SageGPU: Web interface to CUDA development and GPU cloud computing

Igor Rychkov and **Paul Brouwers,**

University of Canterbury, New Zealand, http://dev.math.canterbury.ac.nz/

We acknowledge the Nvidia's hardware donation and the UC College of Engineering's research grant.

1. Purpose and benefits

Our initial aim was to simplify CUDA programming by using a interpreted language to send CUDA code to the GPU. Python and PyCUDA met this need. Sage, http://sagemath.org/ expanded upon this by delivering further flexibility to our programming and publishing needs.

- SageGPU's notebook service provides a web interface into the development environment for Python, C++, and CUDA. This allows anyone with a web browser and internet connection a remote access to our server's GPU (Tesla Fermi C2070) and CPU.
- The server maintenance is provided by the University of Canterbury, NZ. The developer is freed from hardware and software maintenance.
- SageGPU builds on the Sage notebook server, http://nb.sagemath.org/ and uses Python to author and run C++/CUDA programs, via PyCUDA or calling NVCC complier directly.
- The worksheet combines executable cells with Python code, surrounded by documentation in HTML and $\LaTeX$ format allowing for links, images and video. The cells can be evaluated in sequence as the reader progresses through the worksheet.
- Realization of "programs writing other programs": Python is used to organize the workflow and to customize templated C++/CUDA source strings for specific parameters.
- SageGPU is a novel solution for collaboration and publishing:
  - Worksheets can be shared with other SageGPU users to enable collaboration among researchers or teams.
  - The worksheet can be published giving it a permanent URL for sharing of code and results

2. What is Sage?

Features important to SageGPU are in bold.

- Sage is a free open-source mathematics software. It combines the power of many existing open-source packages into a common **Python-based interface**. A worksheet interface is similar to Mathematica notebook: **Shift-Enter to evaluate the cell**, interactive plots, etc. Sage is built out of nearly 100 open-source packages and features a unified interface. Sage can be used to study elementary and advanced, pure and applied mathematics. It combines various software packages and seamlessly integrates their functionality into a common experience. It is well-suited for education and research.
- **The user interface is a notebook in a web browser** or the command line. Using the notebook, Sage connects either locally to your own Sage installation or to **a Sage server on the network**. Inside the Sage notebook you can create embedded graphics, beautifully typeset mathematical expressions, add and delete input, and share your work across the network.
- Special Cells in Sage notebooks
  - %auto – cell content evaluated automatically
  - %hide – to hide cells for better presentation
  - %fortran – inline fortran code that will be compiled and linked automatically
  - %interact – for interactive graphics etc.
  - **%sh – the cell's content is executed in the servers Linux shell**

3. Workflow of working with SageGPU, all in one web document

Edit

- Create the Python cells and C++/CUDA (template) source strings
- "Programs writing other programs": use Python to define the algorithm and auxiliary steps, putting only the very essential, critical parts to the C++/CUDA source strings; enter and test the parameters, and bake a customized, small C++/CUDA code allowing for faster and more efficient compilation step.
- Allocate memory for arrays (e.g., *numpy*)

Compile/load/debug C++/CUDA modules

- using either PyCUDA or calling nvcc from the shell
- with *printf*-debugging against syntax-highlighted, line-numbered view of C++/CUDA source

Run

- calling into system's shell to execute EXE, or using ctypes to load the DLL, or using *PyCUDA* to load the compiled CUDA to the graphics driver
- passing pointers into the compiled DLL or through *PyCUDA* interface
- short or many days runs are possible

Analyze

- Data is available back in Python through *NumPy,ctypes,PyCUDA* interfaces
- Format results in HTML tables, pictures, even interactive controls. This is another example of metaprogramming: Sage notebook amending HTML into itself

Collaborate

- Worksheets can be shared with other SageGPU users to enable collaboration among researchers or teams.

Publish

- The worksheet can be published giving it a permanent URL for sharing of results.
- Better than source code publishing: rich active document containing code, usage, background reading and results.
- The published worksheet can be run again on the server by the reader enabling ready confirmation of the results.

