gputools: an R package for GPU computing

Will Landau, Prof. Jarad Niemi

Outline

- Contents of gputools
- Usage
- Performance
- Other R packages for the GPU

CONTENTS OF gputools

A handful of selected R functions implemented with CUDA C for use on a GPU:

- 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 exact usage?
gpuLm()	lm()	yes
gpuLsfit()	lsfit()	yes
gpuGlm()	glm()	yes
gpuGlm.fit()	glm.fit()	yes

- Hypothesis testing:

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

- Other routines:

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

getAucEstimate()

Estimates the area under a receiver operating characteristic (ROC) curve.

Used to evaluate the performance of a hypothesis test in a multiple testing scenario.

Reference:

Hand, David J. and Till, Robert J. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. Machine Learning. 45, 171-186.

gpuHclust()

Performs hierarchical clustering on a set of points.

The distances among the points must be given in an bject of class "dist".

gpuDistClust()

Given a set of points, computes all pairwise distances and then performs hierarchical clustering on the points.

Both steps are done on the GPU.

gpuFastICA

Performs Independent Component Analysis (ICA) and Projection Pursuit.

ICA, like principle component analysis, is a linear decomposition of a design matrix.

The authors of fastICA claim that, unlike PCA, ICA “unmixes” the underlying sources of variability in the data by assuming a non-Gaussian structure.

This function is exactly like the ICA implementation in the `fastICA` package except that the `gputools` version uses `gpuSvd()` instead of `svd()`.

References:

A. Hyvarinen and E. Oja (2000) Independent Component Analysis: Algorithms and Applications, Neural Networks, 13(4-5):411-430. http://www.cis.hut.fi/aapo/

A. Hyvarinen. Independent Component Analysis: Recent Advances. Philosophical Transactions of the Royal Society A, in press. http://www.cs.helsinki.fi/u/ahyvarin/papers/PTRSA12.pdf.

gpuGranger()

Performs the Granger Causality Test, which tests how well one time series forecasts another.

Reference:

Hacker R.S. and Hatemi-J A. (2006) "Tests for causality between integrated variables using asymptotic and bootstrap distributions: theory and application", Applied Economics, Vol. 38(13), pp. 1489-1500.

gpuMi()

Estimates the mutual information for pairs of vectors using a B spline approach.

Reference:

Carten O. Daub, Ralf Steuer, Joachim Selbig, and Sebastian Kloska. 2004. Estimating mutual information using B-spline functions - an improved similarity measure for analysing gene expression data. BMC Bioinformatics. 5:118. Available from http://www.biomedcentral.com/1471-2105/5/118

gpuSvmPredict()

Classifies points in a data set using a support vector machine.

In machine learning, support vector machine (SVM) is a learning model used for classification and regression.

Reference:

Carpenter, Austin. cuSVM: a cuda implementation of support vector classification and regression.
http://patternsonascreen.net/cuSVM.html

gpuSvmTrain()

Trains a support vector machine.

Reference:

Carpenter, Austin. cuSVM: a cuda implementation of support vector classification and regression. http://patternsonascreen.net/cuSVM.html

USAGE

- gputools is already installed on impact1.stat.iastate.edu, ready to load with `library(gputools)` in R.
- For other GPU systems, download gputools from CRAN with a simple `install.packages("gputools")` in R.
  - WARNING: installation will fail on non-GPU systems since the CUDA C compiler doesn't exist
- Documentation:
  - http://brainarray.mbni.med.umich.edu/Brainarray/Rgpgpu/
  - http://cran.r-project.org/web/packages/gputools/index.html
  - http://cran.r-project.org/web/packages/gputools/gputools.pdf
- Requirements:
  - R (>= version 2.8.0)
  - Nvidia's CUDA toolkit (>= version 2.3)

MANAGING YOUR DEVICES: `chooseGpu()` AND `getGpuId()`

Impact1 has four GPUs, each with a unique index from 0 to 3. To see this for yourself, log into impact1 and run the following:

```
[landau@impact1 ~]$ cd /usr/local/NVIDIA_GPU_Computing_SDK/C/bin/linux/release
[landau@impact1 release]$ ./deviceQuery
```

Here are some pieces of the (quite verbose) output of `./deviceQuery`:

```
[deviceQuery] starting...

./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Found 4 CUDA Capable device(s)

Device 0: "Tesla M2070"
  CUDA Driver Version / Runtime Version          4.1 / 4.1
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (14) Multiprocessors x (32) CUDA Cores/MP:     448 CUDA Cores
  GPU Clock Speed:                               1.15 GHz
  Memory Clock rate:                             1566.00 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
```

```
Device 1: "Tesla M2070"
  CUDA Driver Version / Runtime Version          4.1 / 4.1
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (14) Multiprocessors x (32) CUDA Cores/MP:     448 CUDA Cores
  GPU Clock Speed:                               1.15 GHz
  Memory Clock rate:                             1566.00 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
```

```
Device 2: "Tesla M2070"
  CUDA Driver Version / Runtime Version          4.1 / 4.1
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (14) Multiprocessors x (32) CUDA Cores/MP:     448 CUDA Cores
  GPU Clock Speed:                               1.15 GHz
  Memory Clock rate:                             1566.00 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
```

```
Device 3: "Tesla M2070"
  CUDA Driver Version / Runtime Version          4.1 / 4.1
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (14) Multiprocessors x (32) CUDA Cores/MP:     448 CUDA Cores
  GPU Clock Speed:                               1.15 GHz
  Memory Clock rate:                             1566.00 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
```

Things to note:

- Device 3 is a GPU
- “Tesla M2070” is the name of the model of the GPU.
- Device 3 contains multiple cores, or “sub-processors”. From the output, it has 448 CUDA-capable cores.

