Introduction to GPU computing for statisticicans

Will Landau

Iowa State University

September 16, 2013

Outline

Outline

The single instruction, multiple data (SIMD) paradigm

- SIMD: apply the same command to multiple places in a dataset.

```
1 for(i = 0; i < 1e6; ++i)
2   a[i] = b[i] + c[i];
```

- On CPUs, the iterations of the loop run sequentially.
- With GPUs, we can easily run all 1,000,000 iterations simultaneously.

```
1 i = threadIdx.x;
2 a[i] = b[i] + c[i];
```

- We can similarly *parallelize* a lot more than just loops.

Parallel MCMC by Lee, Yau, Giles, and others

# chains	CPU time (min)	GTX 280 (min)	CPU time / GPU time
8	0.0166	0.0148	1.1
32	0.0656	0.0151	4
128	0.262	.0154	17
512	1.04	0.0174	60
2,048	4.16	0.0248	168
8,192	16.64	0.0720	230
32,768	66.7	0.249	268
131,072	270.3	0.970	279

Parallel sequential MC by Lee, Yau, Giles, and others

Sample size	CPU time (min)	GTX 280 (min)	CPU time / GPU time
8,192	4.44	0.0199	223.1
16,384	8.82	0.0355	263
32,768	17.7	0.0666	265
65,536	35.3	0.131	269
131,076	70.6	0.261	270.5
262,144	141	0.52	271.2

Parallel Bayesian EM by Suchard, Wang, Chan, and others

Sample size	cpu 1 (sec)	gpu 1 (sec)	CPU time / GPU Time
10^2	4.0	71.1	0.1
10^3	40.0	81.3	0.5
10^4	607.0	91.2	6.7
10^5	7793.0	129.6	60.1
10^6	78765.0	680.6	115.7

Other applications

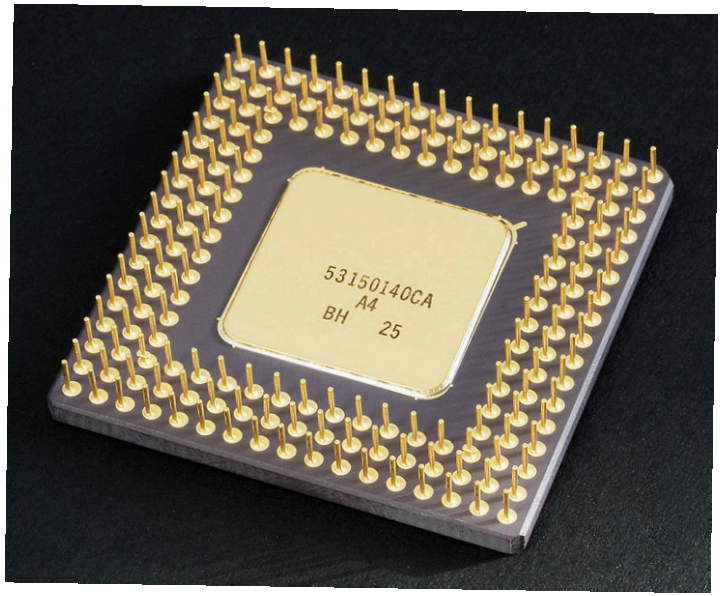
- Clustering
- Bootstrap
- Regression
- Matrix algebra
- EM Algorithm
- Rejection sampling
- Multiple testing
- Cross validation
- ...

Computer processors

- **Processing unit**: a computer chip that performs executive functions.
- **Core**: One of possibly many "sub-processors" placed on the same processing unit, each of which has the full functionality of the processing unit.

