gputools: an R package for GPU computing

Will Landau, Prof. Jarad Niemi

GETTING UP AND RUNNING WITH gputools

- 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)

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

- Other useful algorithms:

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

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

Background: 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:

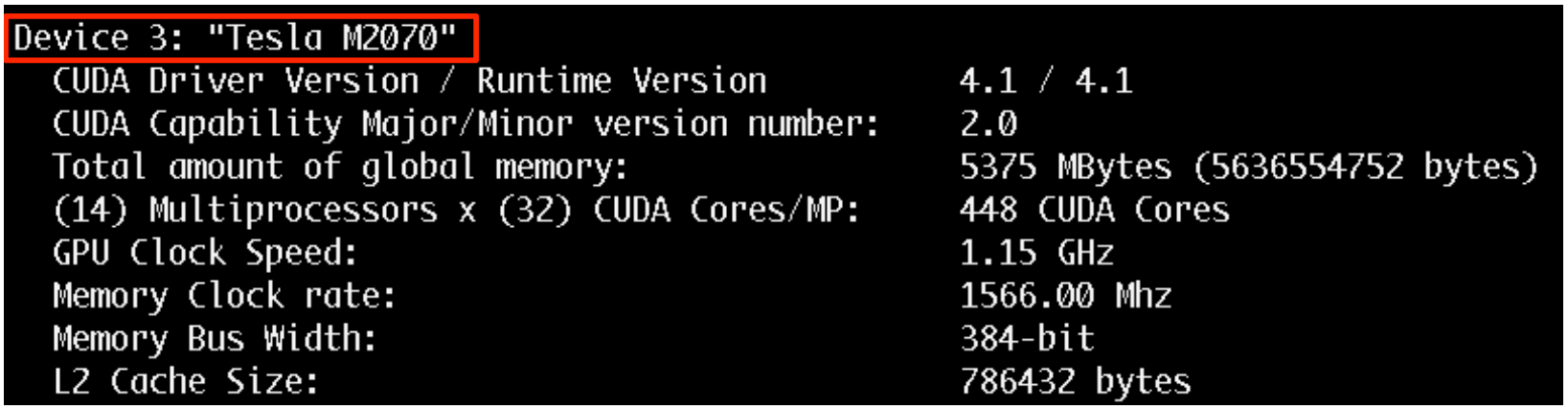
```
[landau@impact1 ~]$ cd /usr/local/NVIDIA_GPU_Computing_SDK/C/bin/linux/release
