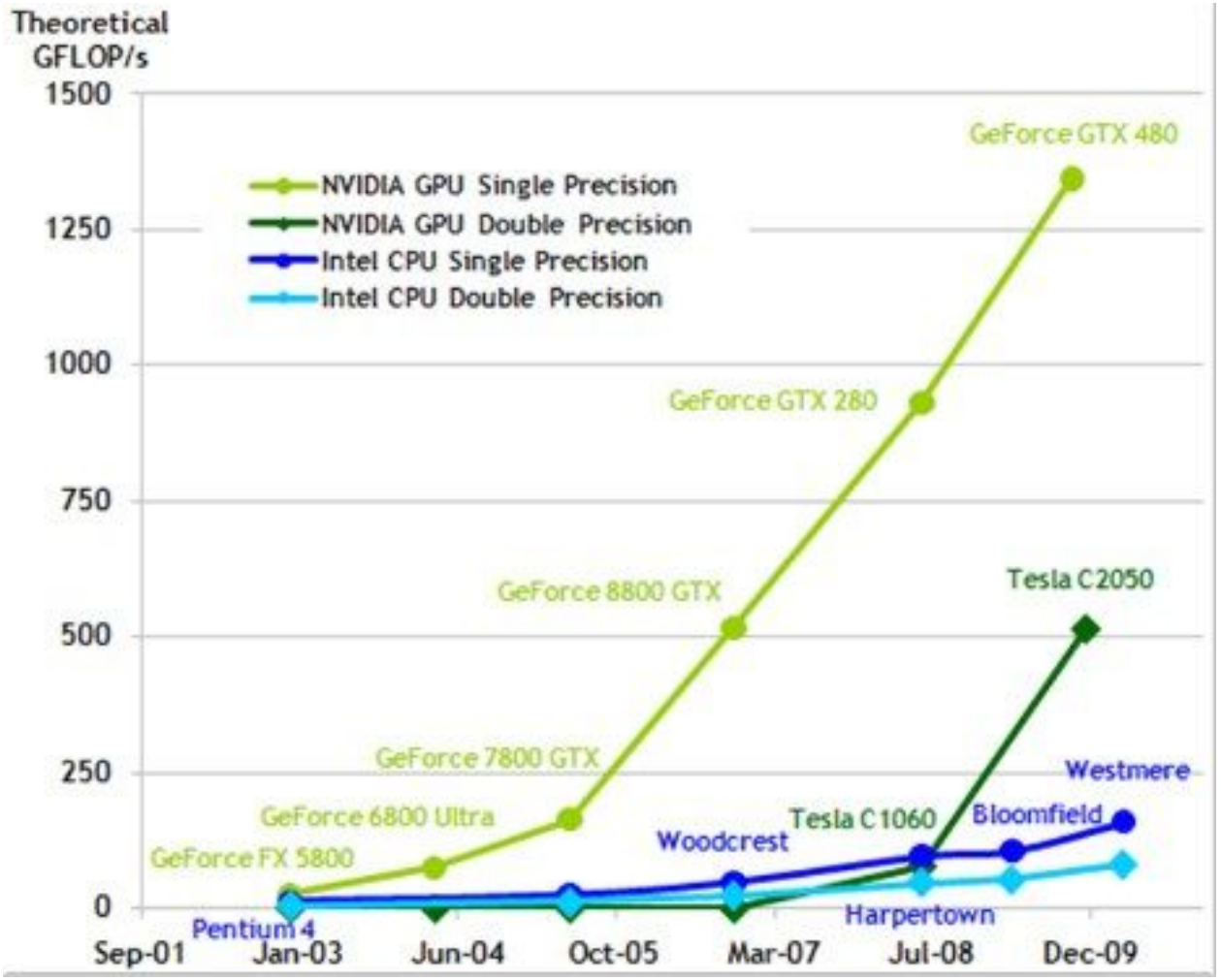
# GPU computing and R

Willem Ligtenberg

OpenAnalytics
willem.ligtenberg@openanalytics.eu

August 16, 2011

# Why GPU computing?



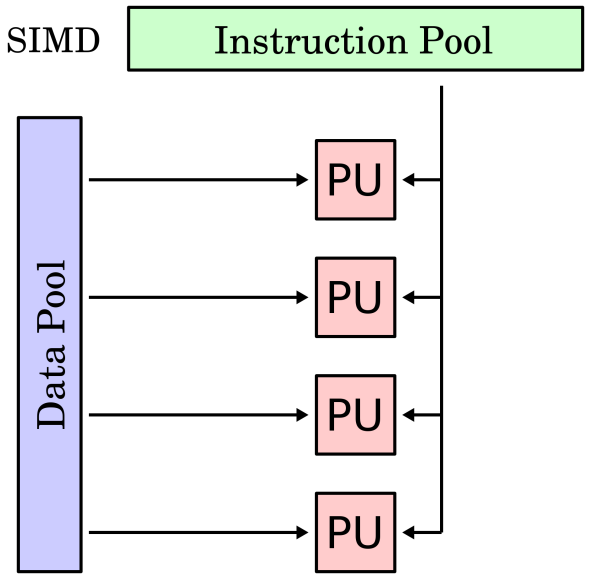

http://theinf2.informatik.uni-jena.de/Lectures/Programming+with+CUDA/SS+2011.html

# When to use GPU computing

GPU's are specifically well suited for:

- (Large) matrix operations

Streaming processors are SIMD (Same Instruction Multiple Data)



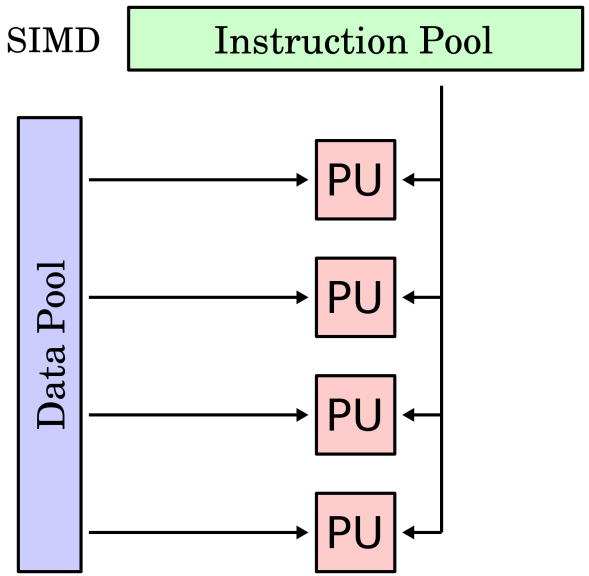

http://en.wikipedia.org/wiki/File:SIMD.svg

# When to use GPU computing

GPU's are specifically well suited for:

- (Large) matrix operations
- (Embarrassingly) parallel operations

Streaming processors are SIMD (Same Instruction Multiple Data)




http://en.wikipedia.org/wiki/File:SIMD.svg

# Introduction to GPU computing and OpenCL

- Initially GPU computing was performed by reshaping problems into texture operations.

# Introduction to GPU computing and OpenCL

- Initially GPU computing was performed by reshaping problems into texture operations.
- CUDA and FireStream were developed to aid developers.

# Introduction to GPU computing and OpenCL

- Initially GPU computing was performed by reshaping problems into texture operations.
- CUDA and FireStream were developed to aid developers.
- Problem they are brand specific

# Introduction to GPU computing and OpenCL

- Initially GPU computing was performed by reshaping problems into texture operations.
- CUDA and FireStream were developed to aid developers.
- Problem they are brand specific
- Enter OpenCL

# What is OpenCL

OpenCL is an industry standard framework for programming computers composed of a combination of CPU's, GPU's and other processors.

# What does that mean?

- OpenCL allows you to make optimal use of the different computational components in one system.

# What does that mean?

- OpenCL allows you to make optimal use of the different computational components in one system.
- Write code that runs on multiple (multi core) platforms e.g. GPU and CPU.

# Packages that make use of the GPU

- gputools

# Packages that make use of the GPU

- gputools
- rgpu

# Packages that make use of the GPU

- gputools
- rgpu
- cudaBayesreg

# gputools

Provides GPU implementations of various statistical algorithms.
Restricted to NVidia cards, because it uses CUDA.