nvidia-smi: CHECK GPU USAGE BEFORE `chooseGpu()`

```
[landau@impact1 ~]$ nvidia-smi
Thu Sep 13 09:37:05 2012
+------------------------------------------------------+
| NVIDIA-SMI 2.290.10   Driver Version: 290.10         |
|-------------------------------+----------------------+----------------------+
| Nb.  Name                     | Bus Id        Disp.  | Volatile ECC SB / DB |
| Fan   Temp   Power Usage /Cap | Memory Usage         | GPU Util. Compute M. |
|===============================+======================+======================|
| 0.  Tesla M2070               | 0000:0B:00.0  Off    |         0          0 |
|  N/A    N/A  P8    Off /  Off |   0%    9MB / 5375MB |    0%     Default    |
|-------------------------------+----------------------+----------------------|
| 1.  Tesla M2070               | 0000:0C:00.0  Off    |         0          0 |
|  N/A    N/A  P8    Off /  Off |   0%    9MB / 5375MB |    0%     Default    |
|-------------------------------+----------------------+----------------------|
| 2.  Tesla M2070               | 0000:0D:00.0  Off    |         0          0 |
|  N/A    N/A  P8    Off /  Off |   0%    9MB / 5375MB |    0%     Default    |
|-------------------------------+----------------------+----------------------|
| 3.  Tesla M2070               | 0000:0E:00.0  Off    |         0          0 |
|  N/A    N/A  P8    Off /  Off |   0%    9MB / 5375MB |    0%     Default    |
|-------------------------------+----------------------+----------------------|
| Compute processes:                                               GPU Memory |
|  GPU  PID     Process name                                       Usage      |
|=============================================================================|
|  No running compute processes found                                         |
+-----------------------------------------------------------------------------+
[landau@impact1 ~]$
```

```
[landau@impact1 ~]$ nvidia-smi -i 0 -q

==============NVSMI LOG==============

Timestamp                       : Thu Sep 13 09:37:54 2012

Driver Version                  : 290.10

Attached GPUs                   : 4

GPU 0000:0B:00.0
    Product Name                : Tesla M2070
    Display Mode                : Disabled
    Persistence Mode            : Disabled
    Driver Model
        Current                 : N/A
        Pending                 : N/A
    Serial Number               : 0323111076435
    GPU UUID                    : GPU-63911bd22733e078-94bd6965-7a0cbc1f-29f7d33f-489fcc8d5229600c5a45b88a
    VBIOS Version               : 70.00.3E.00.03
    Inforom Version
        OEM Object              : 1.0
        ECC Object              : 1.0
        Power Management Object : 1.0
```

```
    PCI
        Bus                           : 0x0B
        Device                        : 0x00
        Domain                        : 0x0000
        Device Id                     : 0x06D210DE
        Bus Id                        : 0000:0B:00.0
        Sub System Id                 : 0x083010DE
        GPU Link Info
            PCIe Generation
                Max                   : 2
                Current               : 2
            Link Width
                Max                   : 16x
                Current               : 16x
    Fan Speed                         : N/A
    Performance State                 : P8
    Memory Usage
        Total                         : 5375 MB
        Used                          : 9 MB
        Free                          : 5365 MB
    Compute Mode                      : Default
    Utilization
        Gpu                           : 0 %
        Memory                        : 0 %
    Ecc Mode
        Current                       : Enabled
        Pending                       : Enabled
```

```
    ECC Errors
        Volatile
            Single Bit
                Device Memory   : 0
                Register File   : 0
                L1 Cache        : 0
                L2 Cache        : 0
                Total           : 0
            Double Bit
                Device Memory   : 0
                Register File   : 0
                L1 Cache        : 0
                L2 Cache        : 0
                Total           : 0
        Aggregate
            Single Bit
                Device Memory   : N/A
                Register File   : N/A
                L1 Cache        : N/A
                L2 Cache        : N/A
                Total           : 0
            Double Bit
                Device Memory   : N/A
                Register File   : N/A
                L1 Cache        : N/A
                L2 Cache        : N/A
                Total           : 0
```

```
    Temperature
        Gpu                           : N/A
    Power Readings
        Power Management              : N/A
        Power Draw                    : N/A
        Power Limit                   : N/A
    Clocks
        Graphics                      : 270 MHz
        SM                            : 540 MHz
        Memory                        : 1566 MHz
    Max Clocks
        Graphics                      : 573 MHz
        SM                            : 1147 MHz
        Memory                        : 1566 MHz
    Compute Processes                 : None
[landau@impact1 ~]$
```

EXAMPLE: MATRIX MULTIPLICATION

Now, suppose I want to do a giant matrix multiplication on Device 3. I'm automatically set to Device 0:

```
> getGpuId()
[1] 0
```

So I change to Device 3:

```
> chooseGpu(3)
[[1]]
[1] 3
```

and then if I want, I can verify the change:

```
> getGpuId()
[1] 3
```

Now, I define the matrices that I want to multiply on Device 3:

```
> A <- matrix(runif(1e+7), nrow = 1e+4)
> B <- matrix(runif(1e+7), ncol = 1e+4)
```

Then, I tell the device to multiply `A` and `B` using the GPU hardware:

```
> ptm <- proc.time(); C <- gpuMatMult(A, B); proc.time() - ptm
   user  system elapsed
  2.959   2.190   5.159
```

Compare the run time to that of the analogous CPU run on impact1:

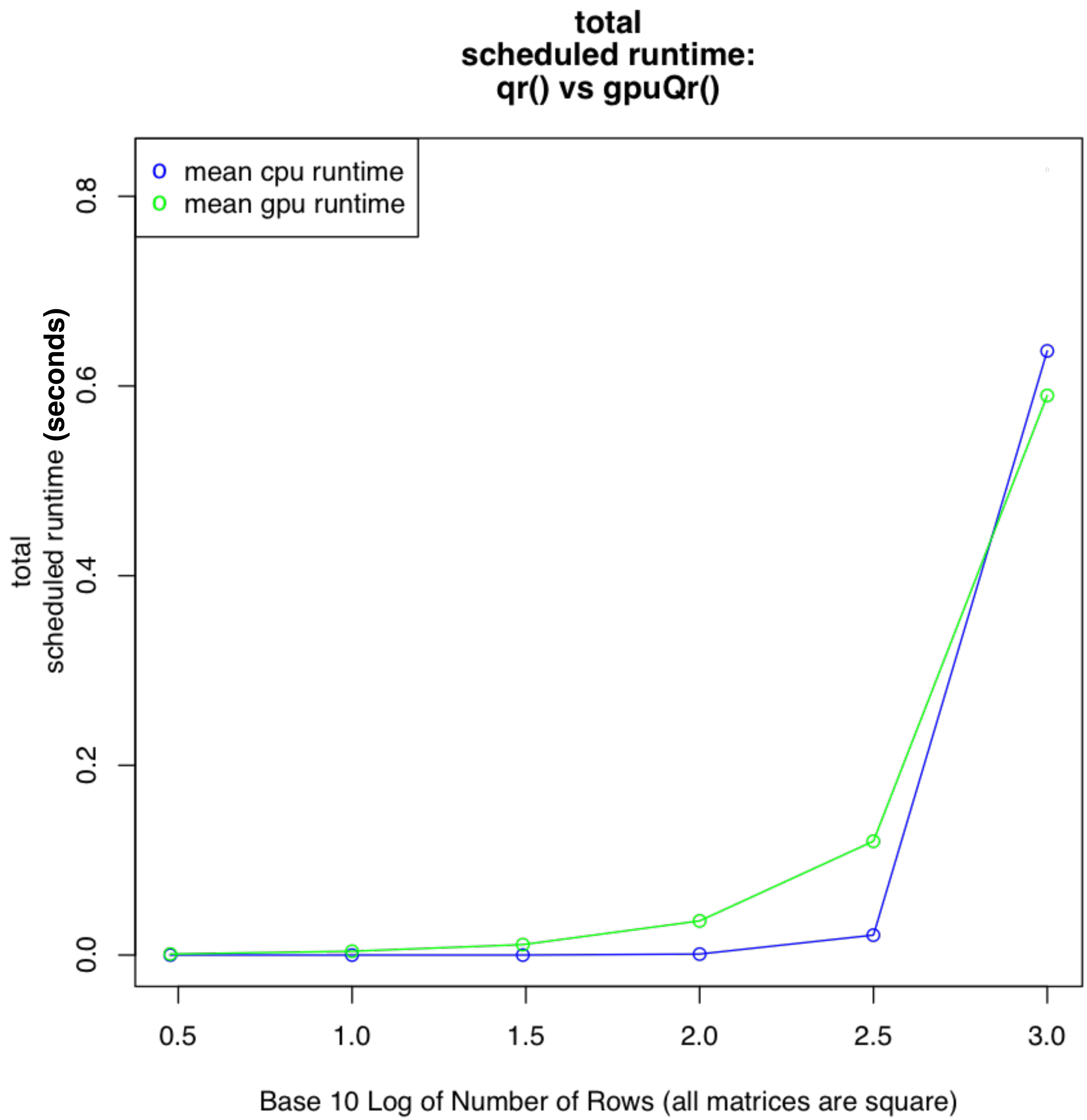
```
> ptm <- proc.time(); D <- A %*% B; proc.time() - ptm
   user  system elapsed
116.389   0.166 116.503
```