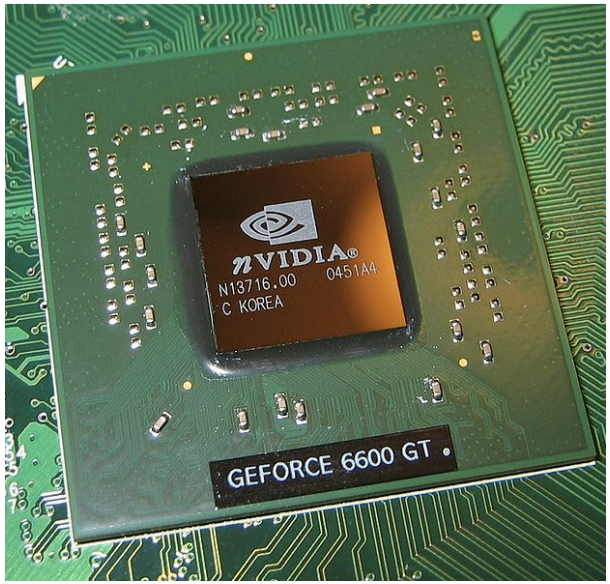
The Central Processing Unit (CPU)

- Regular computer processor.
- Allows parallelism, but not *massive* parallelism on its own.
- Usually contains 1 to 8 cores.
- Examples:
  - Intel 8086 (1979, x86)
  - Intel Core 2 Duo
  - Intel 80486DX2 (below)



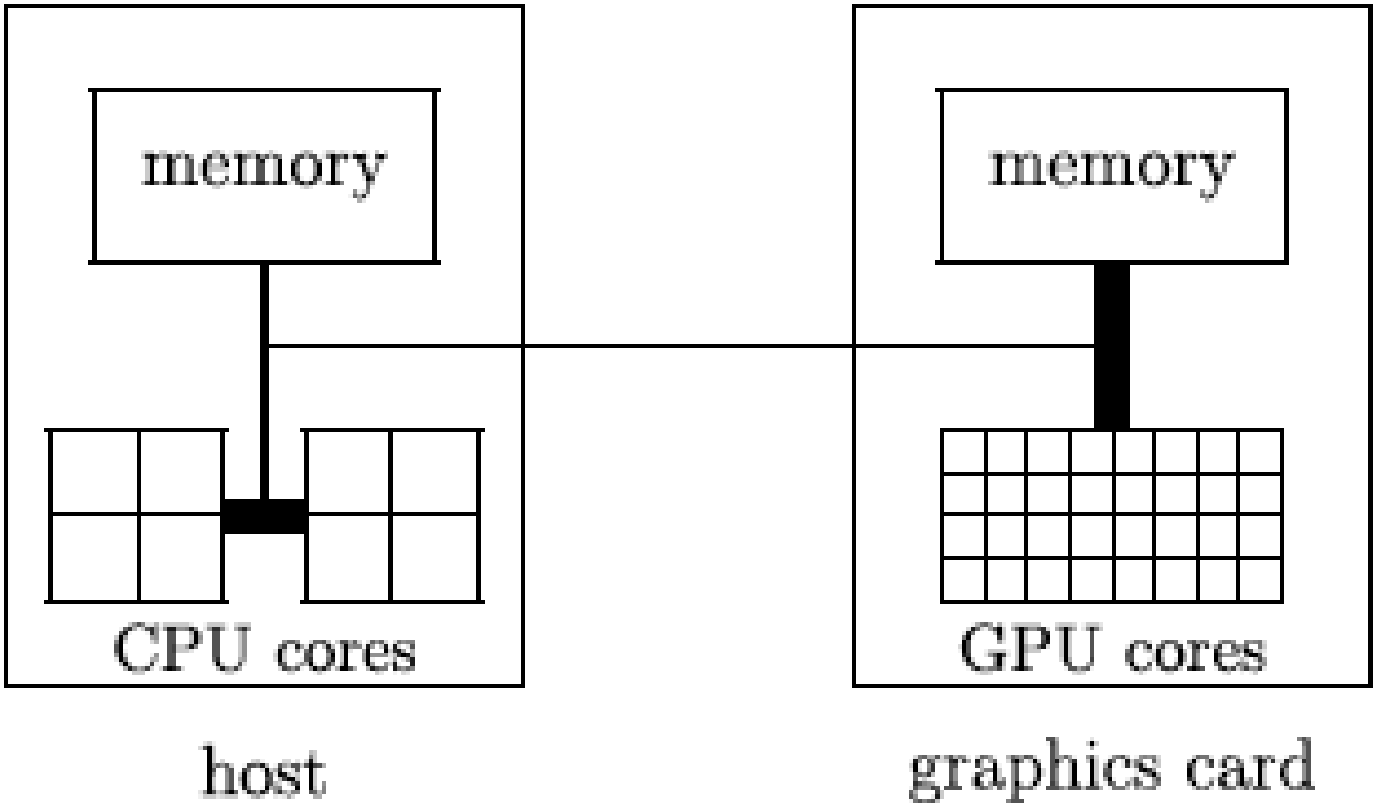
The Graphics Processing Unit (GPU)