# rgpu

R/GPU is a user-friendly package that can evaluate any given R expression by making transparent use of an NVIDIA Graphics Processing Unit (GPU) through CUDA.
Not actively developed anymore.

# cudaBayesreg

Implements the rhierLinearModel from the bayesm package using nVidia's CUDA language and tools to provide high-performance statistical analysis of fMRI voxels.

# The basics of OpenCL

- Discover the components in the system

# The basics of OpenCL

- Discover the components in the system
- Probe characteristic of these components

# The basics of OpenCL

- Discover the components in the system
- Probe characteristic of these components
- Create blocks of instructions (kernels)

# The basics of OpenCL

- Discover the components in the system
- Probe characteristic of these components
- Create blocks of instructions (kernels)
- Set up and manipulate memory objects for the computation

# The basics of OpenCL

- Discover the components in the system
- Probe characteristic of these components
- Create blocks of instructions (kernels)
- Set up and manipulate memory objects for the computation
- Execute kernels in the right order on the right components

# The basics of OpenCL

- Discover the components in the system
- Probe characteristic of these components
- Create blocks of instructions (kernels)
- Set up and manipulate memory objects for the computation
- Execute kernels in the right order on the right components
- Collect the results

# Introducing ROpenCL

ROpenCL is an R library which provides a **user friendly** interface to the **OpenCL** library.

# Confession

# Confession

- I have used other programming languages...

# Confession

- I have used other programming languages...
- Python

# Confession

- I have used other programming languages...
- Python
- PyCUDA/PyOpenCL

# Confession

- I have used other programming languages...
- Python
- PyCUDA/PyOpenCL
- And I like it!

# PyOpenCL for R

- R deserves a library like that

# PyOpenCL for R

- R deserves a library like that
- Like Rcpp for OpenCL

# PyOpenCL for R

- R deserves a library like that
- Like Rcpp for OpenCL
- No need to worry about memory management, let ROpenCL manage it for you

# Code or didn't happen : Vector addition

```
// set and log Global and
  Local work size dimensions
szLocalWorkSize = 256;
szGlobalWorkSize =
  shrRoundUp((int)szLocalWorkSize, iNumElements);

// Allocate and initialize host arrays
srcA = (void *)malloc(sizeof(
  cl_float) * szGlobalWorkSize);
srcB = (void *)malloc(sizeof(
  cl_float) * szGlobalWorkSize);
dst = (void *)malloc(sizeof(
  cl_float) * szGlobalWorkSize);
shrFillArray((float*)srcA, iNumElements);
shrFillArray((float*)srcB, iNumElements);
```

```
library(ROpenCL)

a <- seq(11444777)/10
b <- seq(11444777)
out <- rep(0, length(a))
localWorkSize = 256
globalWorkSize =
  ceiling(length(a)/localWorkSize)*localWorkSize
```

# Code or didn't happen

```
//Get an OpenCL platform
ciErr1 = clGetPlatformIDs(1, &cpPlatform, NULL);
//Get the devices
ciErr1 = clGetDeviceIDs(cpPlatform,
  CL_DEVICE_TYPE_GPU, 1, &cdDevice, NULL);
//Create the context
cxGPUContext = clCreateContext(0, 1, &cdDevice,
  NULL, NULL, &ciErr1);
// Create a command-queue
cqCommandQueue = clCreateCommandQueue(cxGPUContext,
  cdDevice, 0, &ciErr1);
// Allocate the OpenCL buffer memory objects
  for source and result on the device GMEM
cmDevSrcA = clCreateBuffer(cxGPUContext,
  CL_MEM_READ_ONLY, sizeof(cl_float)
  * szGlobalWorkSize, NULL, &ciErr1);
cmDevSrcB = clCreateBuffer(cxGPUContext,
  CL_MEM_READ_ONLY, sizeof(cl_float)
  * szGlobalWorkSize, NULL, &ciErr2);
cmDevDst = clCreateBuffer(cxGPUContext,
  CL_MEM_WRITE_ONLY, sizeof(cl_float)
  * szGlobalWorkSize, NULL, &ciErr2);
```

```
#Get an OpenCL platform
platformIDs <- getPlatformIDs()
#Get the devices
deviceIDs <- getDeviceIDs(platformIDs[[1]])
#Create the context
context <- createContext(deviceIDs[[1]])
#Create a command-queue
queue <- createCommandQueue(context,deviceIDs[[1]])
#Allocate the OpenCL buffer memory objects
  for source and result on the device GMEM
inputBuf1 <- createBuffer(context,
  "CL_MEM_READ_ONLY", globalWorkSize, a)
inputBuf2 <- createBuffer(context,
  "CL_MEM_READ_ONLY", globalWorkSize, b)
outputBuf1 <- createBufferFloatVector(context,
  "CL_MEM_WRITE_ONLY", globalWorkSize)
```

