GPU Programming with R

April 15, 2010

Overview

- Introduction.
- Package highlights.
- Getting started.
- Using gputools.
- Examples.
- Conclusions.

GPU $\equiv$ graphical processing unit

- Special–purpose coprocessor for graphics applications.
- Highly parallel hardware with 32–bit vector–processing capabilities.
- Early numerical applications appear to be due to physicists (cf. www.gpgpu.org):
  - Lattice–Boltzmann computations: Li et al., 2002.
  - Boundary–value problems: Goodnight et al., 2003.
  - Required driver knowlege to program - not easy.
- "CUDA" == "Compute Unified Device Architecture", an API from NVidia, freely available now.
- "Stream", from ATI/AMD, also gaining momentum.
- General–purpose GPU's are now inexpensive - often standard equipment.

- CUDA–enabled numerical software becoming available commercially.
  - Jacket, a Matlab accessory from Acclereyes.
  - Mathematica support.
  - Numerous standalone packages on NVidia website.

- CUDA–enabled numerical software becoming available commercially.
  - Jacket, a Matlab accessory from Acclereyes.
  - Mathematica support.
  - Numerous standalone packages on NVidia website.
- Why not R?

- CUDA–enabled numerical software becoming available commercially.
  - Jacket, a Matlab accessory from Acclereyes.
  - Mathematica support.
  - Numerous standalone packages on NVidia website.
- Why not R?
- Buckner et al. release "gputools" 0.1 in spring, 2009.
  - Tools related to that group's work looking for causal relations in gene–expression data.

- CUDA–enabled numerical software becoming available commercially.
  - Jacket, a Matlab accessory from Acclereyes.
  - Mathematica support.
  - Numerous standalone packages on NVidia website.
- Why not R?
- Buckner et al. release "gputools" 0.1 in spring, 2009.
  - Tools related to that group's work looking for causal relations in gene–expression data.
- Co–collaboration leads to paper submitted last summer.
  - "The gputools package enables GPU computing in R"
  - Buckner, Wilson, Seligman, Athey, Watson, Meng
  - *Bioinformatics*, 2010 26(1):134–135

- CUDA–enabled numerical software becoming available commercially.
  - Jacket, a Matlab accessory from Acclereyes.
  - Mathematica support.
  - Numerous standalone packages on NVidia website.
- Why not R?
- Buckner et al. release "gputools" 0.1 in spring, 2009.
  - Tools related to that group's work looking for causal relations in gene–expression data.
- Co–collaboration leads to paper submitted last summer.
  - "The gputools package enables GPU computing in R"
  - Buckner, Wilson, Seligman, Athey, Watson, Meng
  - *Bioinformatics,* 2010 26(1):134–135
- Remains very much a work in progress.

Scope of talk

- We'll be talking about CUDA, although Stream support is on the horizon.
  - CUDA support appeared early and grew quickly.
  - Stream support will entail work with another graphics interface, "OpenCL".
  - Ultimately, we envision unifying Cuda, Stream support in one package - we're just not there yet.
- Primarily Linux, although 32–bit Mac is supported. 64–bit Mac and Windows are to–do.
- Numerical results are 32–bit, although 64–bit hardware support is improving.

- **gputools** package is joint work by MBNI and Rapid Biologics
  - Josh Buckner and Justin Wilson at MBNI.
  - Mark Seligman at Rapid Biologics.
- Package contains some commonly–invoked **R** utilities as well as more specialized functions.
- Current look–and-feel consists of command–level implementations.
- Support continues to grow - based on both demand and ease of implementation.
- Some implementations are whole–cloth, some are just wrappers around code already ported and some lie between these extremes.