- Processor in a video card or graphics card.
- Massively parallel: originally designed to speed up graphics throughput in video games.
- Cannot run by itself. Needs to be hooked up to a computer with a CPU.
- Contains several hundred cores.
- Higher memory bandwidth than a CPU.
- Examples:
  - NVIDIA GeForce 6600 (bottom left)
  - NVIDIA GeForce GTX 580
  - NVIDIA Tesla M2070 (on our GPU-enabled machines)



CPU / GPU cooperation

- The CPU ("host") is in charge.
- The CPU sends computationally intensive instruction sets to the GPU ("device") just like a human uses a pocket calculator.



More on parallelism

- **Parallelism**: the practice of running multiple calculations simultaneously.
- The architecture of GPUs is extremely well-suited to massively parallel workflows.
- Note: GPU parallelism is one of many kinds of parallelism. Others include:
  - Posix threads (CPU parallelism)
  - parallel cloud computing
  - openMP parallelism
  - openMP parallelism

GPU parallelism speeds up calculations

- Amdahl's Law says that the maximum theoretical speedup (CPU time / GPU time) is

$$\frac{1}{1 - P + \frac{P}{N}}$$

  where:
  - P = fraction of the program (in terms of execution time) that can be parallelized
  - N = number of parallel processors
- As $N \to \infty$, Amdahl's quantity goes to

$$\frac{1}{1 - P}$$

- So if 99% of the program can be parallelized, we could theoretically have a 100-fold speedup.