[landau@impact1 release]$ ./deviceQuery
```

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
```



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.

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:

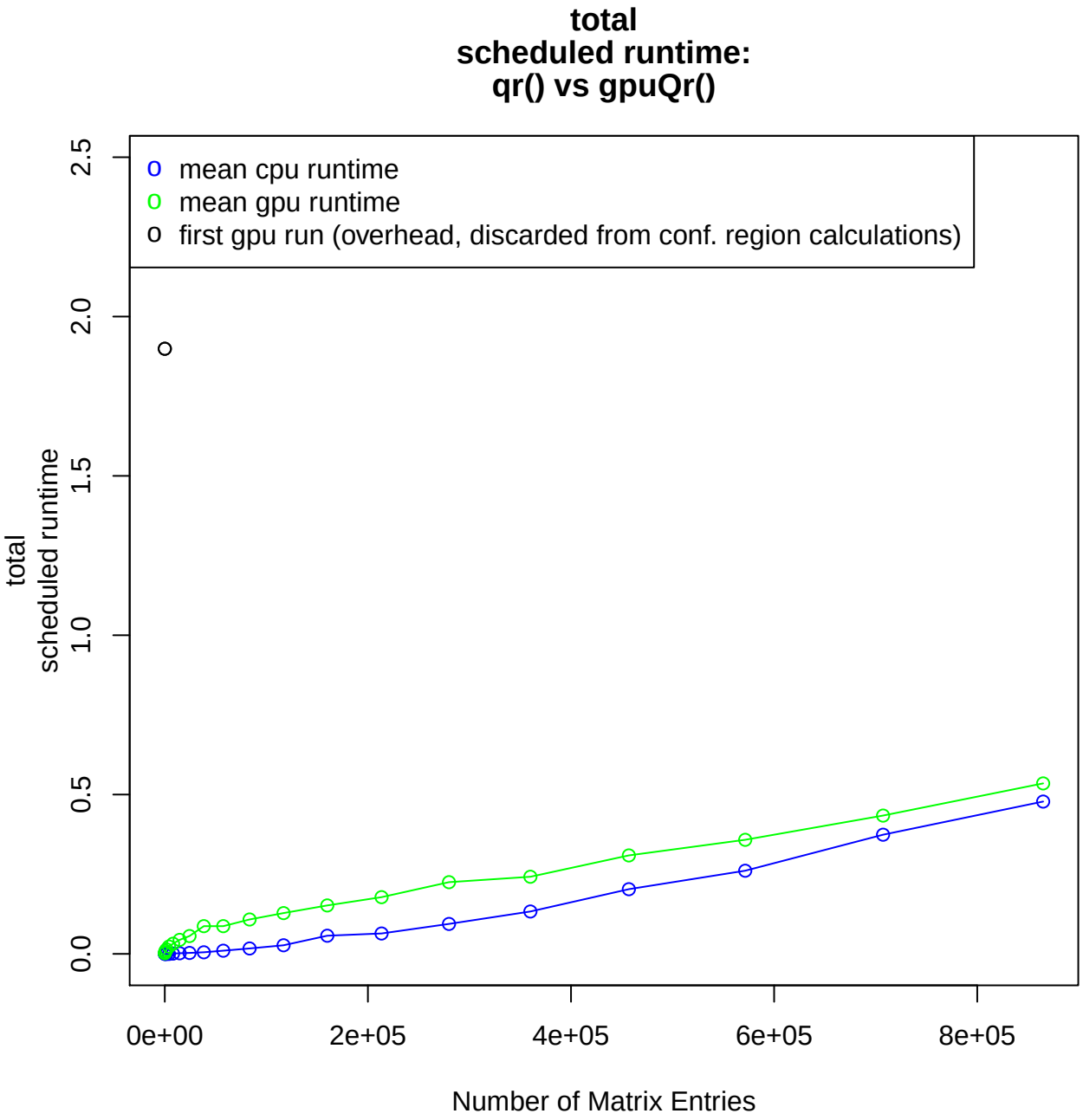
```
> ptm <- proc.time(); D <- A %*% B; proc.time() - ptm
   user  system elapsed
116.389   0.166 116.503
```

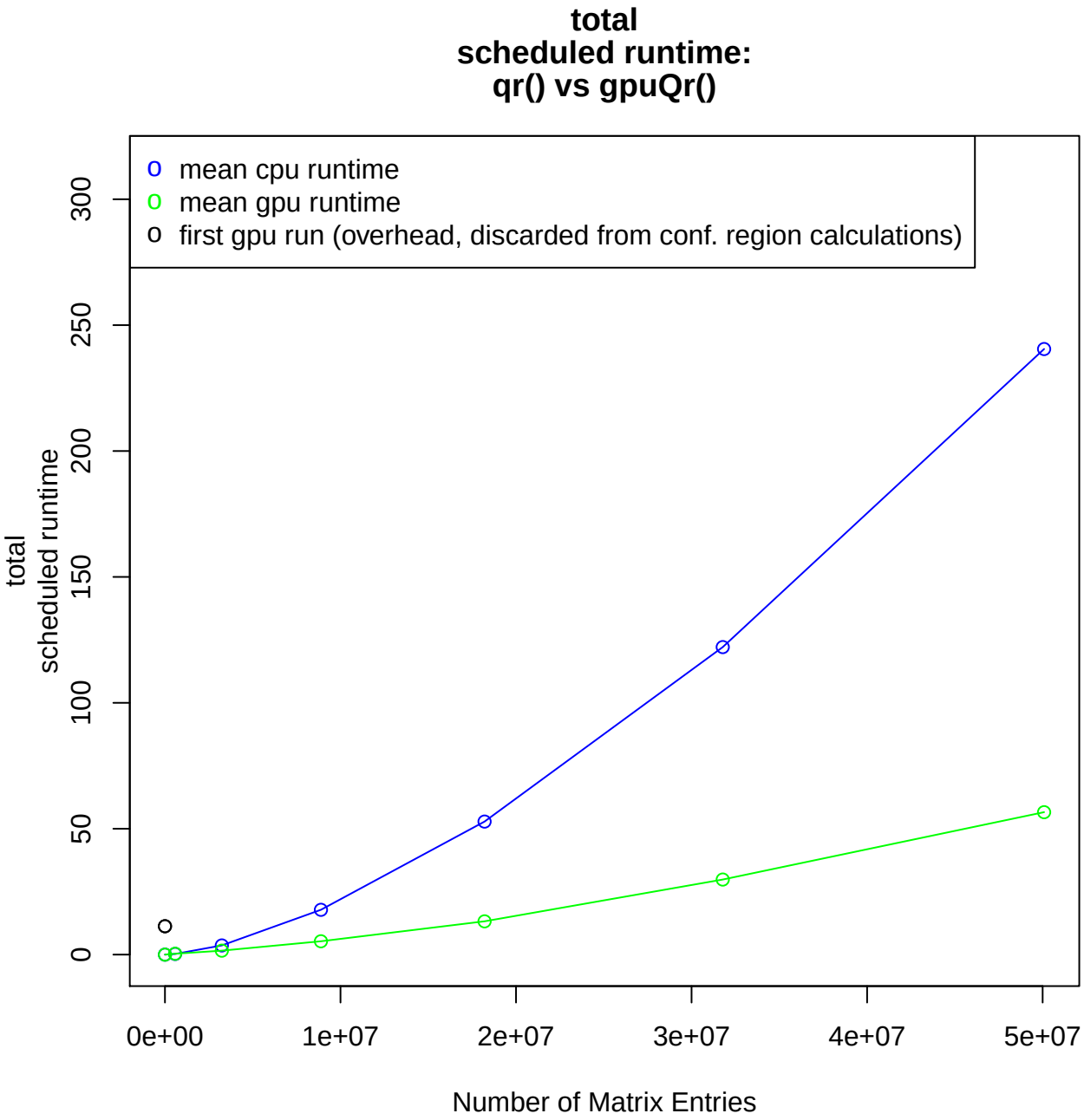
A COMPARISON OF gpuQr() AND qr()

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.

NOTE: ALL TIMES ARE IN SECONDS.

