1000 Downloads of Genetically Improved DNA Analysis Software

CREST Open Workshop on Genetic Improvement 25-26 January 2016

W. B. Langdon
Computer Science, University College London

Special Session on Genetic Improvement

Based on GECCO 2015 p1063-1070
1000 Downloads of Genetically Improved DNA Analysis Software

W. B. Langdon
Computer Science, University College London

Special Session on Genetic Improvement

Based on GECCO 2015 p1063-1070
Genetically Improved BarraCUDA

• **Background**
  – What is BarraCUDA
  – Using GP to improve parallel software, i.e. BarraCUDA

• **Results**
  – 100× speedup
  – GCAT benchmark (arXiv.org)
  – demonstrate 1st GI in use.
    • 1068 sourceforge downloads (10 months).
    • Commercial use by Lab7 (in BioBuilds Nov2015) and IBM Power8
What is BarraCUDA?

DNA analysis program
- 8000 lines C code, SourceForge.
- Rewrite of BWA for nVidia CUDA

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

Diagram:
- Human genome
- Barracuda
- location(s) in human genome
- tens of millions of short DNA sequences
- 16MBytes
- 159744 DNA bases
BarraCUDA 0.7.107b

Manual host changes to call exact_match kernel
GI parameter and code changes on GPU

Human genome

Host 8000 lines C
GPU 696 lines CUDA

prefix tree
4GBytes

159744 strings
unique match?

process parts of
noisy/inexact
strings

exact match
split_inexact
match_caller

common
code
200 lines CUDA

85%
15%

stdout
location(s) in human genome
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)
• Genetic programming 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
## Fixed Parameters

<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
- Genetic Programming continues despite compilation and runtime errors
Evolving BarraCUDA

50 generations in 11 hours

W. B. Langdon, UCL
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 (e.g. 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
Representation

• 15 fixed parameters; variable length list of grammar patches.
  • no size limit, so search space is infinite
• Uniform crossover and tree like 2pt crossover.
• Mutation flips one bit/int or 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

Copy of line 929
\[((int*)l0)[1] = __shfl(((int*)&l)[1],threads_per_sequence/2,threads_per_sequence);
\]
\[*k0 = k;\]

New code

Line 947
Recap

• Representation
  – 15 fixed genes (mix of Boolean and integer)
  – List of changes (delete, replace, insert).
    New rule must be of same type.

• Mutation
  – 1 bit flip or small/large change to int

• append one random change to code

• Crossover
  – Uniform GA crossover
  – GP tree like 2pt crossover

• Evolve for 50 generations
Best K20 GPU Patch in gen 50

<table>
<thead>
<tr>
<th>Parameter</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>scache_global_bwt</td>
<td>on</td>
</tr>
<tr>
<td>cache_threads</td>
<td>2</td>
</tr>
<tr>
<td>BLOCK_W</td>
<td>128</td>
</tr>
</tbody>
</table>

Store bwt cache in registers

Use 2 threads to load bwt cache

Double number of threads

<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*)l)[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

10 is overwritten later regardless

Change 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 (original 15000)
  – K40 2 330 000 DNA sequences/second (original 16 000)
• 100% identical
• manually incorporated into sourceForge
• 1068 downloads (10 months)
GI: To Do List

• Systems
  – GenProg

• Wikipedia

• Bibliography?

• GI workshop (Denver), GI@CEC (Vancouver)

• Other resources: www, email, discussion???

• How to do Genetic Improvement
  – Documentation
  – Tutorials
  – Little examples. Real benchmarks
Conclusions

• Genetic programming
  – Compile into one executable
  – Scoping rules
  – Run compiler until all remaining code compiles
  – Fitness test representative data v. existing code

• On real typical data raw speed up > 100 times

• Impact diluted by rest of code

• On real data speed up can be >3 times (arXiv.org)

• Incorporated into real system
  • 1st use of genetic improvement
Special Session on Genetic Improvement

Humies: Human-Competitive
Cash prizes
GECCO-2016
Genetic Improvement

W. B. Langdon

CREST

Department of Computer Science
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

• new code to extend application (GGGP)

• speed up GPU image processing

• speed up 50000 lines of code

  10000 speed up GI-2015
Compile Whole Population

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

Note Log x scale
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
  - Macro `#define` holds value of parameter
  - Macro used in code, e.g. via conditional compilation
  - Cleared with `#undef` before next mutant is compiled
Example2 Mutating Grammar

\[_{\text{Kkernel.bnf.cu.126}} := "{lastpos=\text{pos_shifted};}"

1 line from grammar

\[_{\text{126}}\]

Fragment of list of mutations
Says delete line 126
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 and then 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
• ≈11 hours
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
<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></td>
</tr>
<tr>
<td>Old Barracuda BWA</td>
<td>36bp</td>
<td></td>
<td>1.7</td>
<td>2.8</td>
<td>3.4</td>
<td>98.91%</td>
</tr>
<tr>
<td></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>New Barracuda BWA</td>
<td>36bp</td>
<td></td>
<td>4.0</td>
<td>6.8</td>
<td>10.5</td>
<td>98.43%</td>
</tr>
<tr>
<td></td>
<td>100bp</td>
<td></td>
<td>0.5</td>
<td>2.0</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>GI Improvement</td>
<td>36bp</td>
<td></td>
<td>2.32</td>
<td>2.43</td>
<td>3.07</td>
<td></td>
</tr>
<tr>
<td>(release code)</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
“Moore’s Law” in Sequences

Growth of GenBank

base pairs

1e+11
1e+10
1e+9
1e+8
1e+7
1e+6
100000

The Genetic Programming Bibliography

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

10617 references

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

Google scholar citations

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