- Contributions from MBNI team include:
  - Correlation - Pearson and Kendall (JB/JW): **cor()**
  - Granger causality (JB): **granger.test** from **MSBVAR**
  - Hierarchical clustering (JB/JW): **hclust**
  - Spline–based mutual information (JB)
  - Matrix multiplication (cudablas wrapper): **%*%**
  - SVM training (wrapper): **svm** from **e1071**
  - SVD (wrapper): **fastICA** package
  - attendant functions and package layout

- Contributions from MLS include:
  - Linear, generalized linear modeling: **lm(), glm()**
  - Least–squares fit: **lsfit()**
  - Rank–revealing QR decomposition: **qr()**
  - Blocked, partial–pivoting QR
  - Matrix cross–products: **crossprod()**

Hardware, tools requirements

- At least one GPU supporting CUDA:
  - NVidia GeForce 8, 9, 100, 200, 400–series, with $>$ 256 MB local graphics memory, as well as Quadro, Tesla and Ion products.
  - Includes desktop, notebook, mobile and cluster–based platforms.
  - "Hardware capability" increases with new models.
    - Levels refer to sophistication of feature set.
    - Current levels are $1.1, 1.2, 1.3$.
    - In particular, capability level 1.3 feautres double–precision support.
    - Check NVidia's website for levels of specific GPU cards.

- Can use GPU both to run display and perform computations in "user time".
- Can run graphics separately, if desired, and use GPU card as standalone coprocessor. This makes sense, in particular, when on–board graphics chips already present.
- Can even have multiple GPU, although **gputools** currently configured for single card.
- Cards use PCIe slots, so mainboard capacity is the chief limitation.
- Most importantly, though, you only need one such card and you may already have one.

- CUDA driver: download from NVidia.
  - Driver software provides interface permitting communication with GPU at the API level - i.e., CUDA support.
  - Drivers constantly being updated.
  - Current generation is 3.0, which includes support for new Fermi line.
  - Follow NVidia's instructions for driver installation.
  - Installation procedure differs according to OS. With Linux, in particular, the procedure differs by distribution.
  - Perhaps CUDA support could be included by default, but it currently is not.

- CUDA Toolkit: download from NVidia.
  - Contains compiler and development tools.
  - **gputools** requires these tools in order to build itself.
  - The low–level functions invoked by the package must be compiled for CUDA hardware (or emulator).
  - Package users need not invoke these tools directly, however.
  - Curious users can build and run programs with them. See NVidia's website.
  - Important: It can be helpful to set environment variable CUDA_HOME to the directory under which the toolkit has been installed, typically "/usr/local/cuda". This is a default variable checked by the **gputools** installation.

Obtaining the package

- The **gputools** package can be downloaded from CRAN.
  - Current released version is 0.2.
  - Current tested support for Linux, 32–bit Mac OS X 10.5, 10.6.
  - Other OS to be supported, but will require more time and hardware.
  - Beta versions are available, and contain the latest changes and enhancements.

Beta downloadable from:

http://brainarray.mbni.med.umich.edu/brainarray/rgpgpu/

Installing the package: Linux, Mac

- Installation is the typical "R CMD INSTALL gputools", from the command line.
- Inside **R**, it's "install.packages(gputools)".
- There is an emulation option available, however:
  - "–configure-args='–enable-emulation"'
  - Enables CUDA instructions to be simulated on the CPU.
  - Slow, but useful for development in the absence of a supported card and driver.
  - Suggest Toolkit version 2.3, however.
  - Emulator support to end with Cuda 3.0.
- If CUDA_HOME not set, use "–configure-args='–with-cuda-home=' [path]"

Running package commands

- Once package installed, no need to specify the card: driver identifies device 0, by default.
  - In the case of multiple GPU cards, however, a given card can be specified by invoking "chooseGpu(deviceId=0)".
- Command–level implementations, prefaced with "gpu".
- Familiar **R** commands prefaced by "gpu": e.g., "gpuCor()" for "cor()".
- For those commands having **R** counterparts, the intent is to implement identical parameter lists and return values.
  - 32–bit floating–point is the major exception to this.

Floating–point arithmetic

- 64–bit arithmetic available on newer cards
  - Quite slow by comparison: 8::1 vs. 32 bits.
  - Upcoming hardware (a.k.a. Fermi) to offer 2::1.
- Most commands implemented as 32–bit, due to perfomance disadvantage.
- SVM training and prediction, as well as Kendall correlation, are 64–bit, hence require hardware capability ≥ 1.3.
- We may begin switching to a 64–bit default as performance improves, or at least offer an option on all commands.

