CREST

Genetically Improved BarraCUDA

GECCO 2015

W. B. Langdon
Computer Science, University College London

9.7.2015

CREST

Genetically Improved BarraCUDA

W. B. Langdon

Computer Science, University College London

UCL

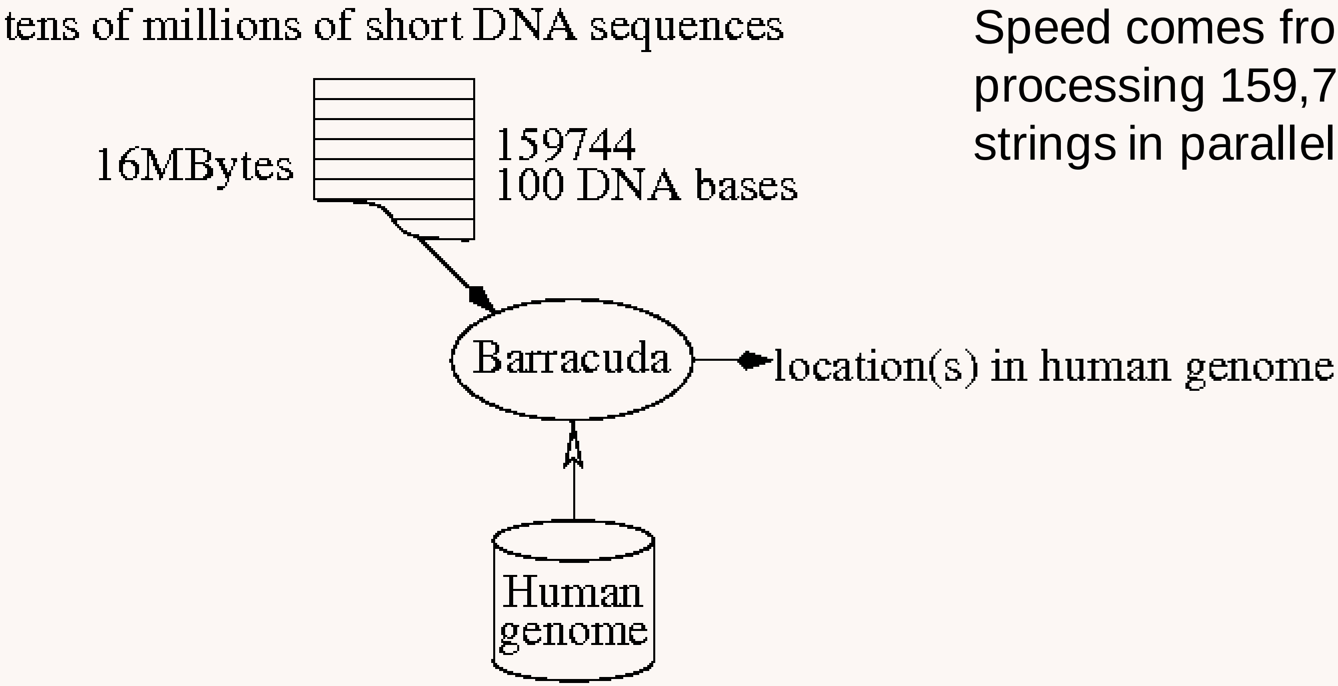
Genetically Improved BarraCUDA

- Background
  - – What is BarraCUDA
  - – Using GP to improve parallel software, i.e. BarraCUDA
- Results
  - – 100× Speedup
  - – GCAT benchmark, demonstrate 1st GI in use (arXiv.org)

What is BarraCUDA ?