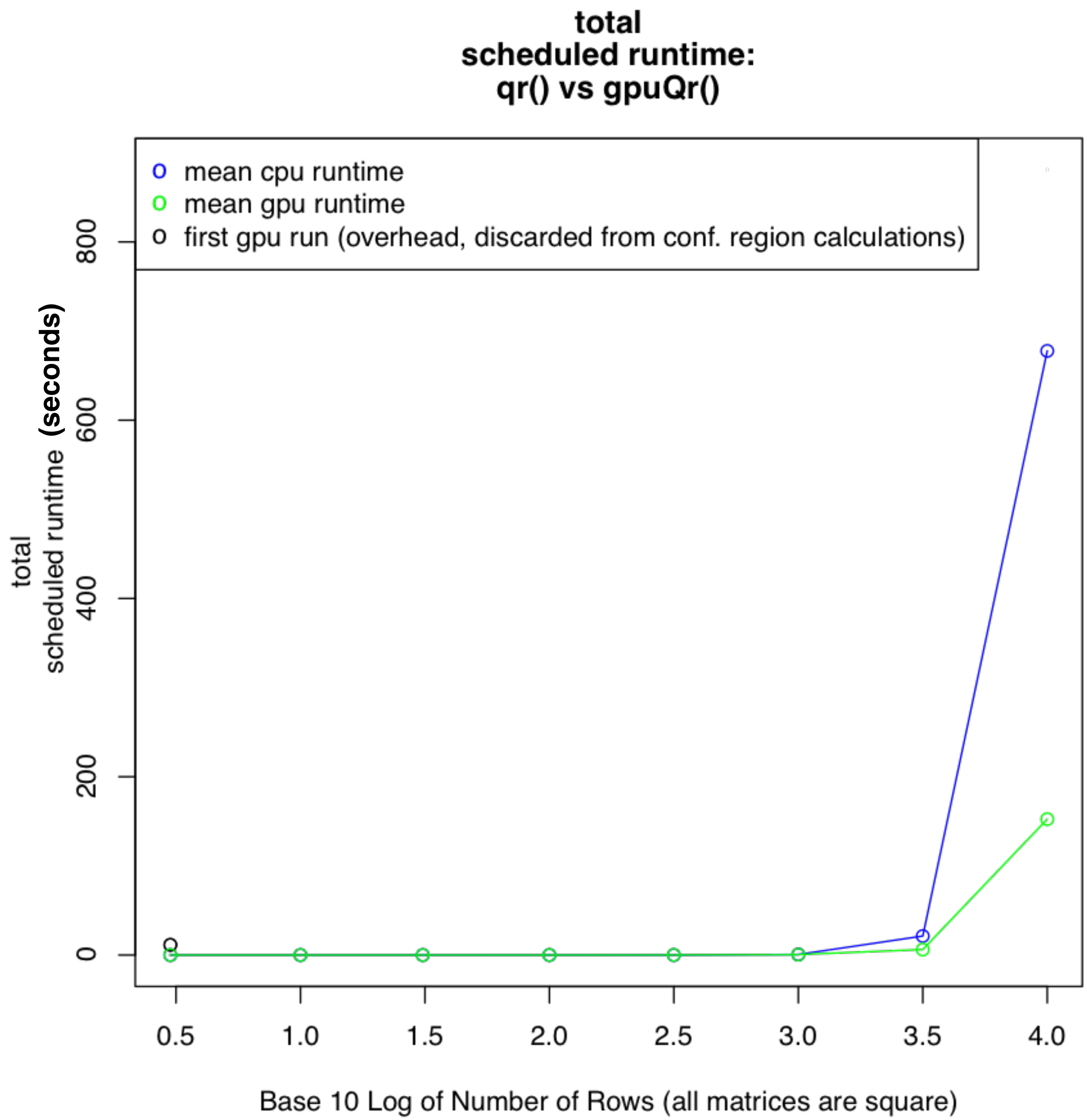
PERFORMANCE

A COMPARISON OF gpuQr() AND qr()

The R script, `gpuQr.r`, compares the performance of `gpuQr(arg)` and `qr(arg)` for square matrices `arg` of varying sizes.

See the results on the next few slides.

