Debugging CUDA

W. B. Langdon

CREST lab,

Department of Computer Science

GECCO 2011 Companion,
pages 415-422

Introduction

- Some ideas on debugging GPGPU code
- 1st of two parts. 2nd part on performance
- Code level debug aids, rather than tools
- Testing
- Example errors
- Lessons

Defensive Programming

- Hard to debug kernel which fails because get no feed back.
- Write description of all kernel parameters *before* each is started to a log file.

```
printf("kernel<<<%d,%d,%d>>>(%d,%d,%d,<%d>,<%d>,<%d>:",
       grid_size, block_size, shared_size,
       height,width,len,
       len*sizeof(int),
       len*width*sizeof(unsigned int),
       len*sizeof(int));
printf("<%d>,<%d><%d>)\n", //outputs
       len*width*sizeof(unsigned int),
       len*width*sizeof(unsigned int),
       3*sizeof(int));

kernel<<<grid_size, block_size, shared_size>>>
(height,width,len,d_in,d_a,d_y,d_out1,d_out2,d_status);
cutilCheckMsg("kernel() execution failed.\n");
```

Defensive Programming - Loops

- In most kernels there are no loops or only one
- Trap all potential infinite loops inside kernel

```
int loop =  0;  //prevent looping forever
do {
  if(found) break;
  if(empty) break;
  //next
} while(loop++ < Nvalue);
```

Kernel Launch Failure

- Always check kernel status immediately with `cutilCheckMsg("kernel_name execution failed.\n");`
  - This (and your log) will help you pinpoint which kernel failed.
  - Sometimes the `cutil` error message can help
- `cuda-memcheck --continue` can sometimes locate array bound errors inside your kernel. Too slow for normal use.

First Kernel

- Write a kernel which does nothing except check:
  - Does input reach the kernel?
  - Does output leave the kernel?
  - Do threads put data in correct place?
  - Is output correct?

```
static __global__ void kernel(
  const int LEN,
  int d_1D_out[1000]    //check kernel creates correct output
) {
  const int tid     = blockDim.x * blockIdx.x + threadIdx.x;
  const int threadN = blockDim.x * gridDim.x;
  for(unsigned int t = tid; t < LEN; t += threadN){
    d_1D_out[t] = threadIdx.x;
  }
}
```

Debugging your First Kernel

- Did your first kernel work?
- Test your debugging system by adding an error.
- Did the kernel fail in the way you expected?
- Did your error trapping code catch the error and report it?
- Did your revision control system allow you to recover your working version reliably, correctly, with a minimum of manual input?

Debug

- More examples of debug code in paper.
- Saving GPU buffers
- Testing…

Testing

- New code is wrong
- Modified code is wrong
- Testing is second best way of finding errors

- Testing Evolutionary Algorithms
- Comparison with known answers
- Regression Testing
- Source code version management

Testing GAs

- Evolutionary Algorithms can evolve high scoring “solutions”.
- “Solution” can be a bug in fitness function. Eg robotics simulations.
- EA can work around bug in itself
- Do not assume your system is working because it evolves good looking answers

CREST

Comparison with Known Answers

- Are there benchmarks with correct answers?
- Is there a serial version (is it bug free)?
- Can you easily create a serial version?
  – Need not be efficient, just correct

Comparison with Known Answers

- Easy to overlook differences and assume they are small and unimportant.
- Insist your GPU produces identical answers.
- Carefully control use of random seeds
- With floating point GPU will produce different answers.
  - Decide in advance size of acceptable difference
  - Do you want -0, NaN etc to be "different"?

Regression Testing

- Modified code is wrong
- Comparing your “improved” code’s output with previous outputs can help locate errors.

Revision Control

- Modified code is wrong
- The best way of locating faults is comparing your “improved” code with the previous version.
- Your revision control system should make it easy to compare versions of your code.
- Ensure you have an automated way of recording which version of your code produced which outputs. This can help greatly in regression testing.