DNA analysis program

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

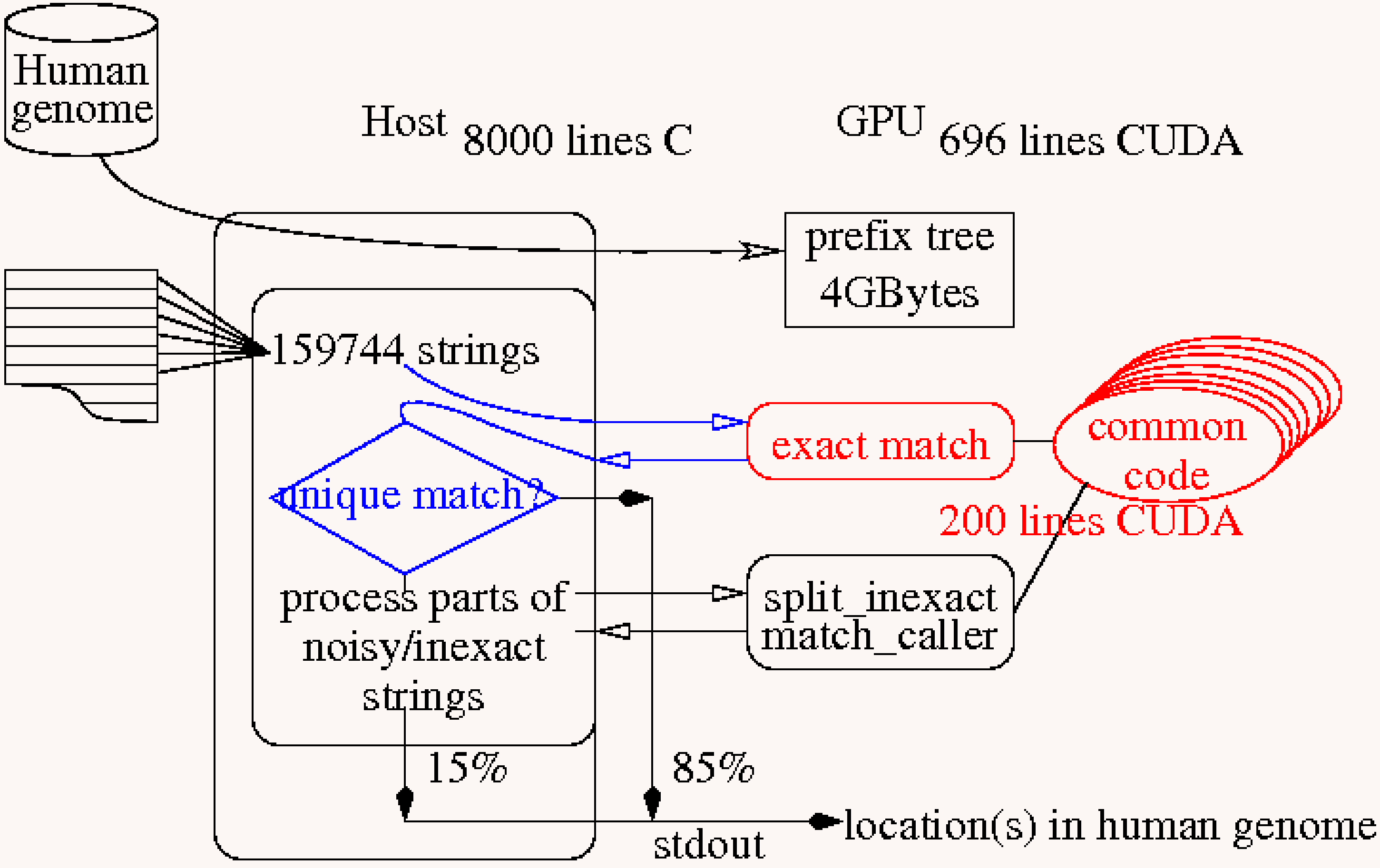

tens of millions of short DNA sequences
16MBytes
159744
100 DNA bases
Barracuda
location(s) in human genome
Human
genome