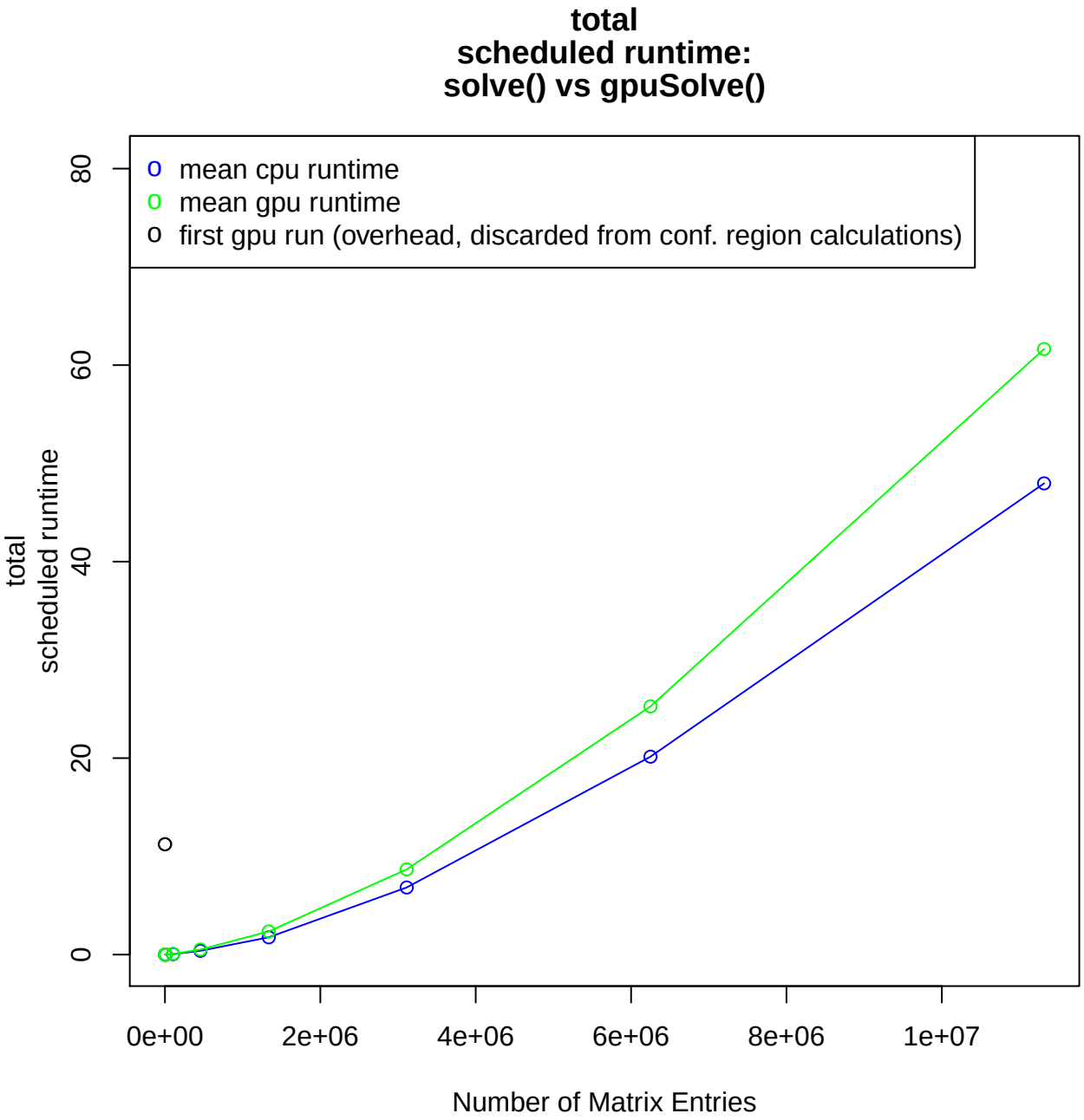

total
scheduled runtime:
qr() vs gpuQr()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0.0
0.5
1.0
1.5
2.0
2.5
0e+00
2e+05
4e+05
6e+05
8e+05
Number of Matrix Entries


total
scheduled runtime:
qr() vs gpuQr()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
50
100
150
200
250
300
0e+00
1e+07
2e+07
3e+07
4e+07
5e+07
Number of Matrix Entries

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.