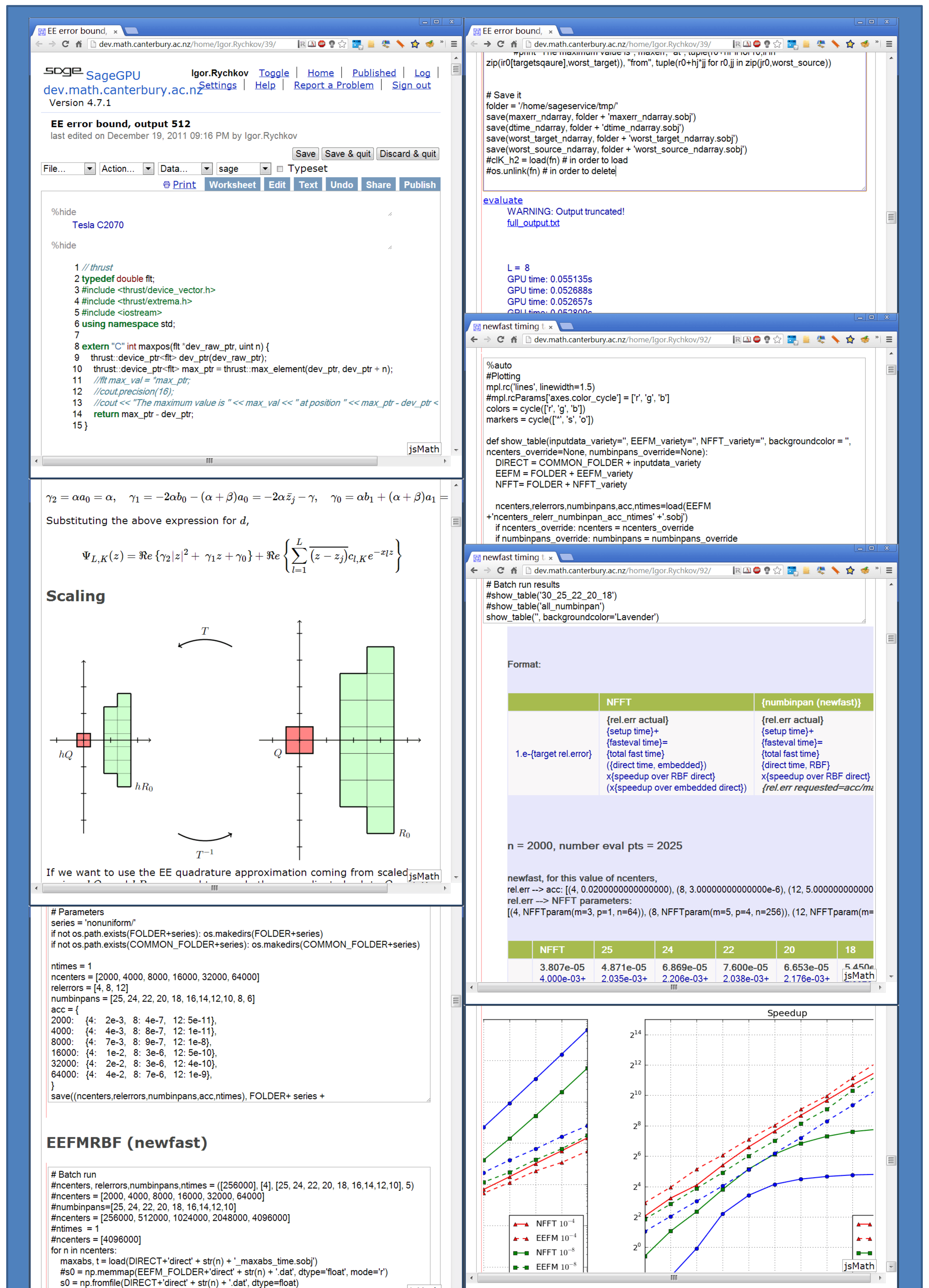

Figure 1. Screenshots showing example worksheet (RBF approximation) with syntax highlighted CUDA source, HTML/ $\LaTeX$ documentation, execution parameters and output, inline Python, tables and graphs.

