Evolving a CUDA Kernel from an nVidia Template

W. B. Langdon
CREST lab,
Department of Computer Science

11.5.2011
Introduction

• Using genetic programming to create C source code
  – How? Why?

• Proof of concept: gzip on graphics card
  – Template based on nVidia kernel
  – BNF grammar
  – Fitness

• Lessons: it can be done!

• Future GISMO:
  Genetic Improvement of Software for Multiple Objectives
GP to write source code

• When to use GP to create source code
  – Small. E.g. glue between systems.
  – Hard problems. Many skills needed.
  – Multiple conflicting ill specified non-functional requirements

• GP as tool. GP tries many possible options. Leave software designer to choose between best.
GP Automatic Coding

- Target small unit.
- Use existing system as environment holding evolving code.
- Use existing test suite to exercise existing system but record data crossing interface.
- Use inputs & *answer* (Oracle) to train GP.
- How to guide GP initially?
- Clean up/validate new code
GP Automatic Coding

- Actual data into and out of module act as *de facto* specification.
- Evolved code tested to ensure it responds like original code to inputs.
- Recorded data flows becomes test Oracle.
Proof of Concept: gzip

- Example: compute intensive part of gzip
- Recode as parallel CUDA kernel
- Use nVidia’s examples as starting point.
- BNF grammar keeps GP code legal, compliable, executable and terminates.
- Use training data gathered from original gzip to test evolved kernels.
- Why gzip
CUDA 2.3 Template

• nVidia supplied 67 working examples.
• Choose simplest, that does a data scan. (We know gzip scans data).
• Naive template too simple to give speed up, but shows plausibility of approach.
• NB template knows nothing of gzip functionality. Search guided only by fitness function.
gzip

• gzip scans input file looking for strings that occur more than once. Repeated sequences of bytes are replaced by short codes.
• $n^2$ reduced by hashing etc. but gzip still does 42 million searches (sequentially).
• Demo: convert CPU hungry code to parallel GPU graphics card kernel code.
Fitness

- Instrument gzip.
- Run gzip on SIR test suite. Log all inputs to longest_match(). 1,599,028 records.
- Select 29,315 for training GP.
- Each generation uses 100 of these.
Fitness

- Pop=1000. 100 kernels compiled together.
  - Compilation time $= 7 \times$ run time.
- Fitness testing
  - first test’s data uploaded to GPU 295 GTX.
  - 1000 CUDA kernels run on first test.
  - Loop until all 100 tests run.
- Answers compared with gzip’s answer.
- performance $= \sum |\text{error}| + \text{penalty}$
  - kernels which return 0 get high penalty.
Performance of Evolving Code
Evolved gzip matches kernel

Parse tree of solution evolved in gen 55. Ovals are binary decision rules. Red 2\textsuperscript{nd} alternative used.
Evolved gzip matches kernel

```c
__device__ int kernel978(const uch *g_idata, const int strstart1, const int strstart2)
{
  int thid = 0;
  int pout = 0;
  int pin = 0;
  int offset = 0;
  int num_elements = 258;
  for (offset = 1 ; G_idata( strstart1+ pin ) == G_idata( strstart2+ pin ) ; offset ++)
  {
    if(!ok()) break;
    thid = G_idata( strstart2+ thid ) ;
    pin = offset ;
  }
  return pin ;
}
```

Blue  - fixed by template. Red  - evolved
Black - default   Grey – evolved but no impact.
Conclusions

• Have shown possibility of using genetic programming to automatically re-engineer source code

• Problems:
  – Will users accept code without formal guarantees?
  – Evolved code passes millions of tests.
  – How many tests are enough?

• First time code has been automatically ported to parallel CUDA kernel by an AI technique.
Discussion

• Why evolve C code
  – Small. E.g. glue between systems.
  – Hard problems. Many skills needed.
  – Multiple conflicting ill specified non-functional requirements
  – GP as tool. GP tries many possible options. Choice by designer
• Will users accept code without formal guarantees?
• Other approaches:
  – Template based on nVidia kernel
  – Other grammars
  – Fitness, co-evolution, interactive evolution
• Other demonstrations
• GISMO: Genetic Improvement of Software for Multiple Objectives
// WBL 30 Dec 2009 $Revision: 1.11 $ Remove comments, blank lines. int g_odata, uch g_idata. Add
strstart1 strstart2, const.
move offset and n, rename n as num_elements
WBL 14 r1.11 Remove crosstalk between threads threadIdx.x, temp -> g_idata[strstart1/strstart2]
__device__ void scan_naive(int *g_odata, const uch *g_idata, const int strstart1, const int strstart2)
{
    //extern __shared__ uch temp[];
    int thid = 0; //threadIdx.x;
    int pout = 0;
    int pin = 1;
    int offset = 0;
    int num_elements = 258;
    <3var> /*temp[pout*num_elements+thid]*/ = (thid > 0) ? g_idata[thid-1] : 0;
    for (offset = 1; offset < num_elements; offset *= 2)
    {
        pout = 1 - pout;
        pin = 1 - pout;
        //__syncthreads();
        //temp[pout*num_elements+thid] = temp[pin*num_elements+thid];
        <3var> = g_idata[strstart+pin*num_elements+thid];
        if (thid >= offset)
            <3var> += g_idata[strstart+pin*num_elements+thid - offset];
    }
    //__syncthreads();
    g_odata[threadIdx.x] = <3var>
BNF grammar

scan_naive_kernel.cu converted into grammar (169 rules) which generalises code.

```plaintext
<line10-18> ::= "" | <line10-18a>
<line10-18a> ::= <line10e> <line11> <forbody> <line18>
<line11> ::= "{n} "if(!ok()) break;n"
<line18> ::= "}n"
<line10e> ::= <line10> | <line10e1>
<line10e1> ::= "for (offset =" <line10.1> ";" <line10e.2> ";offset" <line10.4> ")\n"<line10.1> ::= <line10.1.1> | <intexpr>
<line10.1.1> ::= "1" | <intconst>
<line10e.2> ::= <line10e.2.1> | <forcompexpr>
<line10e.2.1> ::= "offset" <line10.2> <compexpr>
<line10.2> ::= "<" | <compare>
<line10.3> ::= <line10.3.1> | <intexpr>
<line10.3.1> ::= "num_elements" | <intconst>
<line10.4> ::= "*= 2" | <intmod>
<intmod> ::= "++" | <intmod2>
<intmod2> ::= "*=" | <intconst>
```

Fragment of 4 page grammar
gzip longest_match()
Number of Strings to Check

gzip hash means mostly longest_match() has few strings to check.
Training data more evenly spread.
Length of Strings to Check

1% 0 bytes
0% 1 bytes
0% 2 bytes
30% 3 bytes
26% 4 bytes
25% 5 bytes
14% 6 bytes

gzip heuristics limit search ≤ 258
Debug

• Debugging hard
• Eventually replaced last member of evolved population with dummy
• Dummy reflects back input to host PC.
• Enables host to check:
  – Training data has reached GPU
  – Kernel has been run
  – Kernel has read its inputs
  – Kernel’s answer has been returned to host PC.
Fall in number of poor programs

71% useless constants in generation 0

7% constants

Graph showing the decrease in the number of GP individuals over generations with a decrease from 800 to 100.
Evolution of program complexity
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/

Contact W.Langdon to get your GP papers included

href link to list of your GP publications. For example mine is

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