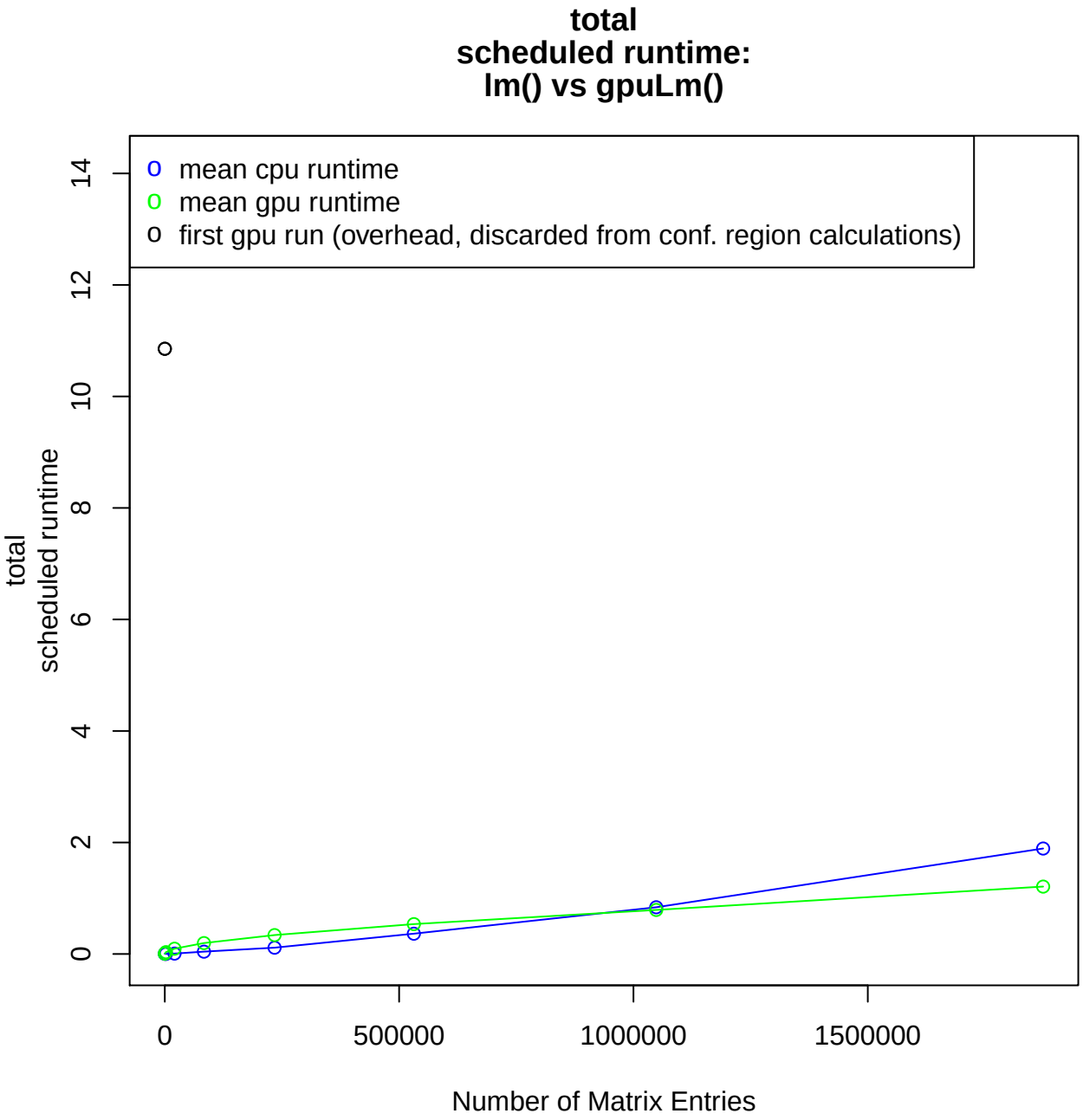
See the results on the next few slides.

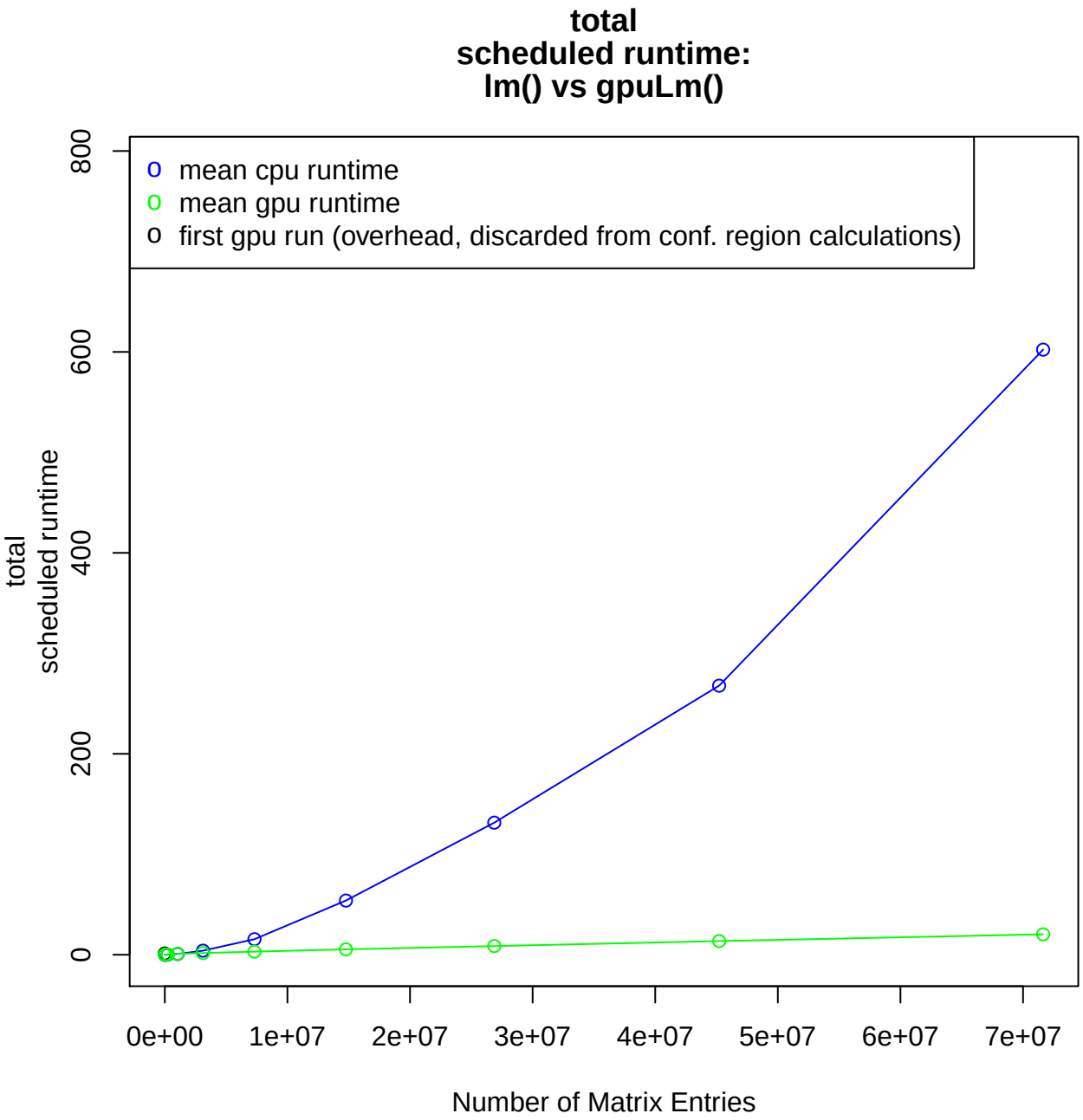

total
scheduled runtime:
solve() vs gpuSolve()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
20
40
60
80
0e+00
2e+06
4e+06
6e+06
8e+06
1e+07
Number of Matrix Entries

A COMPARISON OF gpuLm() AND lm()

The R script, `gpuLm.r`, compares the performance of `gpuLm(y ~ x)` and `lm(y ~ x)` for matrices `y` and `x` of varying sizes.