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

BarraCUDA 0.7.107a

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

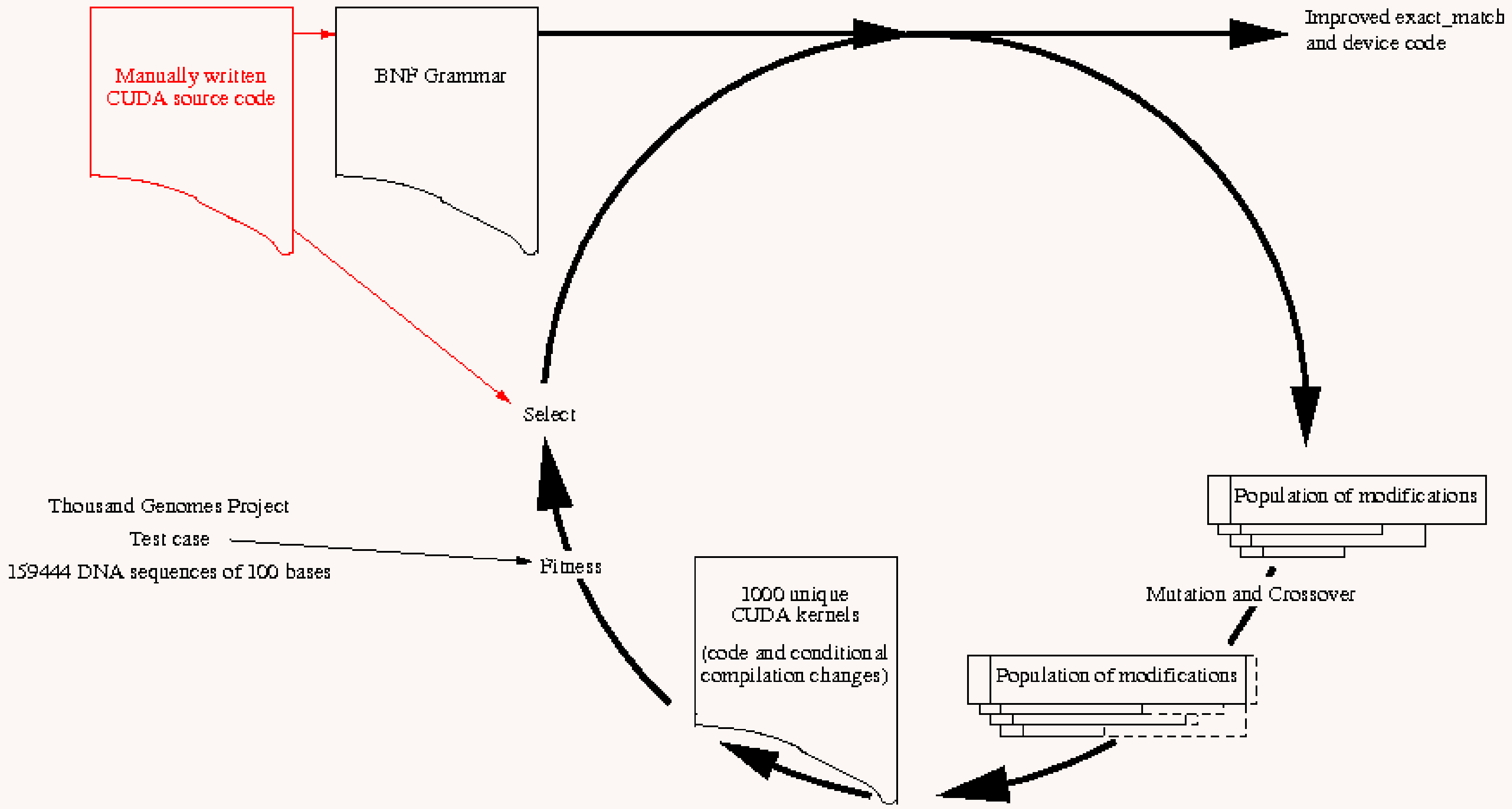
- Re-enable exact matches code
- Support 15 options(conditional compilation)
- GP fitness testing framework
  - – Generate and compile 1000 unique mutants
    - Whole population in one source file
    - Remove mutants who fail to compile and then re-run compiler to compile the others
  - – Run and measure speed of 1000 kernels
    - Reset GPU following run time errors
  - – For each kernel check 159444 answers

Parameter		default	Lines of code affected
BLOCK_W	int	64	all
cache_threads	“” int	“”	44
kl_par	binary	off	19
occ_par	binary	off	76
many_blocks	binary	off	2
direct_sequence	binary	on	63
direct_index	binary	on	6
sequence_global	binary	on	16
sequence_shift81	binary	on	30
sequence_stride	binary	on	14
mycache4	binary	on	12
mycache2	binary	off	11
direct_global_bwt	binary	off	2
cache_global_bwt	binary	on	65
scache_global_bwt	binary	off	35

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

Evolving BarraCUDA


Manually written CUDA source code
BNF Grammar
Improved exact_match and device code
Select
Thousand Genomes Project
Test case
159444 DNA sequences of 100 bases
Fitness
1000 unique CUDA kernels
(code and conditional compilation changes)
Population of modifications
Mutation and Crossover
Population of modifications