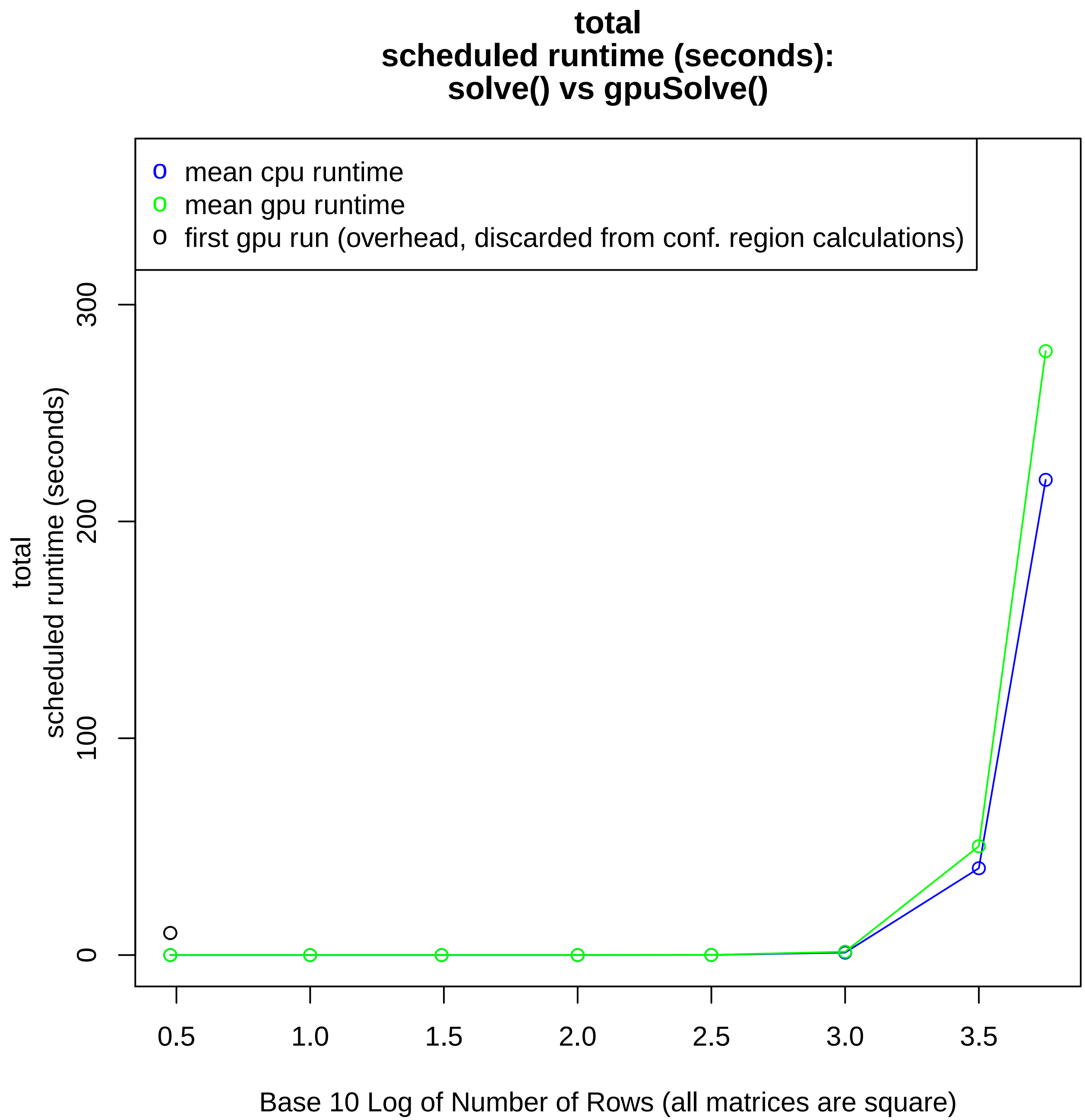

total
scheduled runtime:
qr() vs gpuQr()
mean cpu runtime
mean gpu runtime
total
scheduled runtime (seconds)
0.0
0.2
0.4
0.6
0.8
0.5
1.0
1.5
2.0
2.5
3.0
Base 10 Log of Number of Rows (all matrices are square)


total
scheduled runtime:
qr() vs gpuQr()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime (seconds)
0
200
400
600
800
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Base 10 Log of Number of Rows (all matrices are square)

A COMPARISON OF gpuSolve() AND solve()

The R script, `gpuSolve.r`, compares the performance of `gpuSolve(arg)` and `solve(arg)` for square matrices `arg` of varying sizes.

See the results on the next slide.


total
scheduled runtime (seconds):
solve() vs gpuSolve()
o mean cpu runtime
o mean gpu runtime
o first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime (seconds)
0
100
200
300
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Base 10 Log of Number of Rows (all matrices are square)

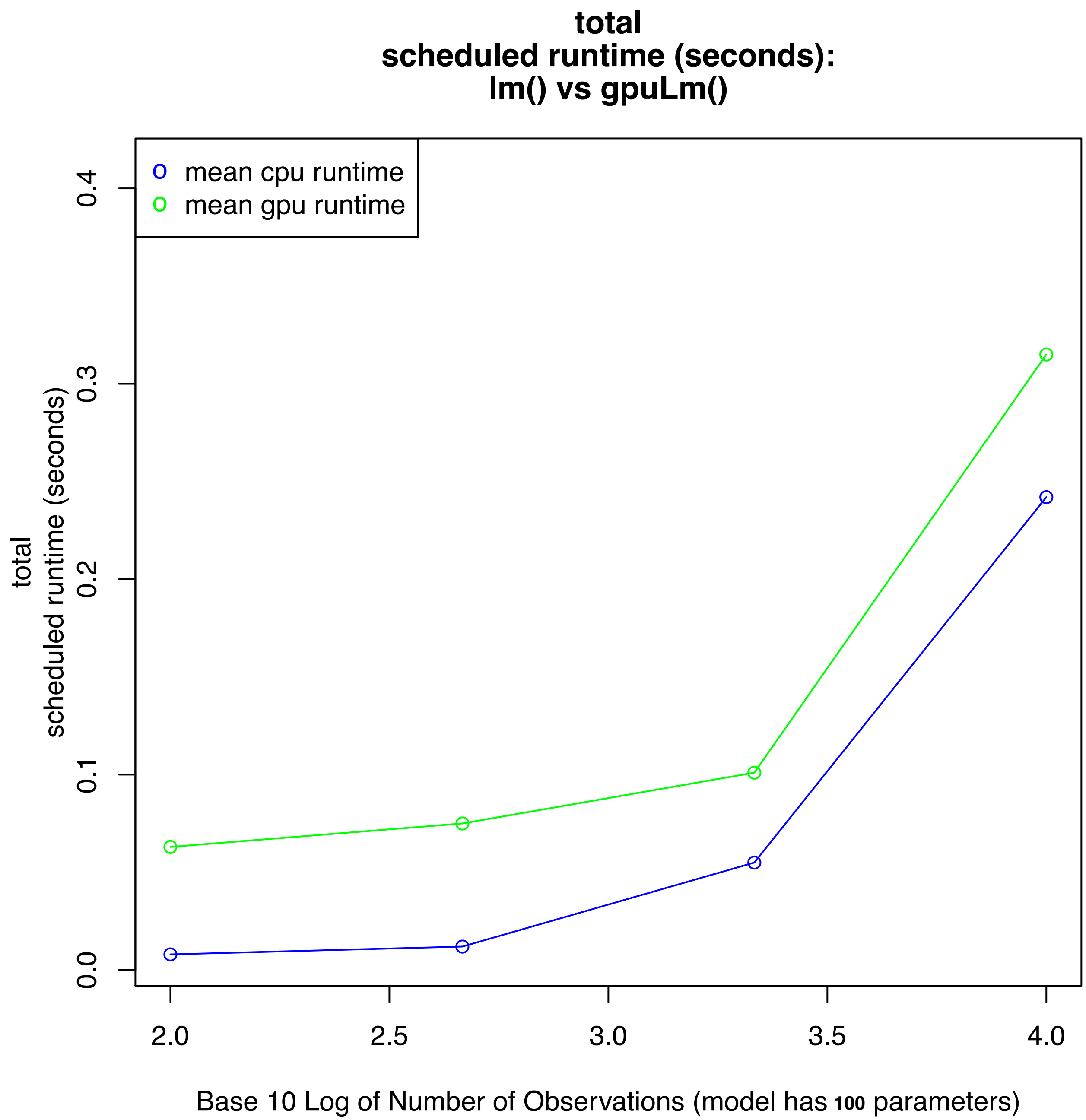
A COMPARISON OF gpuLm() AND lm()