See the results on the next slide.

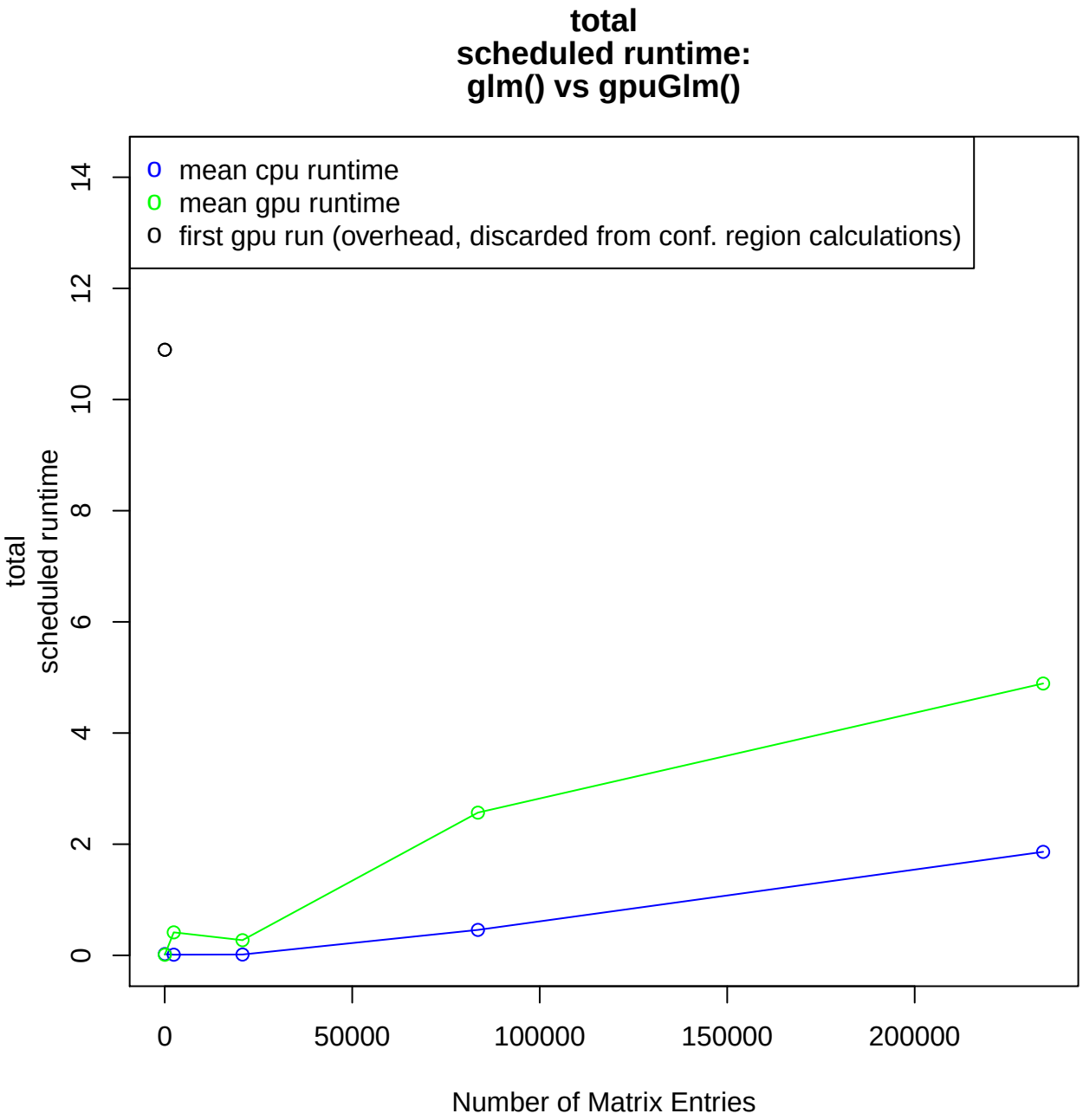

total
scheduled runtime:
lm() vs gpuLm()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
2
4
6
8
10
12
14
0
500000
1000000
1500000
Number of Matrix Entries

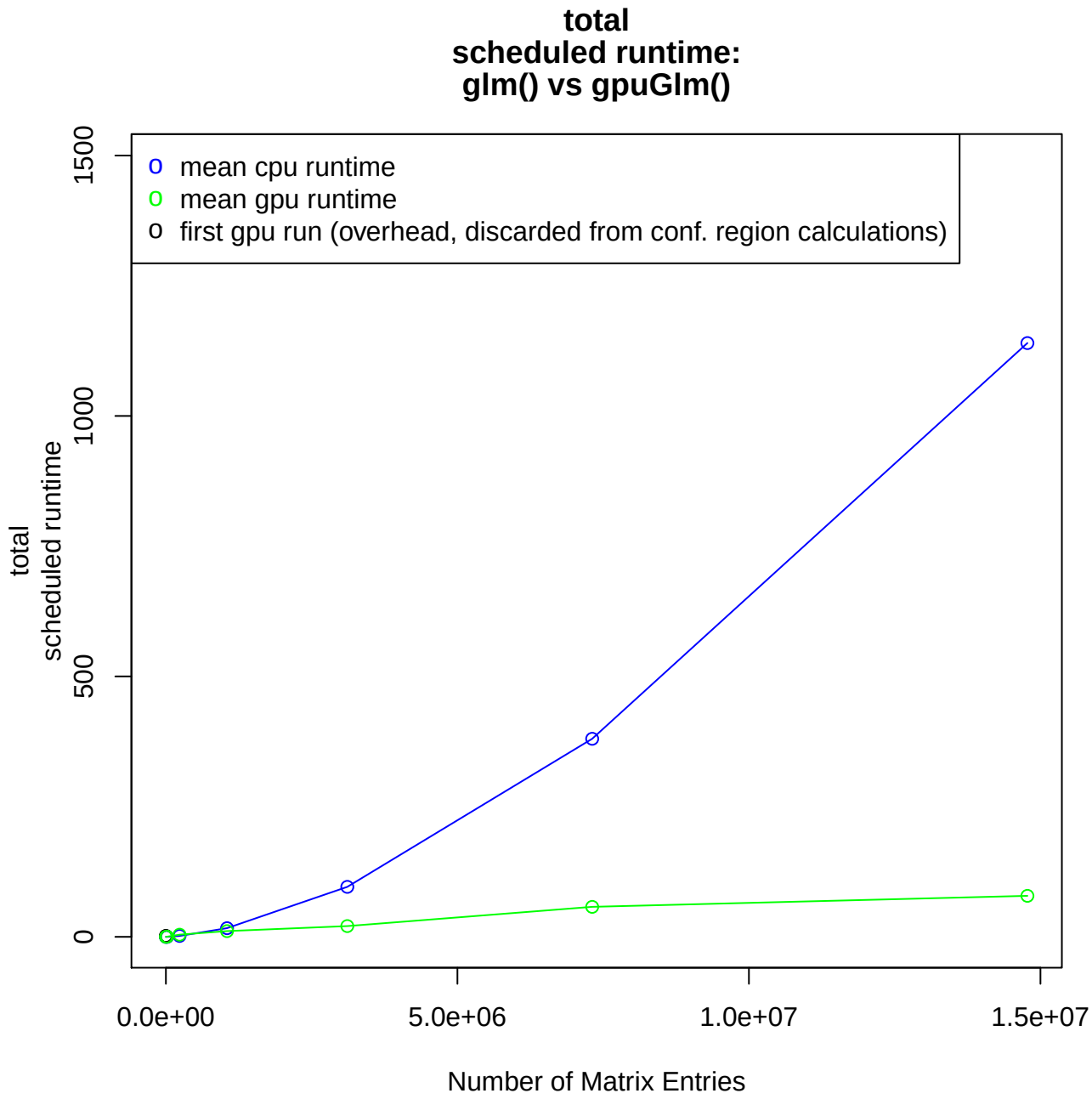

total
scheduled runtime:
lm() vs gpuLm()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
200
400
600
800
0e+00
1e+07
2e+07
3e+07
4e+07