4. Accessing GPU via PyCUDA

Easy to install into Sage, Python and PyCUDA, http://documen.tician.de/pycuda/, a library for CUDA programming. This provides an interface interface to all essential CUDA features, automatic initialization of CUDA with error checking, convenient and thin abstractions like events, *gpuarray* for memory allocation and copy, *SourceModule* (also *ReductionKernel* etc.) for CUDA kernel compilation/calling,

```
import pycuda.autoinit
print pycuda.autoinit.device.name()
import pycuda.driver as cuda
#create two timers for speed-testing
start = cuda.Event(); end = cuda.Event()
from pycuda.compiler import SourceModule
import pycuda.gpuarray as gpuarray
dt = np.long # 64-bit integers
from pycuda.reduction import ReductionKernel
a_gpu = gpuarray.arange(400000, dtype=dt)
sumreduce = ReductionKernel(dt, neutral="0",
      reduce_expr="a+b", map_expr="x[i]",
      arguments="long *x")
sum_gpu =  sumreduce(a_gpu)
print "sum (PyCUDA):", sum_gpu.get()[()]
```

5. Accessing GPU with using the shell to call nvcc

- Author or customize templated C++/CUDA/Thrust source
- Use %sh cell or system call to compile it with nvcc
- use Python's library ctypes for loading dynamic shared libraries (inprocess sharing of GPU memory pointers)

```
thrust_source = """// sumreduce using thrust, C++ source
#include <thrust/device_vector.h>
#include <thrust/reduce.h>
extern "C" long sumreduce(long *dev_raw_ptr, long n) {
   thrust::device_ptr<long> dev_ptr(dev_raw_ptr);
   return thrust::reduce(dev_ptr, dev_ptr + n, long(0), thrust::plus<long>());
}"""
# A folder for saving the source and the DLL into
FOLDER = '/home/sageservice/tmp/'
# Hashed filename is different whenever the source changes!
thrust_fn = FOLDER +'sumreduce' + str(hash(thrust_source))
if not os.path.exists(thrust_fn + '.so'):
   f = open(thrust_fn + '.cu', 'w')
   f.write(thrust_source)
   f.close()
   # System call to nvcc!
   err = call(["nvcc","-arch","sm_20","-Xcompiler","-fpic",
            "-shared","-o", thrust_fn + '.so', thrust_fn + '.cu'])
   cprint(thrust_source) # pretty print for debugging
   print not(err)
# Load using ctypes!
DLL = ctypes.CDLL(thrust_fn + '.so')
DLL.sumreduce.restype = ctypes.c_long
# Call DLL function with GPU array pointers properly cast
sum_thrust = DLL.sumreduce(ctypes.c_ulong(a_gpu.ptr),
                ctypes.c_long(len(a_gpu)))
print "sum (thrust GPU):", sum_thrust
```

6. Example usage

- Research: fast Radial Basis Function evaluation methods, Figure 1.
- Teaching: Parallel algorithms in computational mathematics, course lectures on CUDA programming
- See other published worksheets, Figure 2.

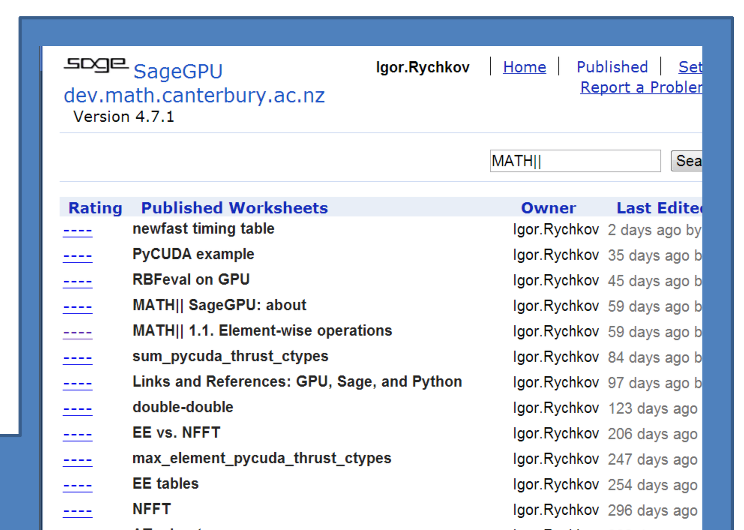

Figure 2. Example worksheets published at http://dev.math.canterbury.ac.nz/pub