51 gens in 11 hours

BNF Grammar

Configuration parameter

```
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
- 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

New code

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

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.
    - no size limit, so search space is infinite
- 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

Best K20 GPU Patch in gen 50

		new
scache_global_bwt	off	on
cache_threads	off	2
BLOCK_W	64	128

Store bwt cache in registers

Use 2 threads to load bwt cache

Double number of threads

line	Original Code	New Code
635		#pragma unroll
578	if(k == bwt_cuda.seq_len)	if(0)
947	`*k0 = k;`	`((int*)l0)[1] = __shfl(((int*)&l)[1],threads_per_sequence/2,threads_per_sequence);*k0 = k;`
126	`*lastpos=pos_shifted;`	

Line 578 `if` was never true

`l0` 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
- 378 downloads

Conclusions

- GP
  - – 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

END

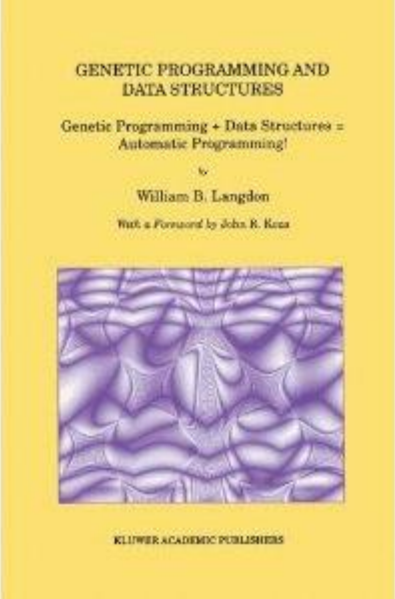
