Genetically Improved BarraCUDA

CREST Annual Research Review: Recent Results and Research Trends
15-16th June 2015

W. B. Langdon
Department of Computer Science
Genetically Improved BarraCUDA

W. B. Langdon
Department of Computer Science
Genetically Improved BarraCUDA

• to be presented at GECCO 2015
  – Improving CUDA DNA Analysis Software with GP
  – Technical Report RN/15/03

• Background
  – What is BarraCUDA
  – GP to improve CUDA kernel

• Results
  – Speedup
  – GCAT benchmark, demonstrate 1\textsuperscript{st} GI in use
What is BarraCUDA?

DNA analysis program

- 8000 lines C code, SourceForge. (Actually used SVN)
- Rewrite of BWA for nVidia CUDA

Speed comes from processing 159,744 strings in parallel on GPU
BarraCUDA 0.7.107

Manual host changes to call exact_match kernel GI parameter and code changes on GPU
Why 1000 Genomes Project?

• Data typical of modern large scale DNA mapping projects.
• Flagship bioinformatics project – Project mapped all human mutations.
• 604 billion short human DNA sequences.
• Download raw data via FTP

$120million 180Terra Bytes
Preparing for Evolution

- Re-enable **exact matches** code
- **Support 15 options** (conditional compilation)
- **GP fitness testing framework**
  - Generate and compile 1000 unique mutants
    - Whole population in one source file
    - Remove mutants who fail to compile and then re-run compiler to compile the others
  - Run and measure speed of 1000 kernels
    - Reset GPU following run time errors
  - For each kernel check 159444 answers
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
<th>Lines of code affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_W</td>
<td>int</td>
<td>64</td>
<td>all</td>
</tr>
<tr>
<td>cache_threads</td>
<td>int</td>
<td>“”</td>
<td>44</td>
</tr>
<tr>
<td>kl_par</td>
<td>binary</td>
<td>off</td>
<td>19</td>
</tr>
<tr>
<td>occ_par</td>
<td>binary</td>
<td>off</td>
<td>76</td>
</tr>
<tr>
<td>many_blocks</td>
<td>binary</td>
<td>off</td>
<td>2</td>
</tr>
<tr>
<td>direct_sequence</td>
<td>binary</td>
<td>on</td>
<td>63</td>
</tr>
<tr>
<td>direct_index</td>
<td>binary</td>
<td>on</td>
<td>6</td>
</tr>
<tr>
<td>sequence_global</td>
<td>binary</td>
<td>on</td>
<td>16</td>
</tr>
<tr>
<td>sequence_shift81</td>
<td>binary</td>
<td>on</td>
<td>30</td>
</tr>
<tr>
<td>sequence_stride</td>
<td>binary</td>
<td>on</td>
<td>14</td>
</tr>
<tr>
<td>mycache4</td>
<td>binary</td>
<td>on</td>
<td>12</td>
</tr>
<tr>
<td>mycache2</td>
<td>binary</td>
<td>off</td>
<td>11</td>
</tr>
<tr>
<td>direct_global_bwt</td>
<td>binary</td>
<td>off</td>
<td>2</td>
</tr>
<tr>
<td>cache_global_bwt</td>
<td>binary</td>
<td>on</td>
<td>65</td>
</tr>
<tr>
<td>scache_global_bwt</td>
<td>binary</td>
<td>off</td>
<td>35</td>
</tr>
</tbody>
</table>
Evolving BarraCUDA kernel

- Convert manual CUDA code into grammar
- Grammar used to control code modification
- GP manipulates patches and fixed params
  - Small movement/deletion of existing code
  - New program source is syntactically correct
  - Automatic scoping rules ensure almost all mutants compile
- Force loop termination
- GP continues despite compilation and runtime errors
Compile Whole Population

Compiling many kernels together is about 20 times faster than running the compiler once for each.
Evolving BarraCUDA