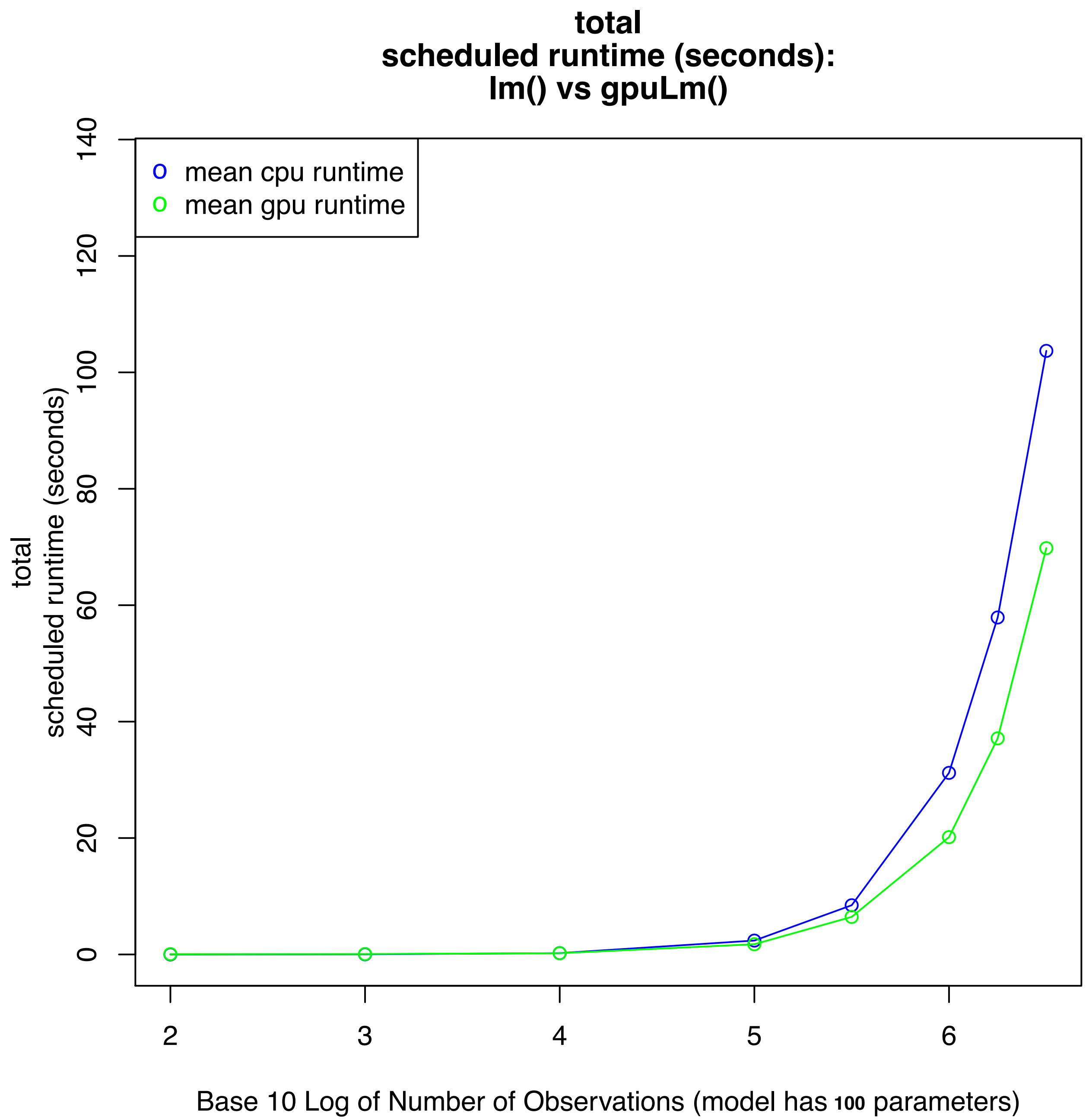
The R script, `gpuLm.r`, compares the performance of `gpuLm(y ~ X)` and `lm(y ~ X)`, where:

- `y` is a random vector of observations.
- `X` is a random design matrix with `length(y)` rows and 100 columns.

The script times each function with varying `length(y)` and `nrow(X)`.

See the results on the next slides.


total
scheduled runtime (seconds):
lm() vs gpuLm()
mean cpu runtime
mean gpu runtime
total
scheduled runtime (seconds)
0.0
0.1
0.2
0.3
0.4
2.0
2.5
3.0
3.5
4.0
Base 10 Log of Number of Observations (model has 100 parameters)



total
scheduled runtime (seconds):
lm() vs gpuLm()
mean cpu runtime
mean gpu runtime
total
scheduled runtime (seconds)
0
20
40
60
80
100
120
140
2
3
4
5
6
Base 10 Log of Number of Observations (model has 100 parameters)

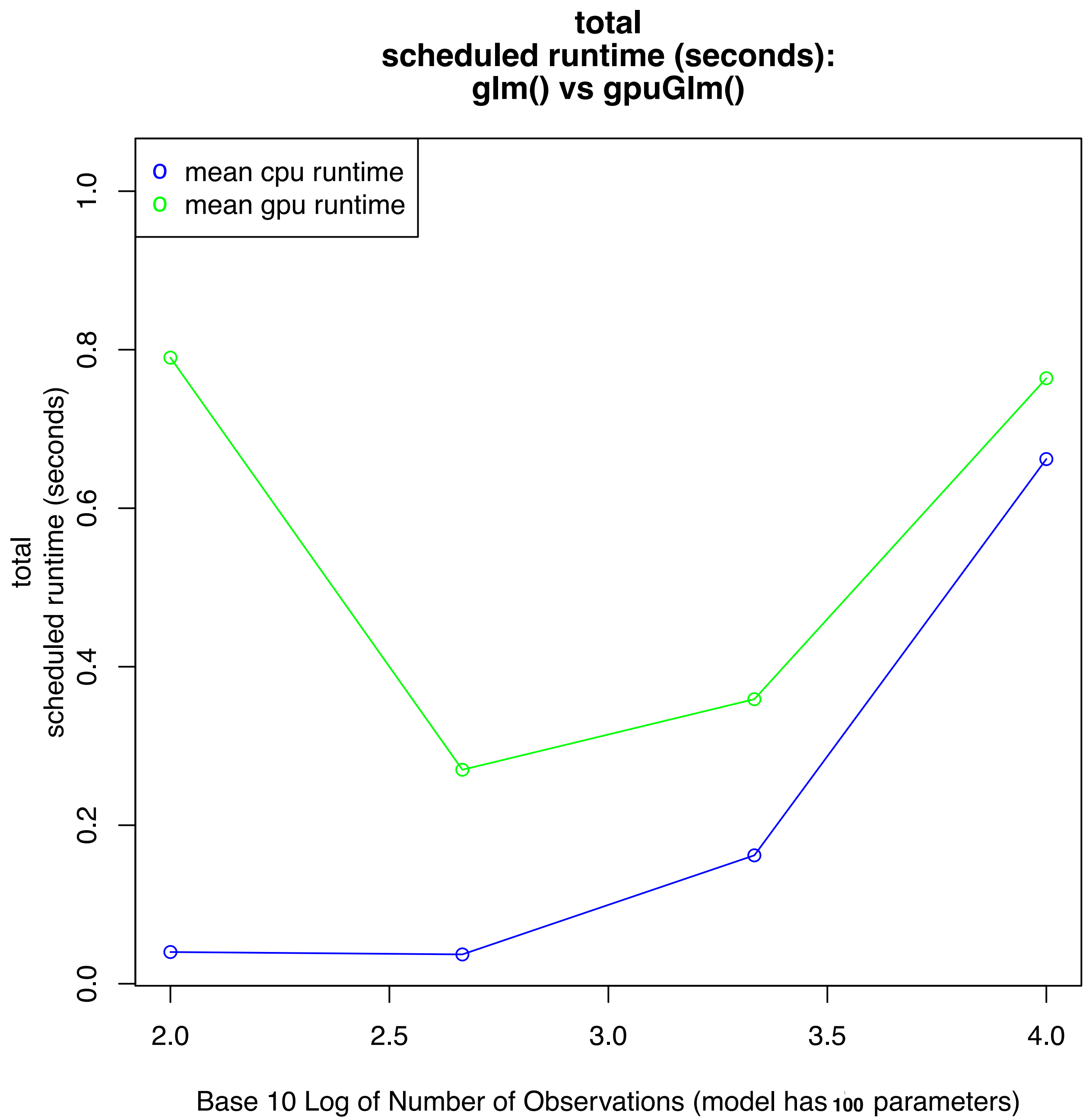
A COMPARISON OF gpuGlm() AND glm()