5e+07
6e+07
7e+07
Number of Matrix Entries

A COMPARISON OF gpuGlm() AND glm()

The R script, `gpuGlm.r`, compares the performance of `gpuGlm(y ~ x)` and `glm(y ~ x)` for matrices `y` and `x` of varying sizes.

See the results on the next few slides.


total
scheduled runtime:
glm() vs gpuGlm()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
2
4
6
8
10
12
14
0
50000
100000
150000
200000
Number of Matrix Entries


total
scheduled runtime:
glm() vs gpuGlm()
mean cpu runtime
mean gpu runtime
first gpu run (overhead, discarded from conf. region calculations)
total
scheduled runtime
0
500
1000
1500
0.0e+00
5.0e+06
1.0e+07
1.5e+07
Number of Matrix Entries

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)

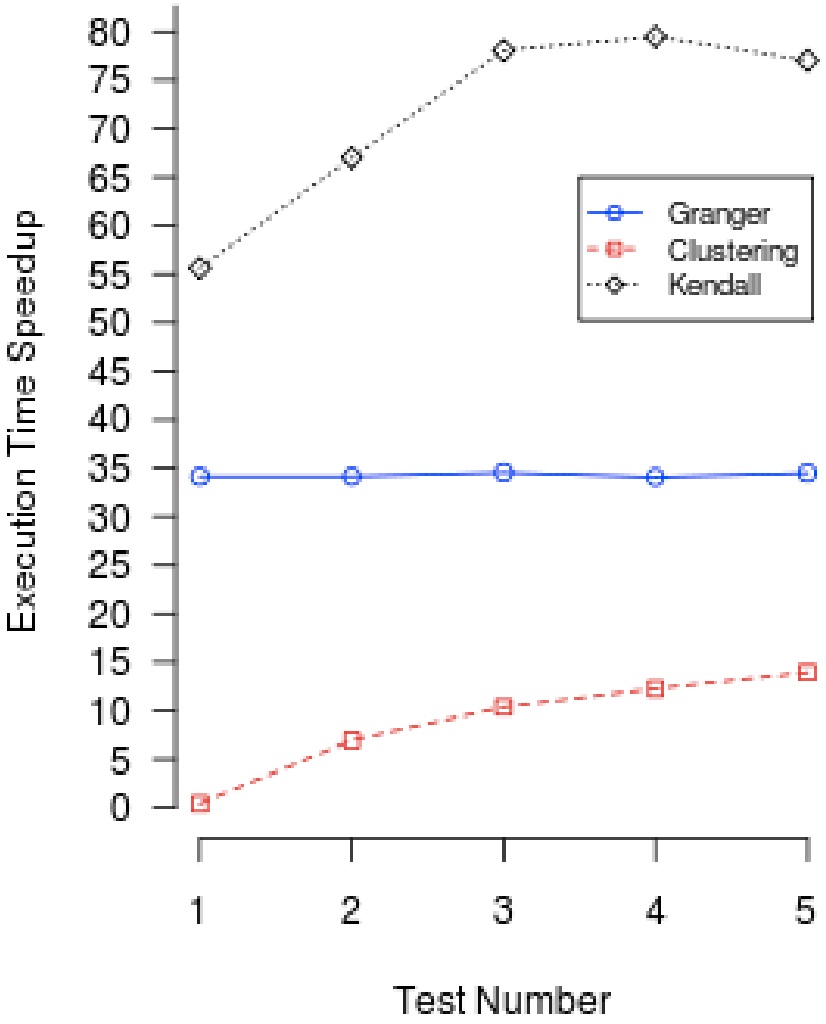


Fig. 2: Granger Times

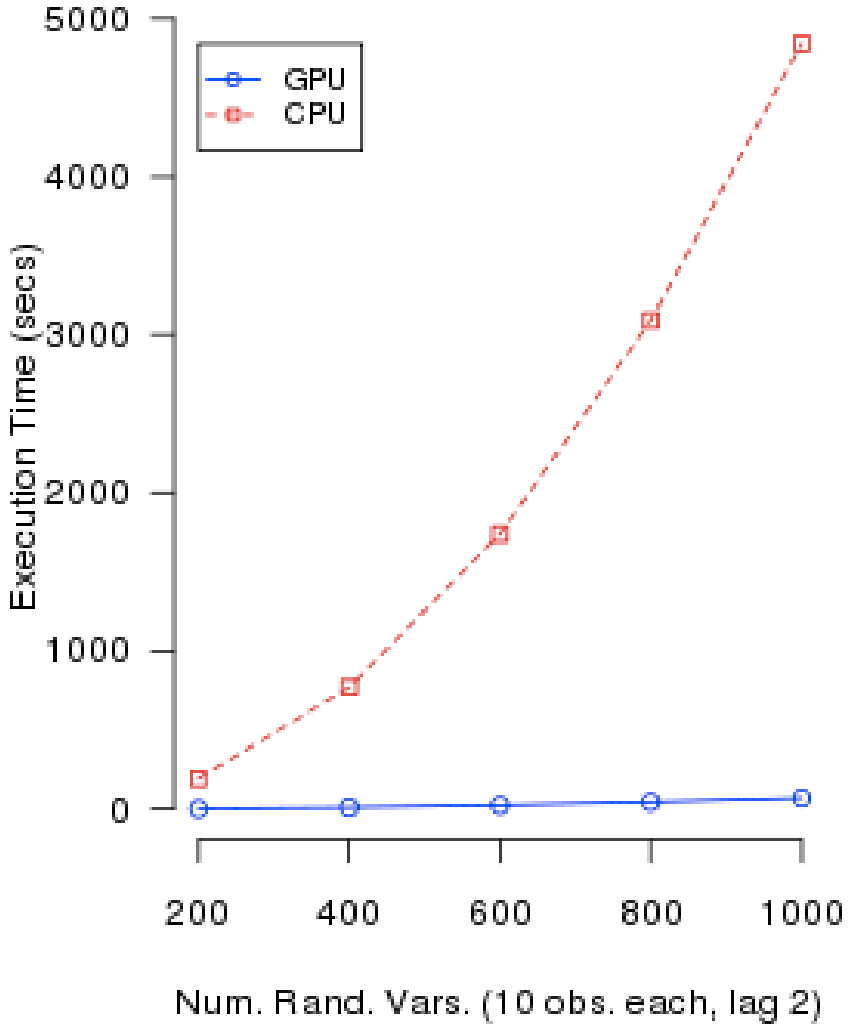


Fig. 3: Cluster Times