Manually written CUDA source code

BNF Grammar

Select

Fitness

1000 unique CUDA kernels
(code and conditional compilation changes)

Population of modifications

Mutation and Crossover

Population of modifications

Improved exact match and device code

Thousand Genomes Project
Test case
if (*lastpos!=pos_shifted)
{
    #ifndef sequence_global
        *data = tmp = tex1Dfetch(sequences_array, pos_shifted);
    #else
        *data = tmp = Global_sequences(global_sequences, pos_shifted);
    #endif /*sequence_global*/
    *lastpos=pos_shifted;
}

CUDA lines 119-127

<119> ::= " if" <IF_119> " \n"
<IF_119> ::= "(*lastpos!=pos_shifted)"
<120> ::= "{"\n<121> ::= "#ifndef sequence_global\n"
<122> ::= "" <_122> "\n"
<_122> ::= "*data = tmp = tex1Dfetch(sequences_array, pos_shifted);"
<123> ::= "#else\n"
<124> ::= "" <_124> "\n"
<_124> ::= "*data = tmp = Global_sequences(global_sequences, pos_shifted);"
<125> ::= "#endif\n"
<126> ::= "" <_126> "\n"
<_126> ::= "*lastpos=pos_shifted;"
<127> ::= "}\n"

Fragment of Grammar (Total 773 rules)
9 Types of grammar rule

- Type indicated by rule name
- Replace rule only by another of same type
- 650 fixed, 115 variable.
- 43 statement (eg assignment, Not declaration)
- 24 IF
  - \(<_{392}\> ::= " if" \(<_{IF\_392}\> " \{\n"
  - \(<_{IF\_392}\> ::= " (par==0)"
- Seven for loops (for1, for2, for3)
  - \(<_{630}\> ::= <okdeclaration_\_> <pragma_630> "for(" <for1_630> ";" "OK()&&" <for2_630> ";" <for3_630> ")\n"
- 2 ELSE
- 29 CUDA specials
CUDA specials and configuration parameters

- BNF special types for CUDA
  - optrestrict apply `__restrict__` to all pointer arguments
  - launchbounds applies on starting CUDA kernel
  - `#pragma unroll`

- 15 Parameters
  - If parameter is not "", C macro `#define` created holding value of parameter.
  - Macro used in code, eg via conditional compilation
  - Cleared with `#undef` before next mutant is compiled
Representation

• 15 fixed parameters; variable length list of grammar patches.
  • no size limit, so search space is infinite
• tree like 2pt crossover.
• mutation adds one randomly chosen grammar change
• 3 possible grammar changes:
  • Delete line of source code (or replace by “”, 0)
  • Replace with line of GPU code (same type)
  • Insert a copy of another line of kernel code
Example Mutating Grammar

\(<_947> \ ::= \ "*k0 = k;" \)
\(<_929> \ ::= \ "((int*)l0)[1] = __shfl(((int*)&l)[1],threads_per_sequence/2,threads_per_sequence) ;" \)

2 lines from grammar

\(<_947> +<_929> \)

Fragment of list of mutations
Says insert copy of line 929 before line 947

New code

\(((int*)l0)[1] = __shfl(((int*)&l)[1],threads_per_sequence/2,threads_per_sequence);*k0 = k; \)

W. B. Langdon, UCL
Example2 Mutating Grammar

\[ \langle_k_{kernel\_bnf.cu\_126} \rangle ::= "\ast_{lastpos=pos\_shifted;}" \]

1 line from grammar

\[ \langle_126\rangle \]

Fragment of list of mutations

Says delete line 126

W. B. Langdon, UCL
Recap

• Representation
  – 15 fixed genes (mix of Boolean and integer)
  – List of changes (delete, replace, insert). New rule must be of same type.
    • no size limit, so search space is infinite

• Mutation
  – small/large change
  – append one random change

• Crossover
  – Uniform crossover
  – tree like 2pt crossover.
Testing exact_match kernel variants

