Using evolutionary computing to optimise BarraCUDA

UKMAC 2016

W. B. Langdon
Computer Science, University College London

Genetically Improved BarraCUDA

- Background
  - What is BarraCUDA
  - Using Genetic Programming to improve parallel software, i.e. BarraCUDA
- Results
  - 100× Speedup
  - GCAT bioinformatics benchmark (arXiv.org)

Why? NextGen DNA sequences

- Goal (idealised): read all of patient’s DNA.
  - How does it differ from other people’s DNA?
  - Do genetic differences (e.g. SNPs) explain diseases, predict outcomes, aid treatments?
- Next generation DNA scanners give short noisy strings. So read genome many times (3 to 30).
- Find best match between DNA string and reference human genome.
- Assemble patient’s genome from billion matches
- Most differences between string and reference human genome are measurement noise

What is BarraCUDA ?

- CUDA program to align millions of short noisy DNA strings to a reference genome.
- CUDA port of existing BWA alignment tool
- 8000 lines C source code, SourceForge

What is BarraCUDA ?

- BWA port published as:

  Petr Klus, Simon Lam, Dag Lyberg, Ming Sin Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo, Brian YH Lam. (2012) BarraCUDA... BMC Res Notes [PMID: 22244497]

  - bioinformatics code/test, GPU

- BarraCUDA presented at 3rd UK GPU 2011
- **Improving CUDA DNA Analysis Software with Genetic Programming**, W.B. Langdon *et al*., GECCO 2015.
- Download barracuda_0.7.107 sourceForge

Burrows-Wheeler Transform

- Store whole human genome (3 10^9 bases) as prefix tree. (Index built offline once)
- Can locate all places in human genome which match DNA read exactly.
- Index is compressed. Index < 4GBytes
- Fast O(length of read)
- Online. Can search in either direction, from any point in string.
- Extend to partial matches by back-tracking

BWT Partial Matches: Tree Search Heuristic

- Search forward until either reach end or there are no exact matches.
- Assume lack of match is because of recent error and back up one base.
- Try in series all the possible changes at that base. If match, continue forward
- If none of them exist in the human genome, back up one more

Problems with Tree Search

- Forward search
  - 159,744 threads process one search each
  - In principle each base needs 2 reads of BTW index in global memory
  - Thread access to BWT index unrelated
- Back tracking
  - When thread starts back tracking depends on its data. I.e. unrelated to others in same warp. Threads diverge.
  - Push lots of bytes onto stack in local memory

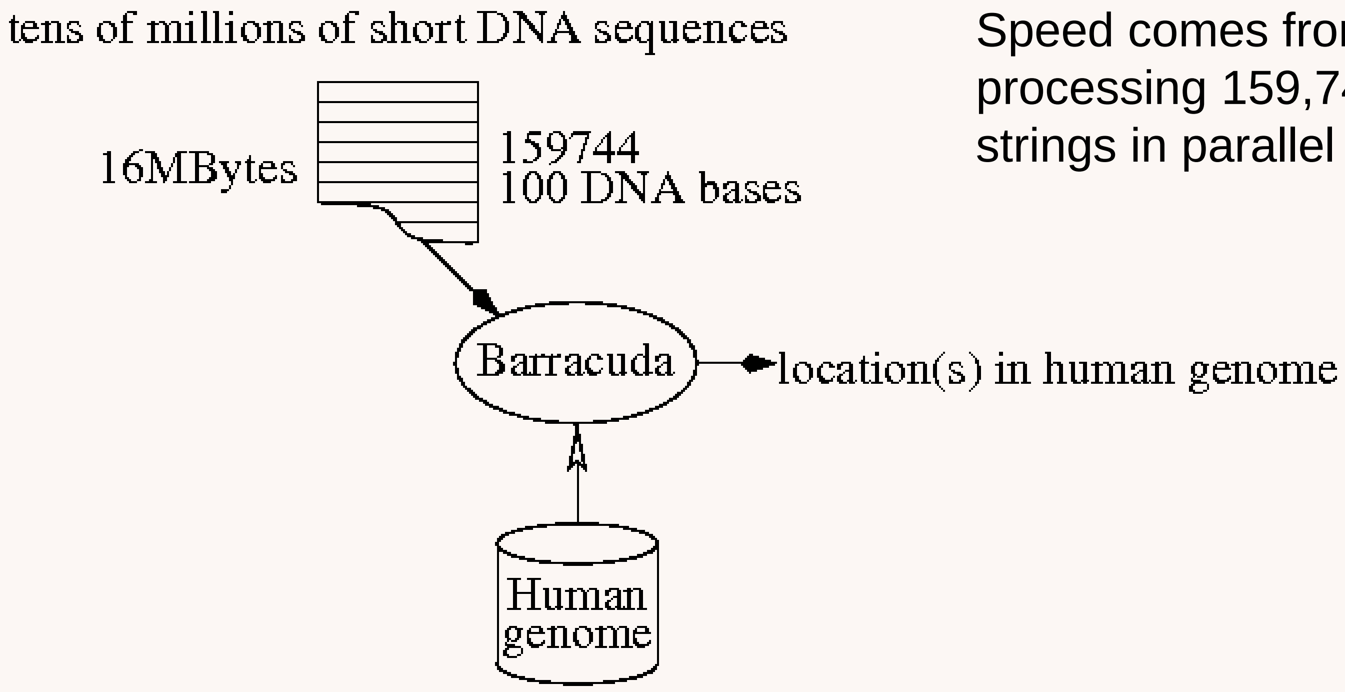
Avoid Tree Search

- In typical data only 15% need tree search
  - 99.45% of warps will diverge
- Forward search only
  - 99.45% of warps one thread stops early but rest continue
- Only 15% use back tracking kernel.