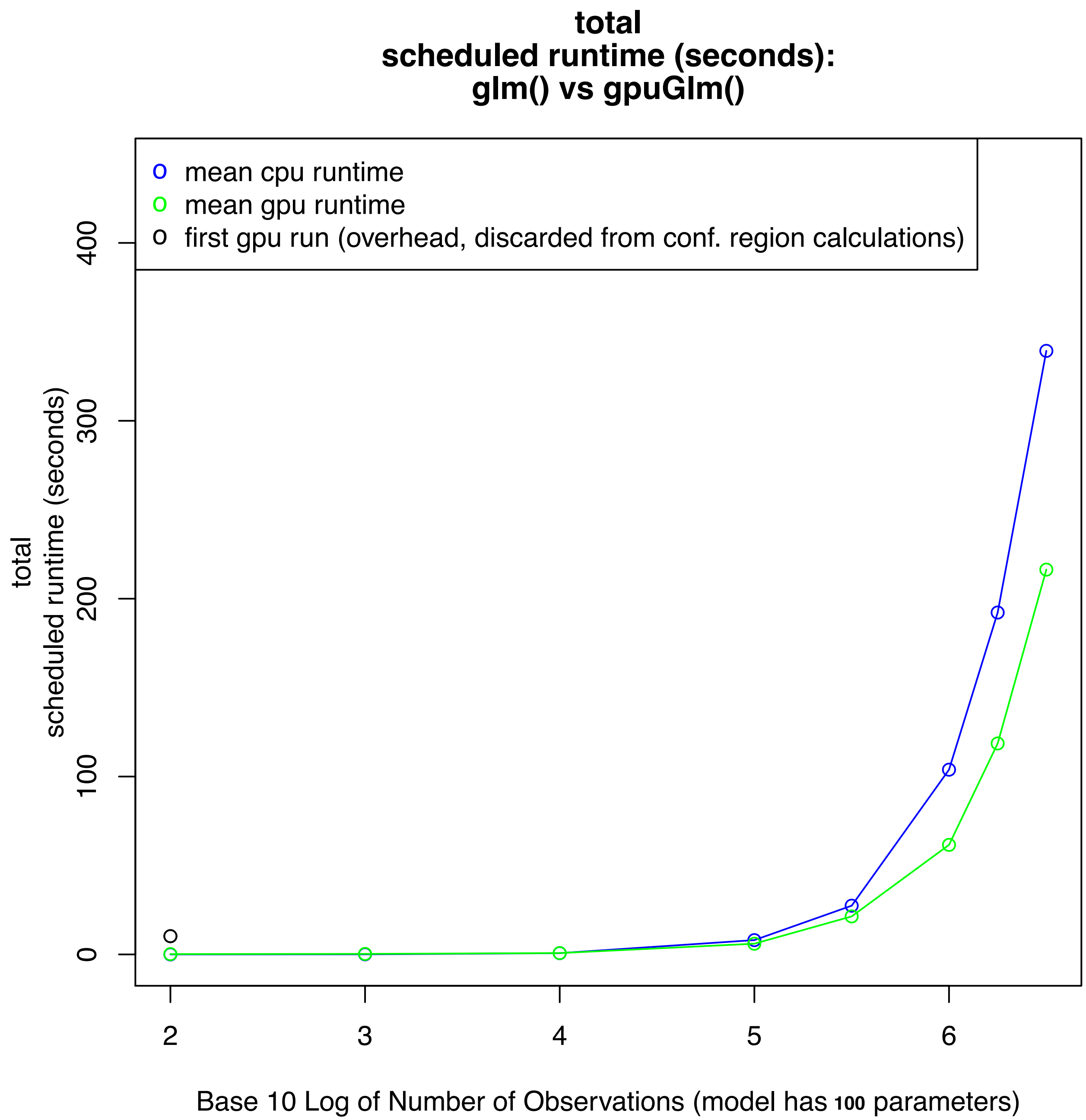
The R script, `gpuGlm.r`, compares the performance of `gpuGlm(y ~ X, family = poisson())` and `glm(y ~ X, family = poisson())`, where:

- `y` is a random vector of observations.
- `X` is a random design matrix with `length(y)` rows and 100 columns.

The script times each function with varying `length(y)` and `nrow(X)`.

See the results on the next slides.


total
scheduled runtime (seconds):
glm() vs gpuGlm()
mean cpu runtime
mean gpu runtime
total
scheduled runtime (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
2.0
2.5
3.0
3.5
4.0
Base 10 Log of Number of Observations (model has 100 parameters)


total
scheduled runtime (seconds):
glm() vs gpuGlm()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime (seconds)
0
100
200
300
400
2
3
4
5
6
Base 10 Log of Number of Observations (model has 100 parameters)

CLAIMS FROM THE AUTHORS OF `gputools`

(All of the following is from http://brainarray.mbni.med.umich.edu/Brainarray/Rgpgpu/.)

"Tested on a subset of GSE6306, non-GPU enabled fastICA took over four hours while gpuFastICA took just 80 seconds!"