- Apply 1000 GP patches (plus original)
- Compile specifically for GPU in use.
- Run on 159744 randomly chosen 100 base pair DNA sequences (fixed sequence).
- Calculate time taken and check answers.
- Only those returning correct answers quicker than manual code can breed.
- Choose fastest 500 to be parents.
- Mutate, crossover: 2 children per parent.
- Repeat 50 generations.
Run time errors

• Automated scoping rules ensure during evolution 96.5% compile. (Each BNF rule annotated with line numbers where it can be copied to.)
  • Mutants which fail to compile are removed from cuda.cuh and the compiler is re-run

• Almost all kernels run and terminate
  • Long running loops are aborted by OK() macro
  • Index out of array bounds are ignored
  • Modern GPUs more resilient to bad code
  • Hardware reported exceptions cause host to reset GPU before testing next kernel.

• Errors implicitly lead to poor fitness: long run times or incorrect answers.
GP Evolution Parameters

- Pop 1000, 50 generations
- 50% crossover:
  - 25% uniform crossover on fixed parameters
  - 25% tree like two point crossover on variable length list of code patches
- 50% mutation
  - 25% change one fixed parameter (bit flip, BLOCK_W another legal value, either adjacent or random).
  - 25% add a random patch to variable list.
- Truncation selection
- $\approx 11$ hours
Best K20 Patch in gen 50

- Store btw cache in registers
- Use 2 threads to load bwt cache
- Double number of theads

<table>
<thead>
<tr>
<th>line</th>
<th>Original Code</th>
<th>New Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>635</td>
<td>#pragma unroll</td>
<td></td>
</tr>
<tr>
<td>578</td>
<td>if(k == bwt_cuda.seq_len)</td>
<td>if(0)</td>
</tr>
<tr>
<td>947</td>
<td>*(k0 = k;</td>
<td>(((int*)l0)[1] = __shfl(((int*)l0)[1], threads_per_sequence/2, threads_per_sequence);*k0 = k;</td>
</tr>
<tr>
<td>126</td>
<td>*lastpos=pos_shifted;</td>
<td></td>
</tr>
</tbody>
</table>

- Line 578 if was never true
- l0 is overwritten later regardless
- Line 126 disables small sequence cache 3% faster
Results

• Ten randomly chosen 100 base pair datasets from 1000 genomes project:
  – K20 1,840,000 DNA sequences/second
  – K40 2,330,000 DNA sequences/second

• 100% identical
• manually incorporated into release
• 352 downloads
GPUs

<table>
<thead>
<tr>
<th>GPU</th>
<th>Total cores</th>
<th>clock</th>
<th>Bandwidth Giga Bytes/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeForce GT 730</td>
<td>96</td>
<td>1.40 GHz</td>
<td>23</td>
</tr>
<tr>
<td>Tesla K20</td>
<td>2496</td>
<td>0.71 GHz</td>
<td>140</td>
</tr>
<tr>
<td>Tesla K40</td>
<td>2880</td>
<td>0.88 GHz</td>
<td>180</td>
</tr>
<tr>
<td>Tesla K80</td>
<td>2496</td>
<td>0.82 GHz</td>
<td>138</td>
</tr>
</tbody>
</table>

Tesla K80 is dual GPU. Figures given for one half.
DNA sequences per second

<table>
<thead>
<tr>
<th>Prog</th>
<th>Length</th>
<th>12 core 2.60GHz CPU</th>
<th>GT 730</th>
<th>2 K20</th>
<th>K80</th>
<th>GCAT accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWA</td>
<td>36bp</td>
<td>1900</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td>4500</td>
<td></td>
<td></td>
<td></td>
<td>98.91%</td>
</tr>
<tr>
<td>Old Barracuda</td>
<td>36bp</td>
<td>3270</td>
<td>5300</td>
<td>6500</td>
<td></td>
<td>97.49%</td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td>1860</td>
<td>8700</td>
<td>11700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Barracuda</td>
<td>36bp</td>
<td>7600</td>
<td>12900</td>
<td>19900</td>
<td></td>
<td>98.43%</td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td>2100</td>
<td>8800</td>
<td>1280</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Twin GPUs work on each of paired ends
- GT730 estimated as if two in use
## DNA sequences per second