How does BarraCUDA work?

Given highly redundant set of short strings, re-assemble them into complete genome

Where did each fragment of DNA come from in the human genome?


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

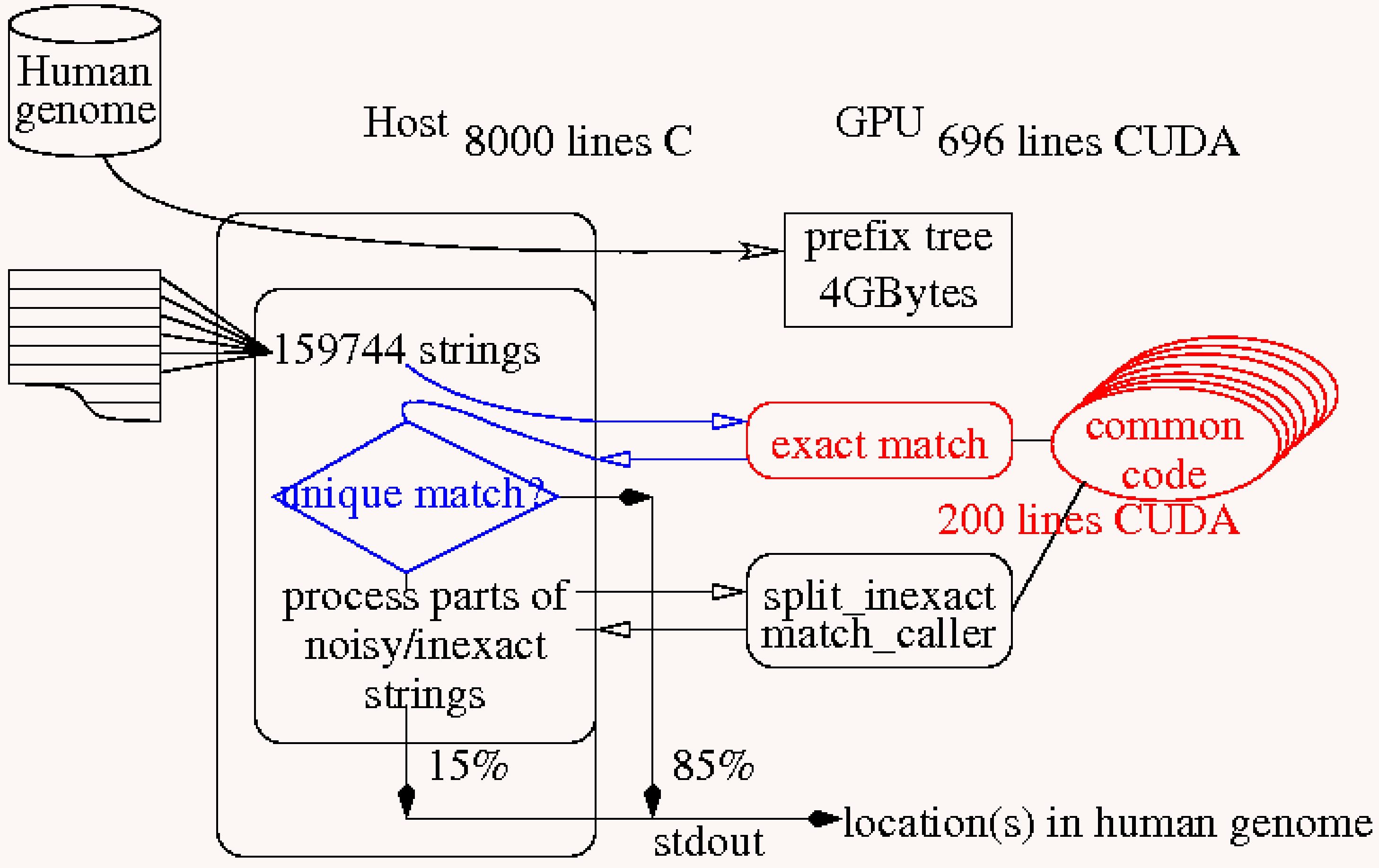
Speed comes from processing 159,744 strings in parallel on GPU

CREST

BarraCUDA 0.7.107

Manual host changes to call exact_match kernel

GP parameter and code changes on GPU



Before Automatic Optimisation

- Re-enable exact matches code
- Manual coding to support 15 options. E.g.
  - configurable cache for BWT index
  - texture or global memory