# Code or didn't happen

```
// Create the program
cpProgram = clCreateProgramWithSource(cxGPUContext,
  1, (const char **)&cSourceCL, &szKernelLength,
  &ciErr1);
ciErr1 = clBuildProgram(cpProgram, 0, NULL,
  NULL, NULL, NULL);
// Create the kernel
ckKernel = clCreateKernel(cpProgram, "VectorAdd",
  &ciErr1);
// Set the Argument values
ciErr1 = clSetKernelArg(ckKernel, 0,
  sizeof(cl_mem), (void*)&cmDevSrcA);
ciErr1 |= clSetKernelArg(ckKernel, 1,
  sizeof(cl_mem), (void*)&cmDevSrcB);
ciErr1 |= clSetKernelArg(ckKernel, 2,
  sizeof(cl_mem), (void*)&cmDevDst);
ciErr1 |= clSetKernelArg(ckKernel, 3,
  sizeof(cl_int), (void*)&iNumElements);
```

```
kernel <- "__kernel void VectorAdd(__global const
  float* a, __global const int* b,
  __global float* c, int iNumElements)
{
    // get index into global data array
    int iGID = get_global_id(0);
    // bound check (equivalent to the limit on a
    //'for' loop for standard/serial C code
    if (iGID >= iNumElements)
    {
        return;
    }
    // add the vector elements
    c[iGID] = a[iGID] + b[iGID];
}"
kernel <- createProgram(context, kernel,
  "VectorAdd", inputBuf1, inputBuf2,
  outputBuf1, length(out))
```

# Code or didn't happen

```
// Asynchronous write of data to GPU device
ciErr1 = clEnqueueWriteBuffer(cqCommandQueue,
  cmDevSrcA, CL_FALSE, 0, sizeof(cl_float)
  * szGlobalWorkSize, srcA, 0, NULL, NULL);
ciErr1 |= clEnqueueWriteBuffer(cqCommandQueue,
  cmDevSrcB, CL_FALSE, 0, sizeof(cl_float)
  * szGlobalWorkSize, srcB, 0, NULL, NULL);
// Launch kernel
ciErr1 = clEnqueueNDRangeKernel(cqCommandQueue,
  ckKernel, 1, NULL, &szGlobalWorkSize,
  &szLocalWorkSize, 0, NULL, NULL);
// Synchronous/blocking read of results, and
  check accumulated errors
ciErr1 = clEnqueueReadBuffer(cqCommandQueue,
  cmDevDst, CL_TRUE, 0, sizeof(cl_float)
  * szGlobalWorkSize, dst, 0, NULL, NULL);
```

```
enqueueWriteBuffer(queue, inputBuf1,
  globalWorkSize, a)
enqueueWriteBuffer(queue, inputBuf2,
  globalWorkSize, b)
enqueueNDRangeKernel(queue, kernel,
  globalWorkSize, localWorkSize)
result <- enqueueReadBuffer(queue,
  outputBuf1, globalWorkSize, out)
```

# OpenCL

- A little over a week ago the OpenCL package has been published on CRAN by Simon Urbanek.

# OpenCL

- A little over a week ago the OpenCL package has been published on CRAN by Simon Urbanek.
- And currently it seems to be a very thin layer around OpenCL.

# OpenCL

- A little over a week ago the OpenCL package has been published on CRAN by Simon Urbanek.
- And currently it seems to be a very thin layer around OpenCL.
- The goal of ROpenCL is to abstract a little more, like PyOpenCL.

# Contact details

Willem Ligtenberg
willem.ligtenberg@openanalytics.eu

install.packages("ROpenCL", repos = "http://repos.openanalytics.eu", type = "source")