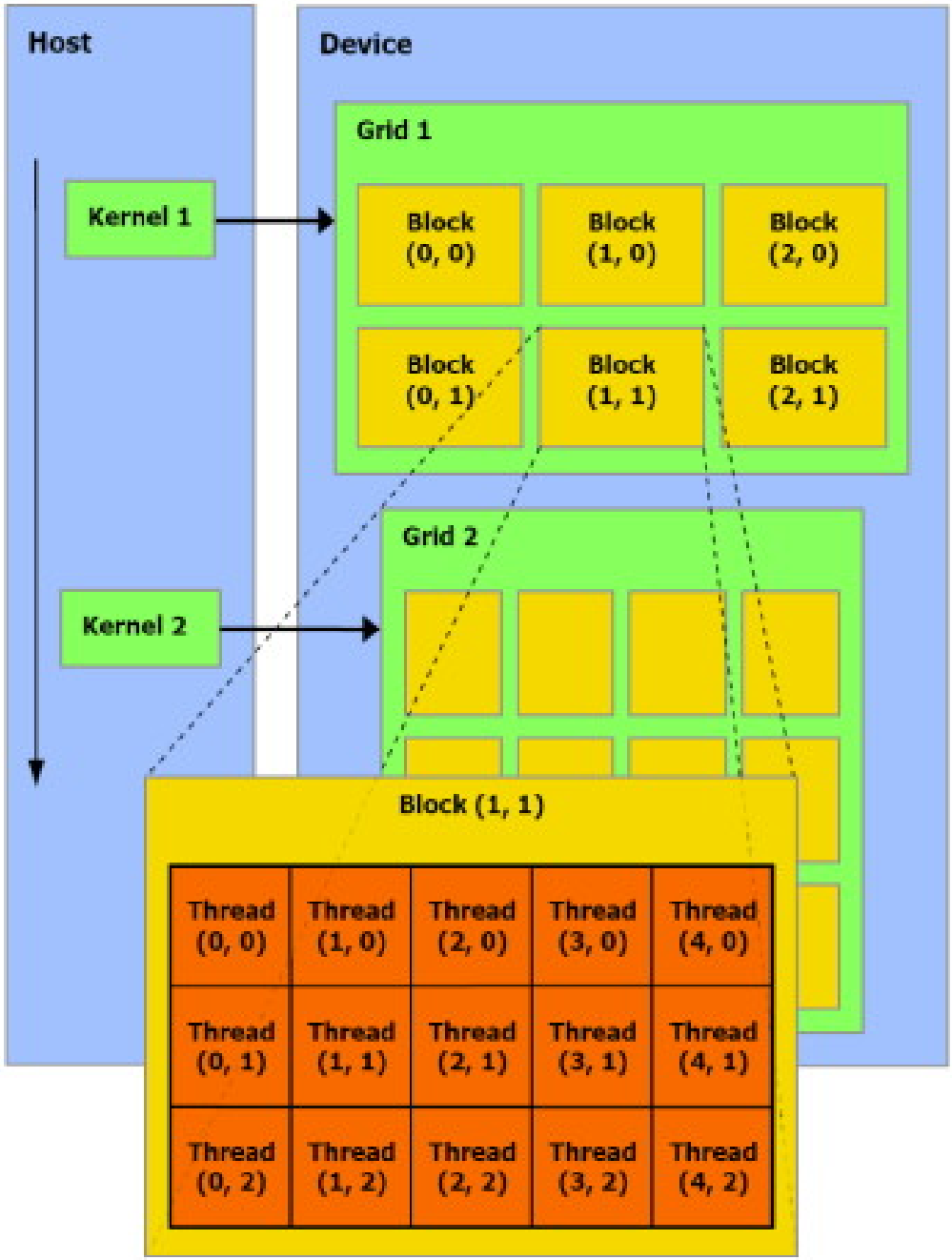
How GPU parallelism works

1. The CPU sends a command called a **kernel** to a GPU.
2. The GPU executes several duplicate realizations of this command, called **threads**.
   - These threads are grouped into bunches called **blocks**.
   - The sum total of all threads in a kernel is called a **grid**.

- Toy example:
  - CPU says something like, "Hey, GPU. Sum pairs of adjacent numbers. Use the array, (1, 2, 3, 4, 5, 6, 7, 8). Use 2 blocks of 2 threads each."
  - GPU thinks: "Sum pairs of adjacent numbers" is a kernel that I need to apply to the array, (1, 2, 3, 4, 5, 6, 8).
  - The GPU spawns 2 blocks, each with 2 threads:

Block	0		1	
Thread	0	1	0	1
Action	$1+2$	$3+4$	$5+6$	$7+8$

- All four actions above happen simultaneously.
- I could have also used 1 block with 4 threads and given the threads different paris of numbers.



Outline

CUDA: making a gaming toy do science

- GPUs were originally meant to speed up graphical displays for Windows OS and video games.

- **CUDA**: Compute Unified Device Architecture.
- Before CUDA, programmers could only program on GPUs using graphics languages, which are appropriate for video games but clumsy for science.
- CUDA devices support CUDA C, an extension of C for programs that use GPUs.

CUDA-enabled servers at Iowa State

- impact1.stat.iastate.edu (Red Hat Enterprise Linux Server release 6.2)
- impact2.stat.iastate.edu (CentOS release 6.3)
- impact3.stat.iastate.edu (Red Hat Enterprise Linux Server release 6.4)
- impact4.stat.iastate.edu (CentOS release 6.4)

Specs of our CUDA systems

- No graphical user interface or remote desktop capabilities. (Use the Linux command line.)
- 24 CPUs and 4 Tesla M2070 GPUs, where each GPU has 448 cores.:
- For more specs, log into impact1, 2, or 3 and enter into the command line:

```
1 cd /usr/local/NVIDIA_GPU_Computing_SDK/C/bin
      /linux/release
2 ./deviceQuery
```

Logging in

- Open a command line program (Terminal in Linux and Mac OS, Cygwin or MinGW in Windows).
- Enter:

```
1 ssh -p 323 ISU_ID@impact1.stat.iastate.edu
```

- Note: in general, the port number for `ssh` is not always 323.
- Refer to http://www.linuxcommand.org/ or contact me at landau@iastate.edu for help with the Linux command line.
- Contact Stat IT at statit@iastate.edu or me if:
  - You can't log on, or
  - You want to be able to log on without entering your password every time, or
  - You want to shorten `ssh -p 323 ISU_ID@impact1.stat.iastate.edu` into a more manageable alias on your local machine.