http://www.cs.ucl.ac.uk/staff/W.Langdon/ http://www.epsrc.ac.uk/ EPSRC

Genetic Improvement

W. B. Langdon

CREST

Department of Computer Science



GENETIC PROGRAMMING AND DATA STRUCTURES

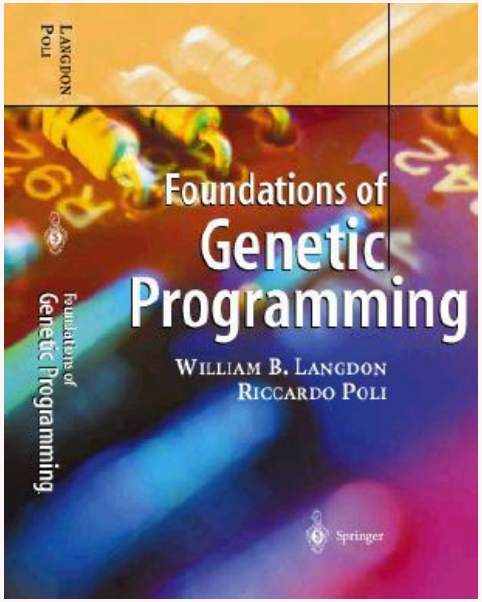
Genetic Programming + Data Structures = Automatic Programming!

by

William B. Langdon

With a Foreword by John R. Koza

KLUWER ACADEMIC PUBLISHERS



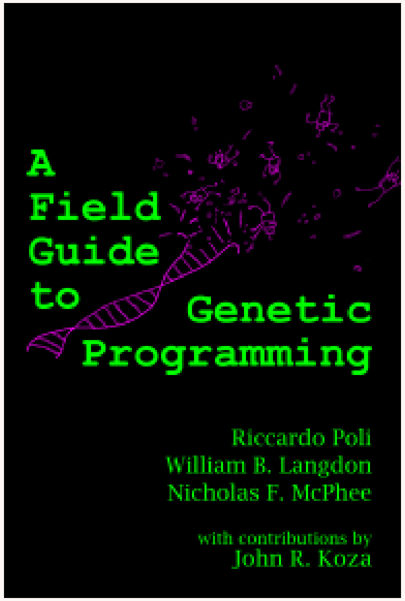
LANGDON POLI

Foundations of Genetic Programming

WILLIAM B. LANGDON
RICCARDO POLI

Foundations of Genetic Programming

Springer



A Field Guide to Genetic Programming

Riccardo Poli
William B. Langdon
Nicholas F. McPhee

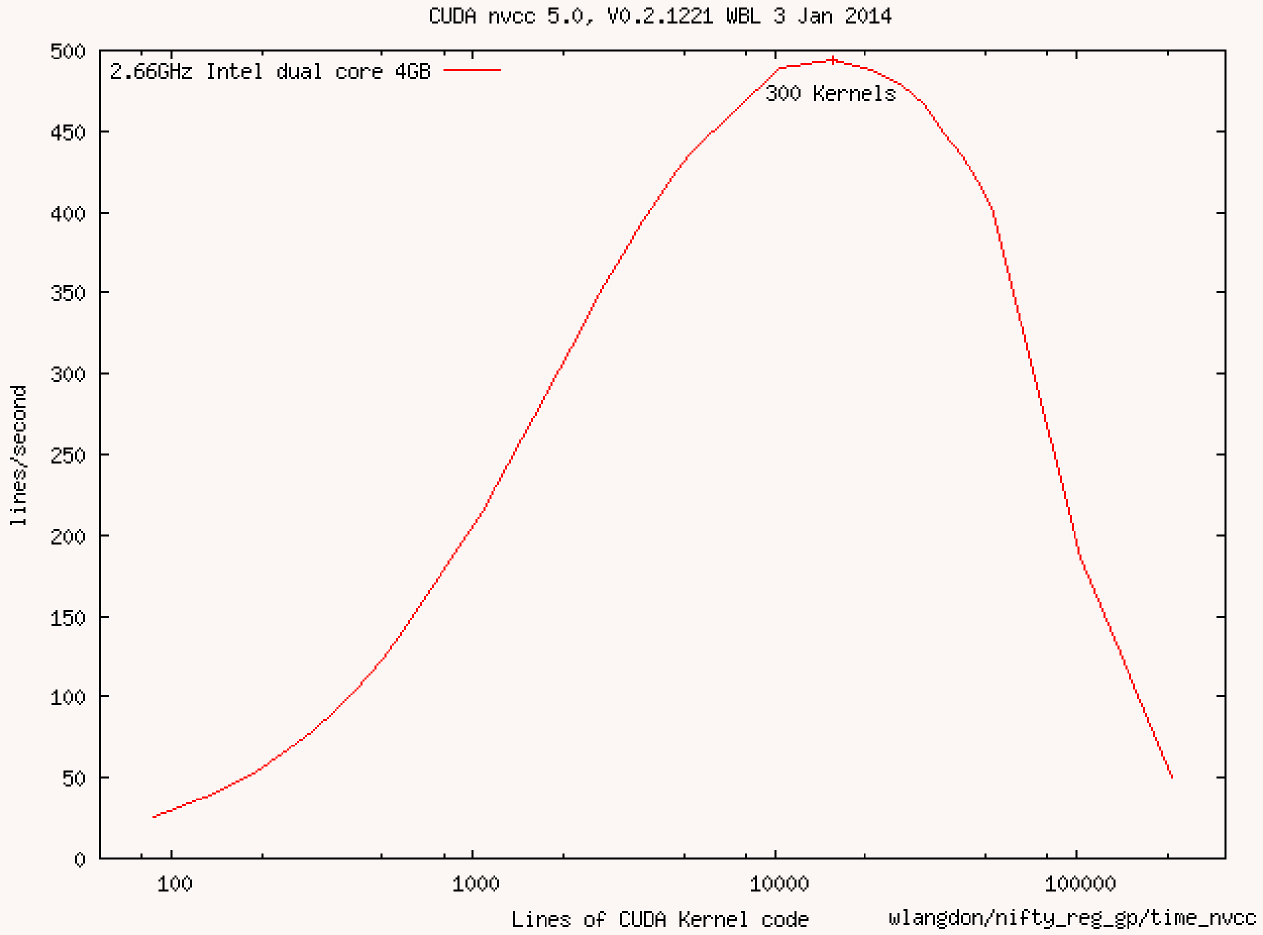
with contributions by
John R. Koza

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
- speed up GPU image processing EuroGP'14 GECCO'14
- speed up 50000 lines of code IEEE TEC

10000 speed up GI-2015

Compile Whole Population



Note Log x scale

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

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