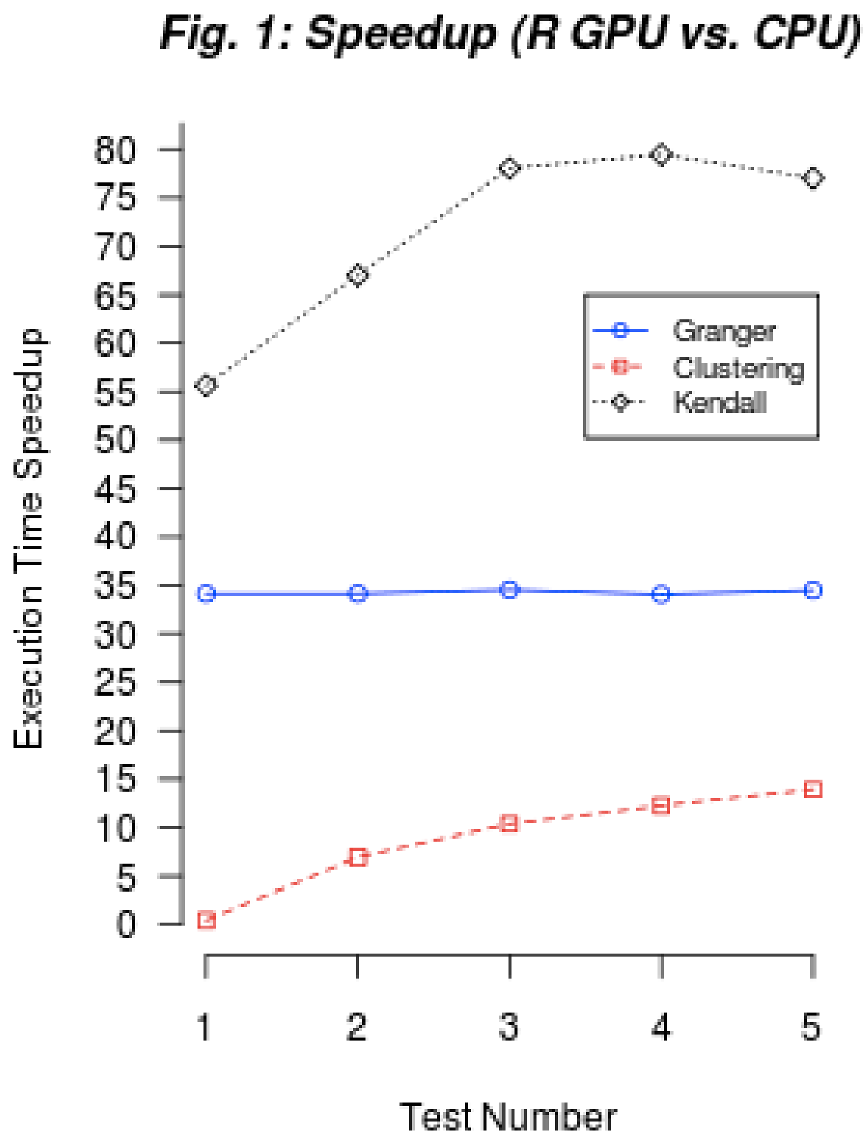


Fig. 1: Speedup (R GPU vs. CPU)
Execution Time Speedup
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
Test Number
1
2
3
4
5
Granger
Clustering
Kendall

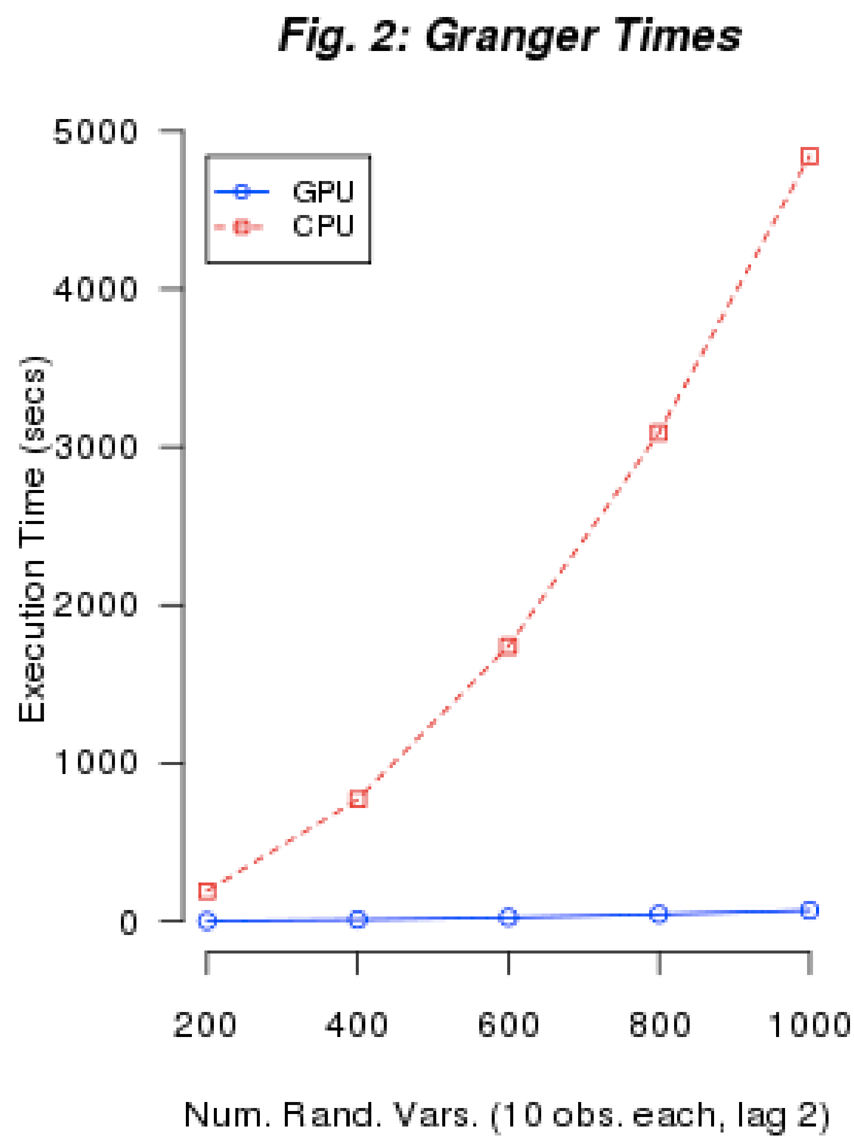
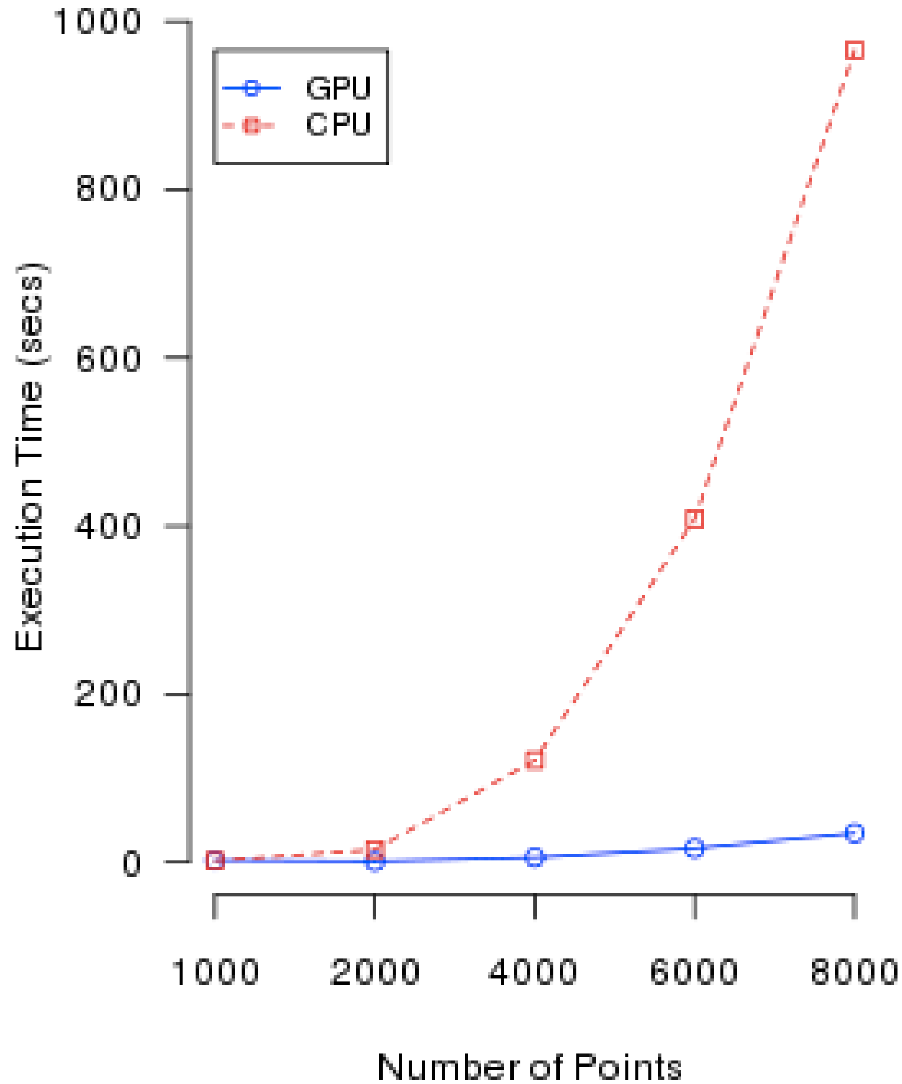


