):

<https://github.com/Senui/saxpy-benchmark/>

(forked repo)

HOST



/usr

/bin

...

inside_docker



/tutorial



GPU Driver

Demo Docker setup

Hands-On Session



Go through the code and understand it by reading the comments:

`saxpy-benchmark/src/saxpy_ocl1_hg.cpp`

OpenCL API: <https://github.com/khronos.org/OpenCL-CLHPP/>

Exercise: Heterogeneous Runtime

Edit the file such that:

- Half of the computation on GPU
- Other half on CPU
- Add timer for CPU execution (GPU timer already there)
- Error stays 0

Follow-up exercise (if time permits):

Profile the time it takes for data transfers (CPU → GPU, and GPU → CPU),
And compare them against the execution times

Q: Are they what you expect? Why (not)?

Q: How could you effectively 'hide' the data transfers?

Q: How would your observations differ with a dedicated GPU?

Useful Commands

- The source files can be found in:

```
saxpy-benchmark/src
```

- You will only need to edit: `saxpy_ocl1_hg.cpp`
- To compile your program simply run `make`
- To run your program run:

```
./saxpy_ocl1_hg
```

in the src directory

QUESTIONS?

Problems

- ERROR: clGetPlatformIDs(code: -1001)
- Most likely because you have an older Intel CPU that is not supported with OpenCL

Is development for Ivy Bridge architecture going to start soon? #128

 Closed ivanmlerner opened this issue on Feb 9 · 2 comments



ivanmlerner commented on Feb 9

+ 😊 ...

Hello, according to intel's page, this driver stack will substitute both closed source drivers and beignet, when is support for Ivy Bridge going to get in?



bfliflet commented on Feb 9

Contributor + 😊 ...

There are no plans that I am aware of to internally back port this library for Gen7 class (ie. Ivybridge and Haswell) hardware. There is a fair amount of runtime work but the bigger effort would be related to the compiler itself which was completely rearchitected in the Gen8/9 timeframe. That said, there's nothing prohibiting the underlying compute-runtime SW architecture from supporting it through community provided submissions.

Assi
No o
—
Labe
None
—
Proje
None
—
Miles
No m
—
Notif