GPU Bugs

- Too many examples!!!
  - For example, see proceedings (pages 415-423)
- I have chosen three related to GPU

GPU Bugs – Missing threads

```
__device__ void save_data(const unsigned int mask,...) {
  ...
}


...

if(data) {
  ... lookup data ...
  if(missing) save_data(data,...);
}
```

- From the calling code, we can see save_data() is only called by threads for which data is both non-zero and missing.
- This is not obvious when looking at save_data()'s code. Where I assumed all threads in a warp were calling it.

volitile

- volatile turns off nvcc optimisation whereby it uses per thread registers.
- Using shared memory to communicate between threads
- Make every pointer to shared memory volatile

```
__device__ void insert(const unsigned int mask,
                       const int Nmask, volatile unsigned int* shared_mask) {
```

C not fully defined, int >>24

- C right shift operation can either perform an arithmetic or a logical shift.
- To fix this I declared the variable `unsigned int` rather than `int`

```
         int x = 0x80000000;
unsigned int y = 0x80000000;
x >> 24; //gives 0xffffff80 (-127)
y >> 24; //gives 0x00000080 ( 128)
```

Discussion

- Debug driven from host
  - printf, GPU debug direct to monitor, GPU emulator gone
- CUDA
  - CUDA works
    - Mostly (nvcc etc pretty stable) visual profiler poor
  - C, I guess you can have bugs in other languages
  - openCL
- Linux
  - Eclipse?
  - Microsoft visual studio?
- Commercial Tools?

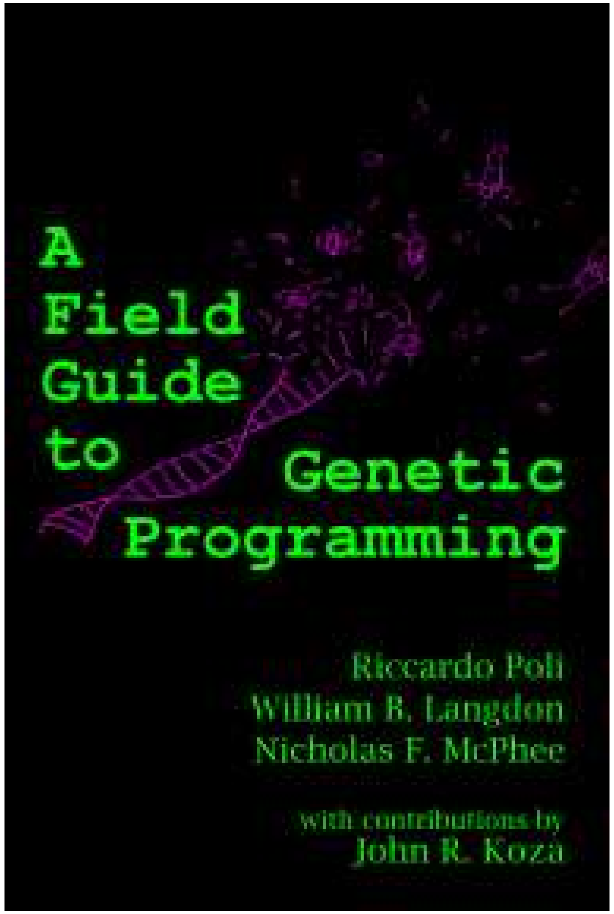
Conclusions

- YOU ARE THE BOOTLE NECK
- Writing working high performance GPGPU code is hard.
- Four CIGPU events BUT creating evolutionary algorithms to effectively use GPU is still hard
- Establish libraries of debugged code?
- Can problem be expressed as matrix manipulation? Use cublas library?

END

http://www.epsrc.ac.uk/ EPSRC

A Field Guide To
Genetic Programming
http://www.gp-field-guide.org.uk/



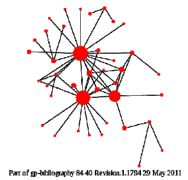
Free
PDF

The Genetic Programming Bibliography

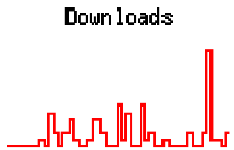
The largest, most complete, collection of GP papers.
http://www.cs.bham.ac.uk/~wbl/biblio/

With 7554 references, and 5,895 online publications, the GP Bibliography is a vital resource to the computer science, artificial intelligence, machine learning, and evolutionary computing communities.

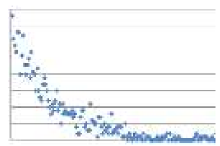
RSS Support available through the Collection of CS Bibliographies. XML RSS



A web form for adding your entries. Wiki to update homepages. Co-authorship community. Downloads



A personalised list of every author's GP publications.



Search the GP Bibliography at
http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html