Important directories

- **/home/ISU_ID** Your private home folder on SMB (the collective storage system for all the stat dept linux servers). Files in here are stored remotely on SMB, not locally on impact1-3.
- **/Cyfiles/ISU_ID** Your private Cyfiles folder. Files in here are stored remotely on the Cyfiles server, not locally on impact1-3.
- **/tmp**
  - Everything in here is stored locally on impact1, etc., wherever you're logged in.
  - To avoid communicating over a network when you want fast computation, put large datasets here.
  - Note: **/tmp** automatically empties periodically.
- **/usr/local/NVIDIA_GPU_Computing_SDK**
  - Example CUDA C code. Stored locally on impact1, etc.
  - You do not have write privileges here.

Outline

GPUs, parallelism, and why we care

CUDA and our CUDA systems

GPU computing with R

GPU-enabled R packages

- `WideLM` - used to quickly fit a large number of linear models to a fixed design matrix and response vector.
- `magma` - a small linear algebra with implementations of backsolving and the LU factorization.
- `cudaBayesreg` - implements a Bayesian model for fitting fMRI data.
- `gcbd` - a Debian package for benchmarking linear algebra algorithms such as the QR, SVD and LU.factorizations.
- `gputools` - probably the most useful of these 5.

Contents of gputools

- Choose your device:

gputools function	CPU analog	Same usage?
chooseGpu()	none	NA
getGpuId()	none	NA

- Linear algebra:

gputools function	CPU analog	Same usage?
gpuDist()	dist()	no
gpuMatMult()	%*% operator	no
gpuCrossprod()	crossprod()	yes
gpuTcrossprod()	tcrossprod()	yes
gpuQr()	qr()	almost
gpuSolve()	solve()	no
gpuSvd()	svd()	almost

- Simple model fitting:

gputools function	CPU analog	Same usage?
gpuLm()	lm()	yes
gpuLsfit()	lsfit()	yes
gpuGlm()	glm()	yes
gpuGlm.fit()	glm.fit()	yes

- Hypothesis testing:

gputools function	CPU analog	Same usage?
gpuTtest()	t.test()	no
getAucEstimate()	???	???