Fig. 2: Granger Times
GPU
CPU
5000
4000
3000
2000
1000
0
Execution Time (secs)
200
400
600
800
1000
Num. Rand. Vars. (10 obs. each, lag 2)

Fig. 3: Cluster Times



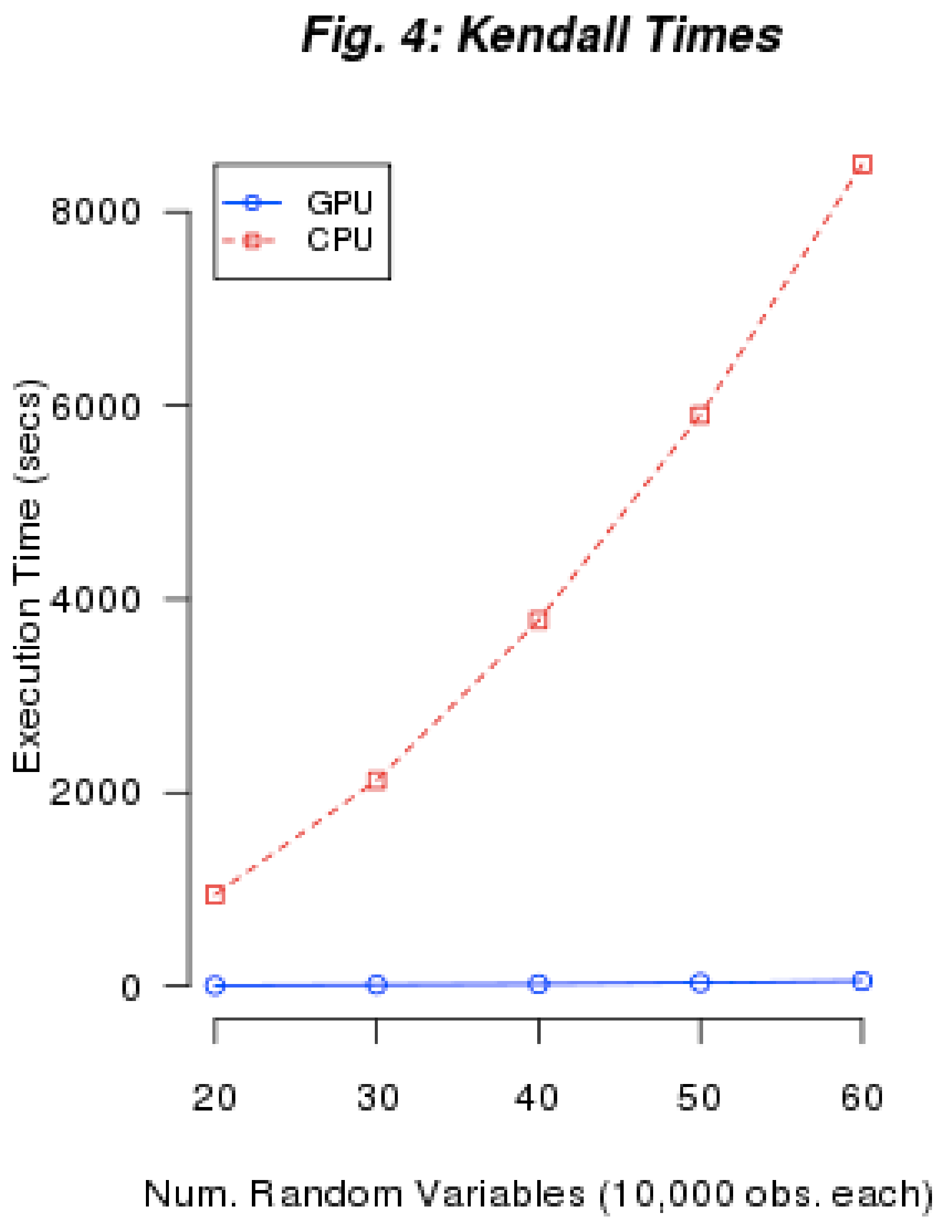


Fig. 4: Kendall Times
GPU
CPU
Execution Time (secs)
8000
6000
4000
2000
0
20
30
40
50
60
Num. Random Variables (10,000 obs. each)

OTHER R PACKAGES FOR THE GPU

- 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.

Outline

- Contents of gputools
- Usage
- Performance
- Other R packages for the GPU

GPU SERIES MATERIALS

These slides, a tentative syllabus for the whole series, and code are available at:

https://github.com/wlandau/gpu.

After logging into you home directory on impact1, type:

```
git clone https://github.com/wlandau/gpu
```

into the command line to download all the materials.

REFERENCES

Josh Buckner, Mark Seligman, Justin Wilson. "R+GPU". http://brainarray.mbni.med.umich.edu/Brainarray/Rgpgpu/#introduction.

Carten O. Daub, Ralf Steuer, Joachim Selbig, and Sebastian Kloska. 2004. Estimating mutual information using B-spline functions - an improved similarity measure for analysing gene expression data. BMC Bioinformatics. 5:118. Available from http://www.biomedcentral.com/1471-2105/5/118

Carpenter, Austin. cuSVM: a cuda implementation of support vector classification and regression. http://patternsonascreen.net/cuSVM.html

Dirk Eddelbuettel. "Package gcbd". http://cran.r-project.org/web/packages/gcbd/gcbd.pdf.

Hacker R.S. and Hatemi-J A. (2006) "Tests for causality between integrated variables using asymptotic and bootstrap distributions: theory and application", Applied Economics, Vol. 38(13), pp. 1489-1500.

Hand, David J. and Till, Robert J. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. Machine Learning. 45, 171-186.

A. Hyvarinen and E. Oja (2000) Independent Component Analysis: Algorithms and Applications, Neural Networks, 13(4-5):411-430. http://www.cis.hut.fi/aapo/

A. Hyvarinen. Independent Component Analysis: Recent Advances. Philosophical Transactions of the Royal Society A, in press.
http://www.cs.helsinki.fi/u/ahyvarin/papers/PTRSA12.pdf.

Mark Seligman, Chris Fraley. “Package WideLM”.
http://cran.r-project.org/web/packages/WideLM/WideLM.pdf.

Brian J Smith. “Package cudaBayesreg”.
http://cran.r-project.org/web/packages/cudaBayesreg/cudaBayesreg.pdf.