```
<_Kkernel_bnf.cu_126> ::= "*lastpos=pos_shifted;"
```

1 line from grammar

```
<_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

GPUs

GPU	Total cores	clock	Bandwidth Giga Bytes/sec
GeForce GT 730	96	1.40 GHz	23
Tesla K20	2496	0.71 GHz	140
Tesla K40	2880	0.88 GHz	180
Tesla K80	2496	0.82 GHz	138

Tesla K80 is dual GPU. Figures given for one half.

DNA sequences per second

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

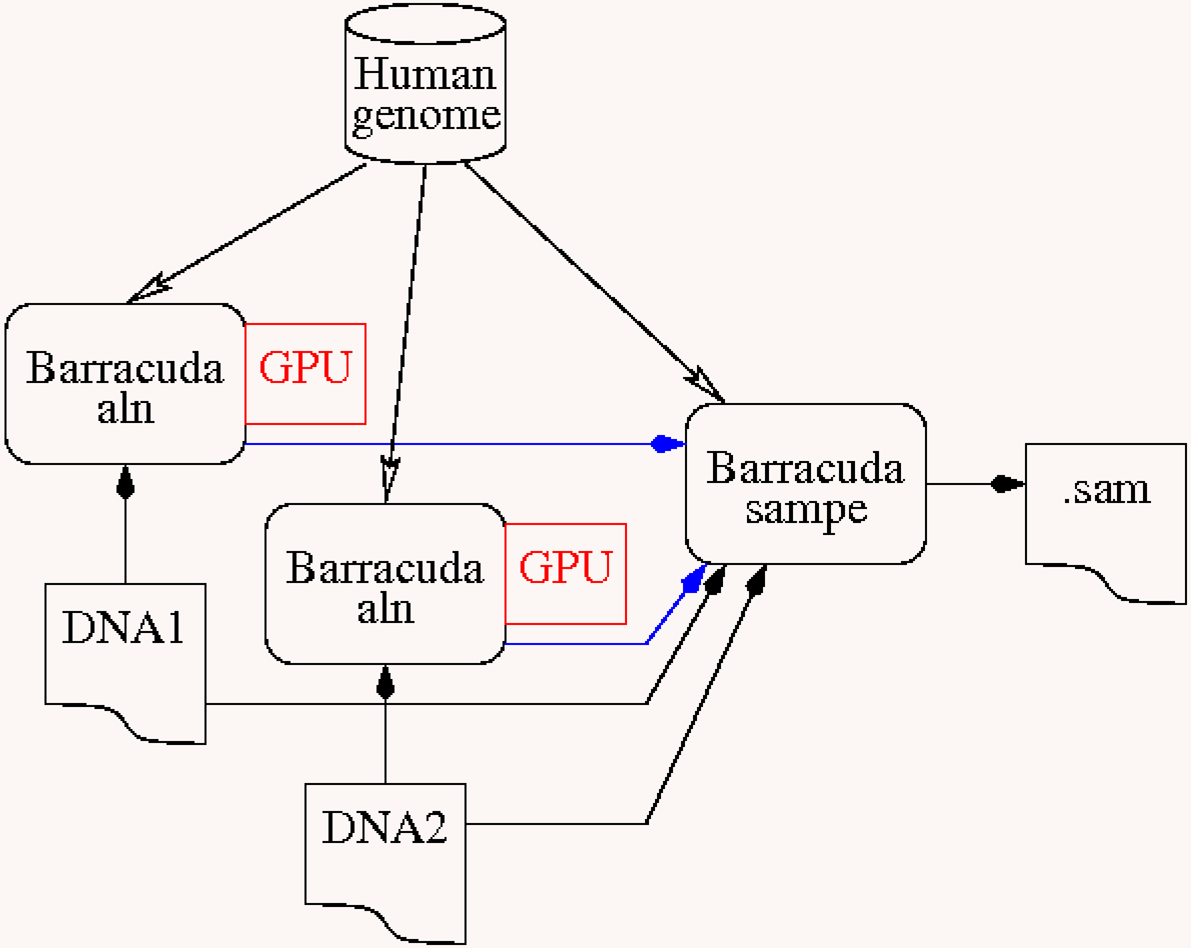
- Twin GPUs work on each of paired ends
- GT730 estimated as if two in use