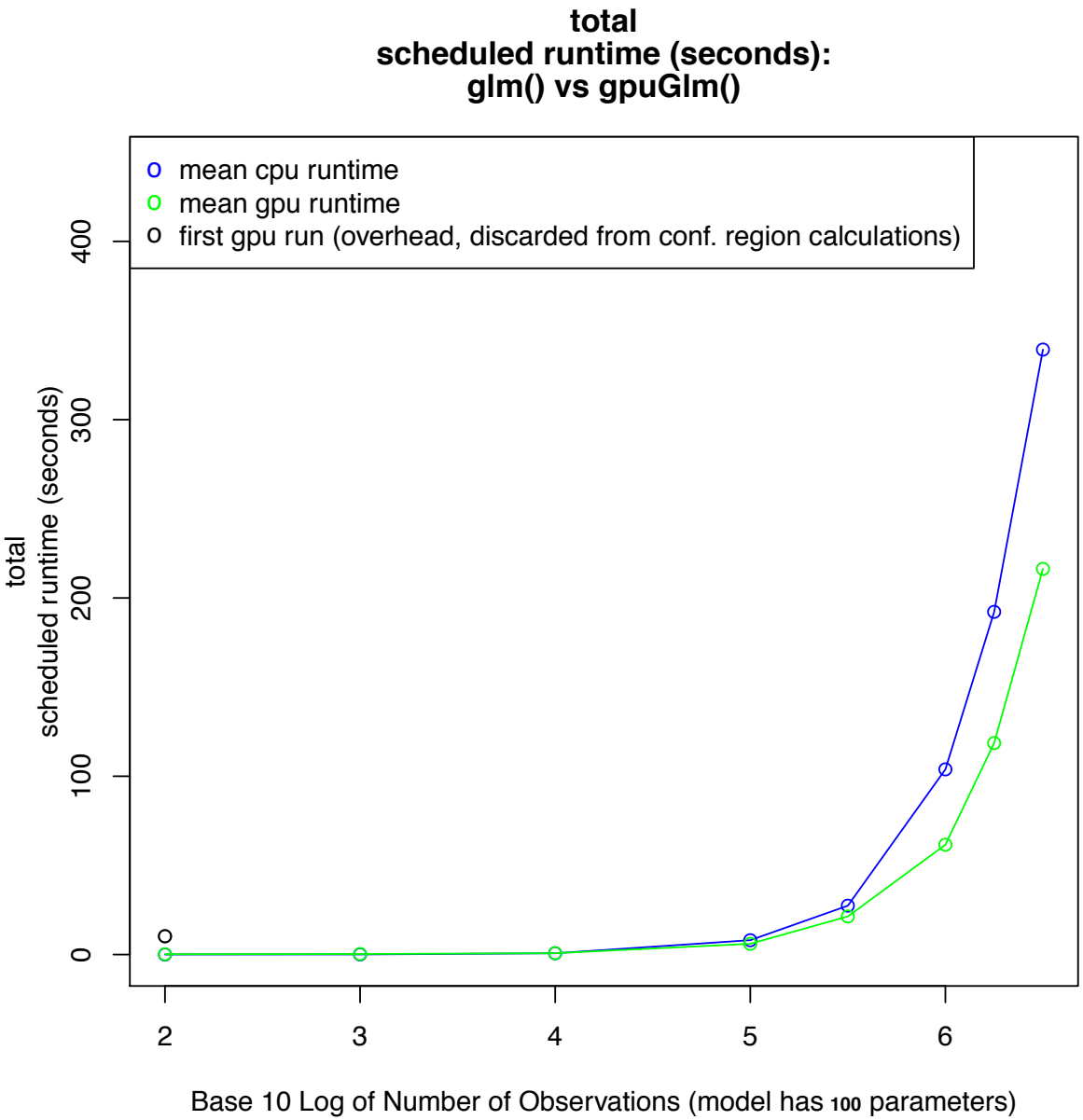
- Other routines:

gputools function	CPU analog	Same usage?
gpuHclust()	hclust()	no
gpuDistClust()	hclust(dist())	no
gpuFastICA()	fastICA() (fastICA package)	yes
gpuGranger()	grangertest() (lmtest package)	no
gpuMi()	???	???
gpuSvmPredict()	www.jstatsoft.org/v15/i09/paper	no
gpuSvmTrain()	www.jstatsoft.org/v15/i09/paper	no