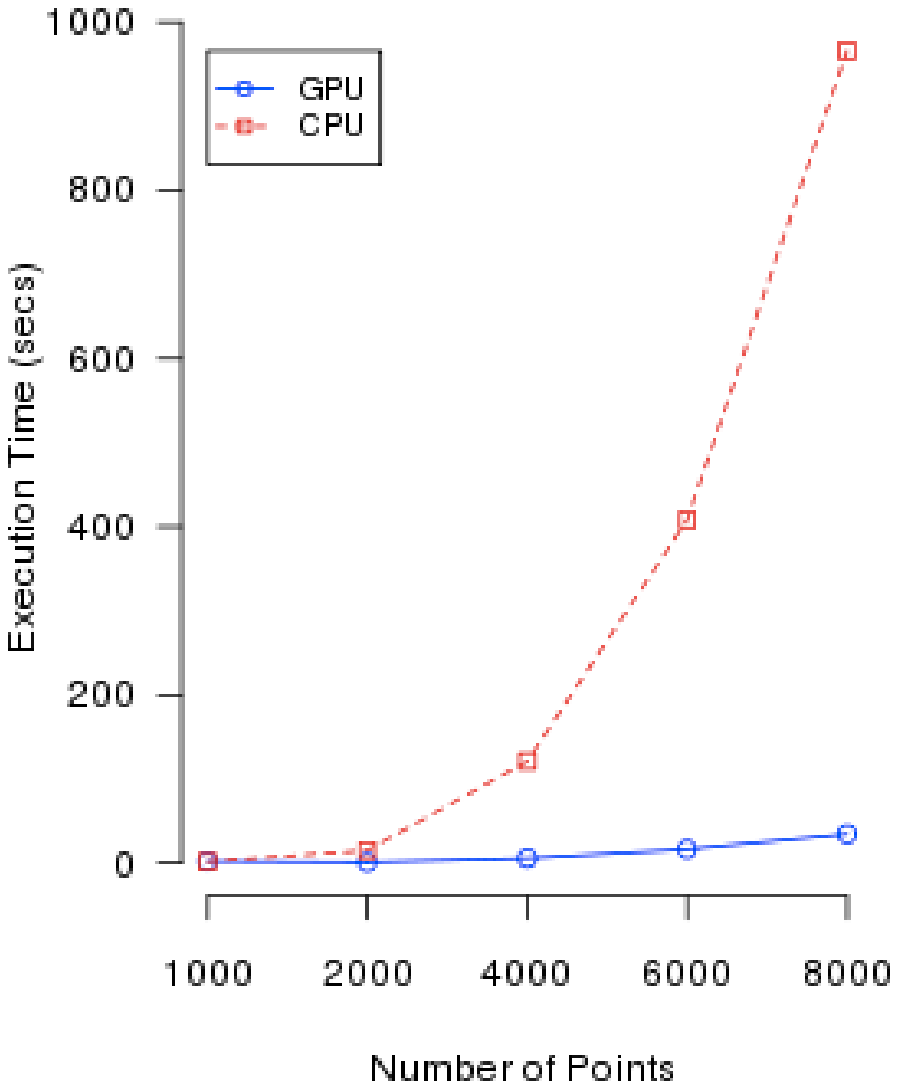
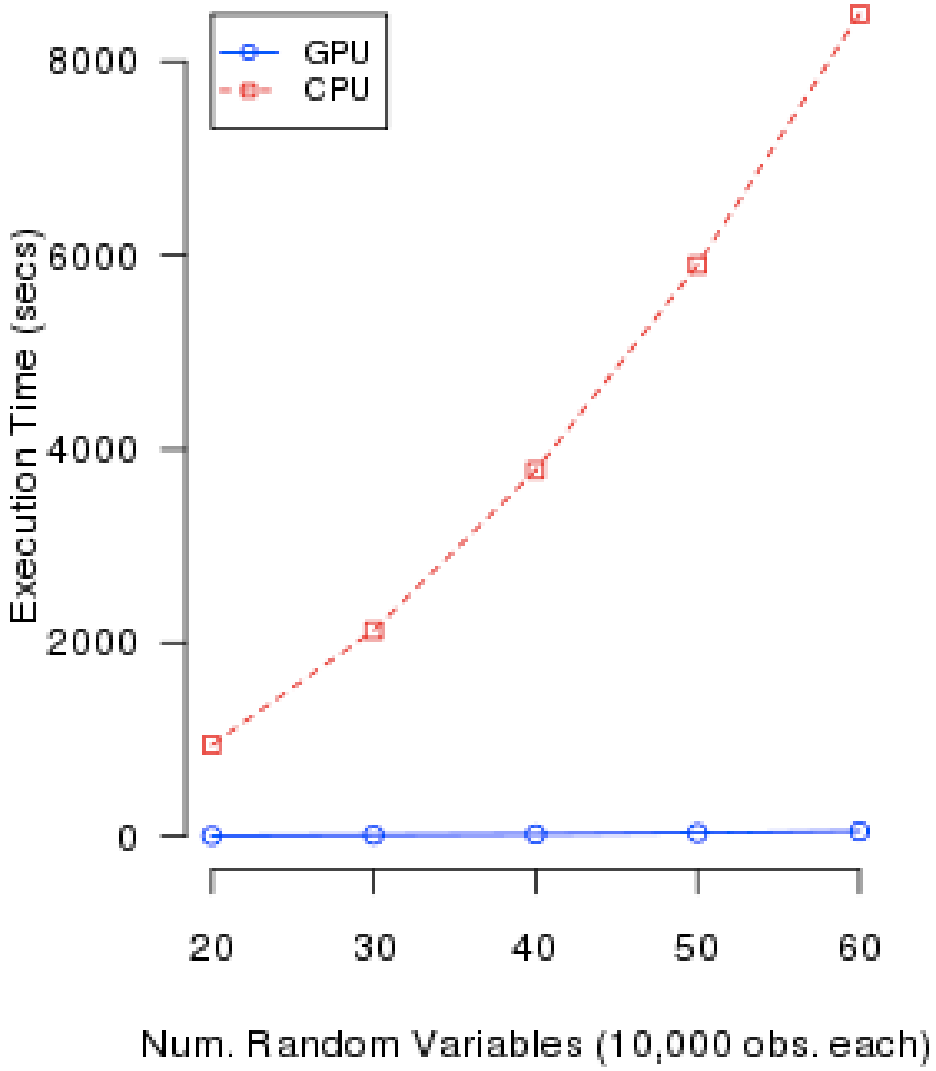


Fig. 4: Kendall Times



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.

LECTURE SERIES MATERIALS

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

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

REFERENCES

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

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