Performance considerations - an aside

- The GPU can execute thousands of identical instruction streams, at a somewhat slower clockspeed than the CPU.
- The reason why thousand–fold parallelism remains mostly a theoretical peak has a lot to do with the layout of the data.
  - Some applications are "embarassingly parallel", but most must trade data back and forth between threads: a lot of waiting.
  - Actual placement of data in the GPU makes a difference: there is a memory hierarchy, much as with a CPU (i.e., register/cache/RAM).
  - Applications run fastest when each thread is doing the same thing: data–gated branches mess this up.

- (data layout issues, continued)
  - Getting data between CPU and GPU is quite slow: ca. 500 cycles.
  - Often best to recast the application around these constraints.
- Numerical linear algebra can be recast to minimize such communcation, at the expense of regular updates.
  - QR decomposition, for example, benefits significantly from blocking - i.e., transforming multiple columns at once.
  - Requires more sophisticated implementation, though, if rank–revealing version is required.
- Conversely, less–communicative algorithms can be accelerated to a greater extent.

Performance, overall

- Less–communicative algorithms seeing speedups over 20x on data sets of moderate size:
  - Granger causality (gpuGranger): > 60x.
  - Hierarchical clustering (gpuHClust): > 20x.
- Numerical linear algebra requires larger data sets and experiences less dramatic speedup.
  - Linear modeling (gpuLm): breakeven at 1000x1000 matrix.
  - 15x seen on 4000x8000.
- Speedup factors vary with CPU, memory configuration and, of course, GPU. These figures give some indication of the quality.

Full explanation in help files

- These examples illustrate GPU analogues of functions provided by the standard **R** distribution.
- Most of the example commands support more paramters and options than covered here.
- Others not mentioned include Granger, t-test, SVM methods.
- Check current version for contents.
- Help file should be present for all implemented commands.

gpuCor(x)

- Matrix x of column RV's.
- Input, output formats similar to **cor()**.
- Pearson and Kendall supported, not Spearman.
- Compute capacity ≥ 1.3 needed for Kendall.

gpuMatMult(A, B)

- Conforming matrices A, B.
- Output identical to $A\% * \%B$, up to precision.
- Similarly for **gpuCrossprod(), gpuTcrossprod()**.
- All implemented as wrappers around Cuda's low-level BLAS subroutine .

gpuSolve(x,y=NULL)

- Invert or solve.
- Format of output, input conforms to **solve()**.
- Direct calls to low–level Cuda BLAS.

gpuLm(y   ., data=x)

- Response y and design matrix x.
- Tolerance uses single–precision default.
- Output conforming to **lm()**, although pivot may differ.
- For rank–deficient matrices, ranks may differ. Note that **lm()** is not rank–revealing.
- Seriously considering making RR available as option.
- Similarly for **gpuGlm()**.

gpuLsfit(x, y, wt=weights)

- Design *x*, response *y*, weight *weights*.
- Output conforming to **lsfit(): lm()** without the icing.
- Same considerations about rank, precision, tolerance as **gpuLm()** apply here.

gpuQr(x)

- Matrix x.
- Output conforms to **qr()**.
- Uses rank–revealing pivot, unlike **qr()**.
- Probably wiil relegate RR to option.

gpuSvd(x): Requies CULA tools

- Matrix x.
- Output similar to **svd()**.
- CULA is a third–party toolset with tuned BLAS for GPU.
- **gputools** installation must detect its presence to utilize.
- Install into "/usr/local/cula" or use "CULA_HOME".
- Eventually will provide our own.

What we hope has been shown

- The **gputools** package is easy to install and use.
- A CUDA–ready (and soon, Stream–ready) card is needed, but these are becoming commodities.
- No background in either graphics or parallelization is required of the user.
- Significant performance gains can be realized, depending both on the command invoked and the size of the data.
- Many more features of **R** should benefit from adaptation to the GPU.
- Suggestions always welcome.
- Contacts: bucknerj@umich.edu, Rgpgpu@rapidbiologics.com