<table>
<thead>
<tr>
<th>Program</th>
<th>Length</th>
<th>12 core 2.60GHz CPU</th>
<th>GT 730 £53.89</th>
<th>2 K20</th>
<th>K80</th>
<th>GCAT accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWA</td>
<td>36bp</td>
<td>1900</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td>4500</td>
<td></td>
<td></td>
<td></td>
<td>98.91%</td>
</tr>
<tr>
<td><strong>Old Barracuda</strong></td>
<td>36bp</td>
<td></td>
<td>1.7</td>
<td>2.8</td>
<td>3.4</td>
<td></td>
</tr>
<tr>
<td><strong>BWA</strong></td>
<td>100bp</td>
<td></td>
<td>0.4</td>
<td>1.9</td>
<td>2.6</td>
<td>97.49%</td>
</tr>
<tr>
<td><strong>New Barracuda</strong></td>
<td>36bp</td>
<td></td>
<td><strong>4.0</strong></td>
<td>6.8</td>
<td><strong>10.5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>BWA</strong></td>
<td>100bp</td>
<td></td>
<td>0.5</td>
<td>2.0</td>
<td>2.8</td>
<td>98.43%</td>
</tr>
<tr>
<td>GI Improvement (release code)</td>
<td>36bp</td>
<td></td>
<td>2.32</td>
<td>2.43</td>
<td>3.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td></td>
<td>1.13</td>
<td>1.00</td>
<td>1.09</td>
<td>1.6</td>
</tr>
</tbody>
</table>

- Twin GPUs work on each of paired ends
- GT730 estimated as if two in use
Each DNA end with dedicated GPU
Conclusions

• Compile into one executable
  – Scoping rules.
  – Run compiler until all remaining code compiles
• On real typical data raw speed up > 100 times
• Incorporated into real system
  • 1\textsuperscript{st} use of GI
• Impact diluted by rest of code
• On real data speed up can be >3 times
Conclusions

• Genetic programming can automatically re-engineer source code. E.g.
  – hash algorithm
  – Random numbers which take less power, etc.
  – mini-SAT (*Humie* award)
• fix bugs (>10^6 lines of code, 16 programs)
• create new code in a new environment (graphics card) for existing program, gzip [WCCI '10]
• new code to extend application ([GGGP] SSBSE'14 EuroGP'14 GECCO'14)
• speed up GPU image processing
• speed up 50000 lines of code
  10000 speed up [GI-2015]
GP Automatic Coding

• Use existing code as test “Oracle”. (Program is its own functional specification)
Scope

• Line can be copied where all its vars are in scope

• `<IF_Kkernel_bnf.cu_119>` line 109 to 168
  ```
  if (*lastpos!=pos_shifted)
  ```

• Line 99 `unsigned int * lastpos`,

• Line 109 `unsigned const int pos_shifted = ..`

• Line 168 `}` end of function read_char()
Comparisons

• Barracuda before and after GI
• BWA (12 cores)
• Bowtie2
• nvBowtie2
“Moore’s Law” in Sequences

Growth of GenBank

[Graph showing the growth of GenBank from 1982 to 2009, with a logarithmic y-axis in base pairs, from 100,000 to 10^11]
The Genetic Programming Bibliography

http://www.cs.bham.ac.uk/~wbl/biblio/

10291 references and 9879 online publications

RSS Support available through the Collection of CS Bibliographies.

A web form for adding your entries. Co-authorship community. Downloads

A personalised list of every author’s GP publications.

blog.html

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