Configuration parameter

```
#ifndef sequence_global
  *data = tmp = tex1Dfetch(sequences_array, pos_shifted);
#else
  *data = tmp = Global_sequences(global_sequences,pos_shifted);
#endif /*sequence_global*/
```

CUDA lines 121-125

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

Evolutionary Framework

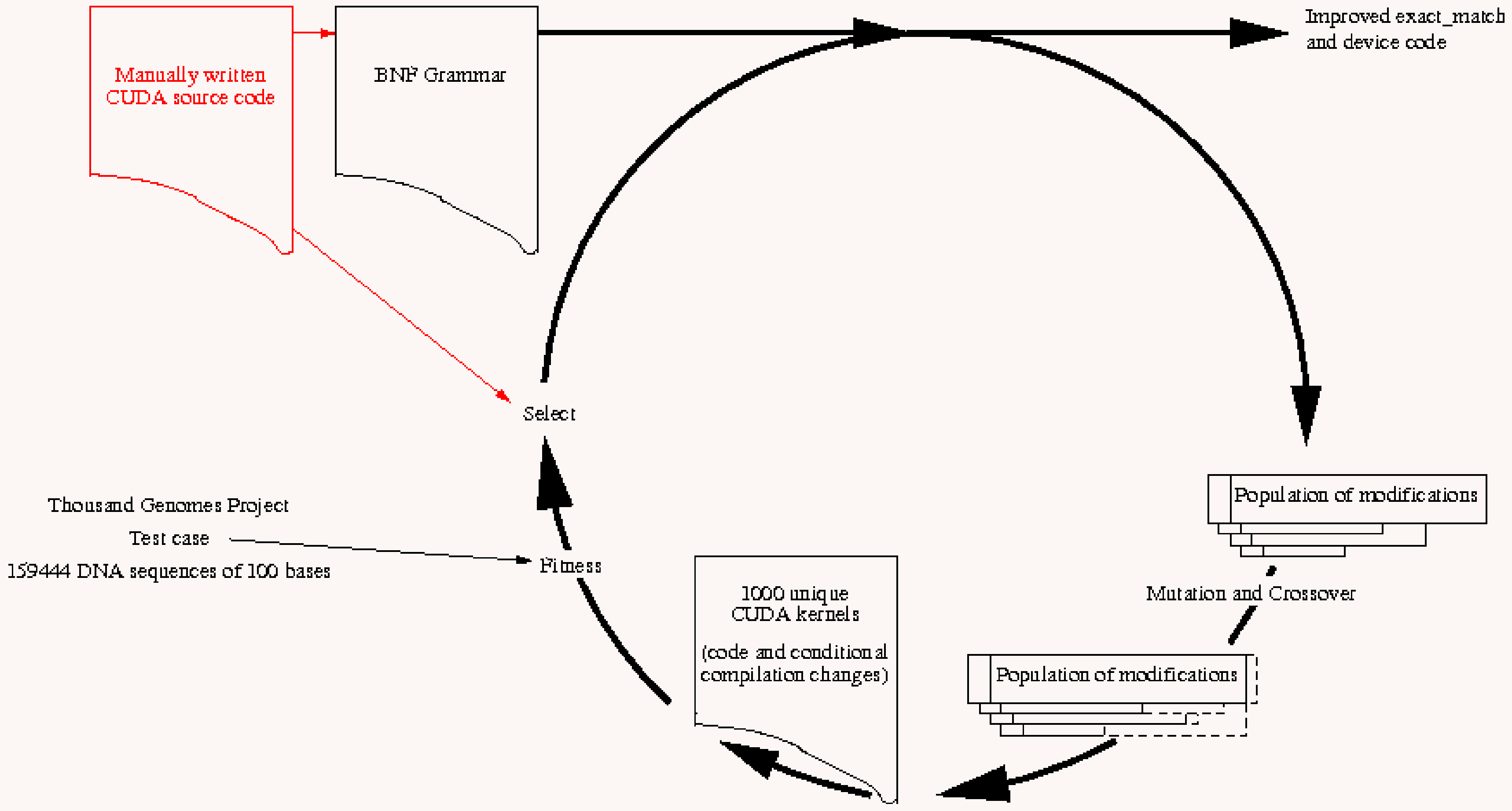
- GP fitness testing framework
  - – Generate and compile 1000 unique mutants
  - – Run and measure speed of 1000 kernels
    - Reset GPU following run time errors
  - – For each kernel check 159444 answers

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

CREST

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

Representing code changes

- 15 fixed parameters; variable length list of grammar patches.
- uniform crossover; two point 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 crossover on parameters changes
  - Two point crossover on code changes

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 (1,546 downloads)

CREST

General Lessons

- CUDA programming remains hard
- Tune block size, -arch, etc. automatically
  - not by theory or thinking hard.
- Best data storage may be GPU dependent
- Leave design choices (e.g. data location) to automatic per-GPU optimiser.
  - 1 para: try all values.
  - n parameters gives p^n explosion: Assuming they interact try genetic programming

Conclusions

- Evolving code
  - We looked at many changes
  - Pragmatically tuning 15 parameters give big payback
- 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 BarraCUDA