DNA sequences per second

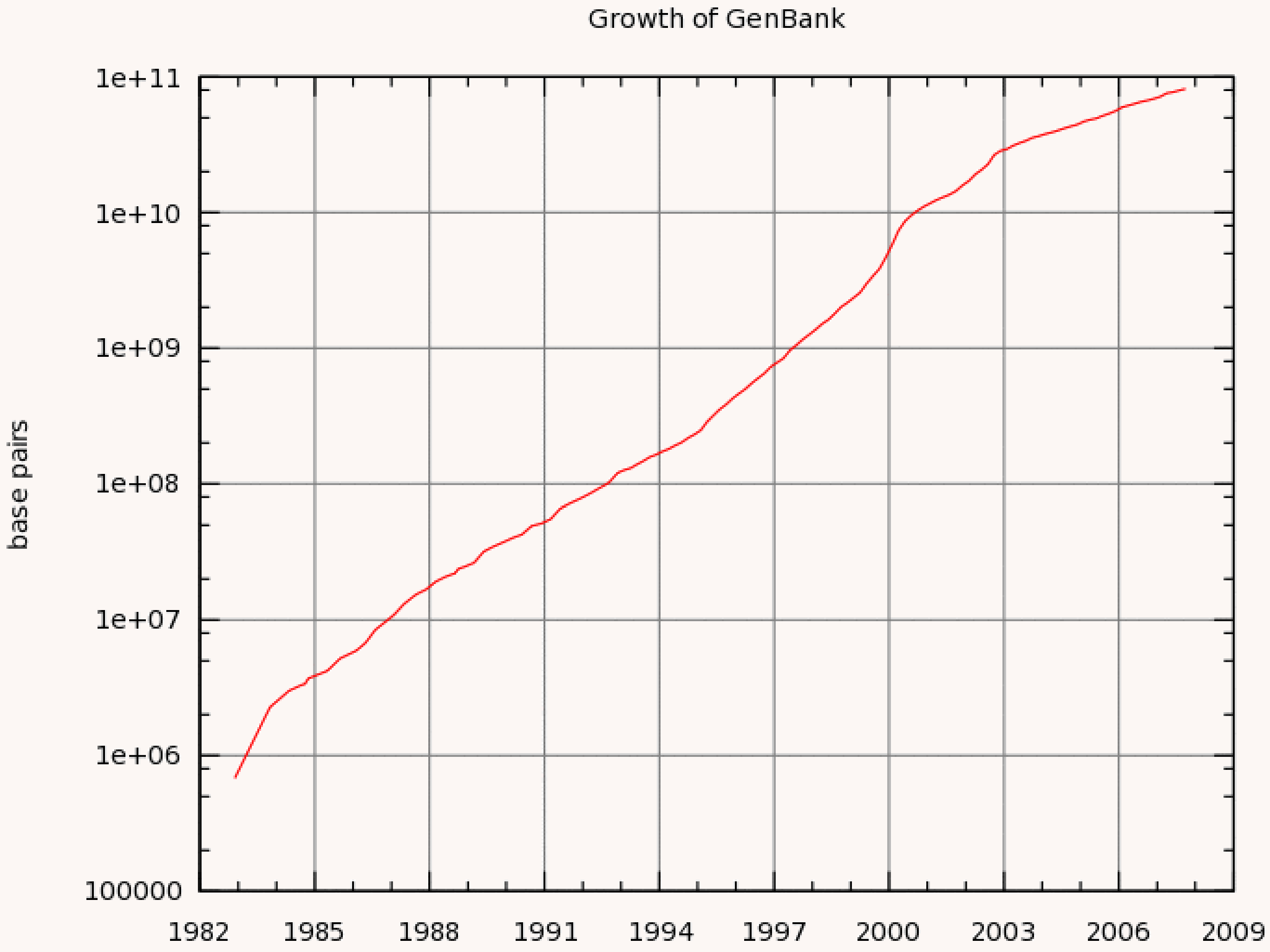
Program	Length	12 core 2.60GHz CPU	GT 730 £53.89	2 K20	K80	GCAT accuracy
BWA	36bp	1900				
	100bp	4500				98.91%
Old Barracuda BWA	36bp		1.7	2.8	3.4	
	100bp		0.4	1.9	2.6	97.49%
New Barracuda BWA	36bp		4.0	6.8	10.5	
	100bp		0.5	2.0	2.8	98.43%
GI Improvement (release code)	36bp		2.32	2.43	3.07	
	100bp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

- 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

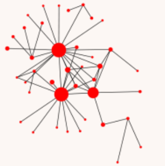


The Genetic Programming Bibliography

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

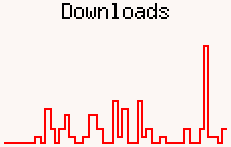
10318 references

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



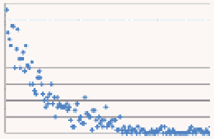
Part of gp-bibliography 84 40 Revision:1.1794 29 May 2011

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