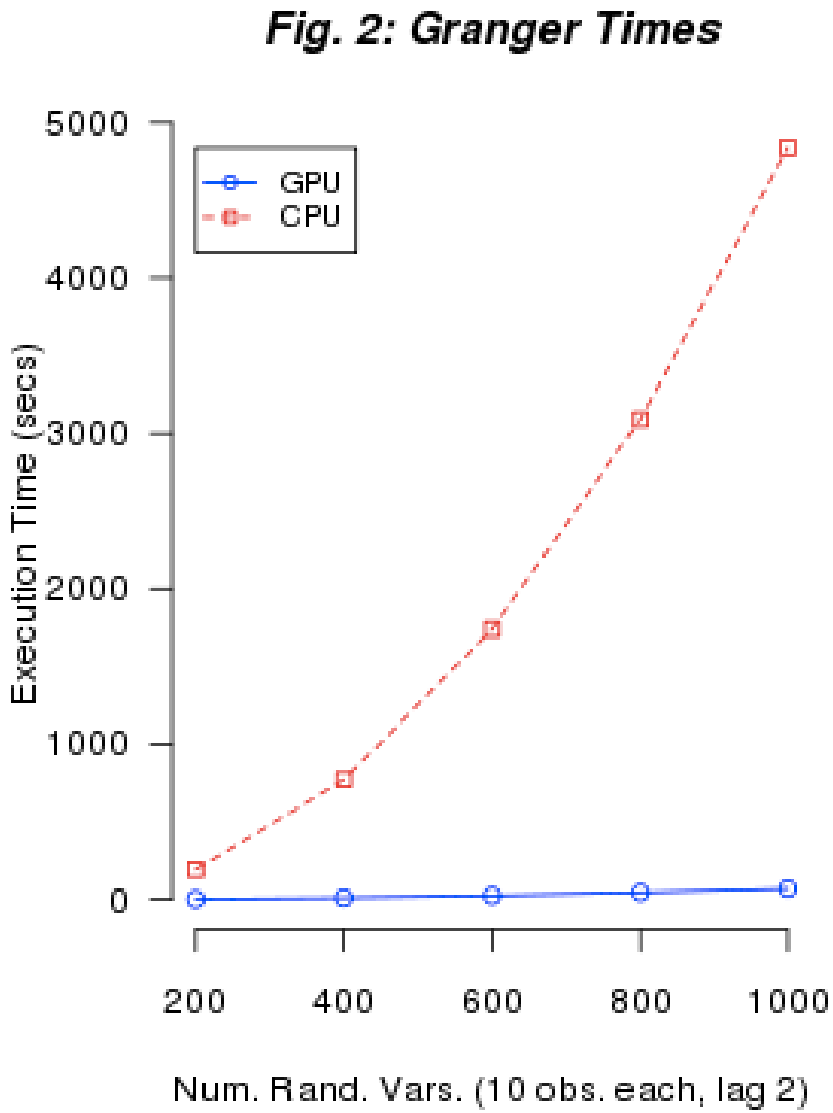
Example

```
> getGpuID()
[1] 0
> chooseGpu(3)
[[1]]
[1] 3
> getGpuID()
[1] 3
> A <- matrix(runif(1e7), nrow = 1e4)
> B <- matrix(runif(1e7), nrow = 1e4)
> ptm <- proc.time(); C <- gpuMatMult(A, B);
> proc.time() - ptm
   user  system elapsed
  2.959   2.190   5.159
> ptm <-  proc.time(); C <- A %*% B;
> proc.time() - ptm
   user  system elapsed
116.389   0.166 116.503
```

Speedup



Speedup



Tentative Syllabus

1. Intro and `gputools`
2. A codeless intro to GPU parallelism
3. Intro to CUDA C
4. CUDA C: K-means and MCMC
5. CUDA C: Shared memory and performance measurement
6. CUDA C: Race conditions, atomics, and warps
7. CUBLAS and CULA: linear algebra libraries for CUDA C
8. CURAND: a GPU-enabled library for fast random number generation
9. Thrust: the GPU analog of the C++ Standard Template Library
10. Intro to Python: preparation for PyCUDA
11. PyCUDA: a Python module for GPU computing

Outline

GPUs, parallelism, and why we care

CUDA and our CUDA systems

GPU computing with R

Resources

1. J. Sanders and E. Kandrot. CUDA by Example. Addison-Wesley, 2010.
2. D. Kirk, W.H. Wen-mei, and W. Hwu. *Programming massively parallel processors: a hands-on approach.* Morgan Kaufmann, 2010.
3. A. Lee, C. Yau, M.B. Giles, A. Doucet, and C.C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced monte carlo methods. *Journal of Computational and Graphical Statistics*, 19(4): 769-789, 2010.
4. M.A. Suchard, Q. Wang, C. Chan, J. Frelinger, A. Cron, and M. West. Understanding gpu programming for statistical computation: Studies in massively parallel mixtures. *Journal of Computational and Graphical Statistics.* 19(2): 419-438, 2010.

That's all for today.

- Series materials are available at http://will-landau.com/gpu.