CREST

Evolving Faster Nifty Reg 3D Medical Image Registration CUDA kernels

W. B. Langdon

Centre for Research on Evolution, Search and Testing

Computer Science, UCL, London

GISMOE: Genetic Improvement of Software for Multiple Objectives

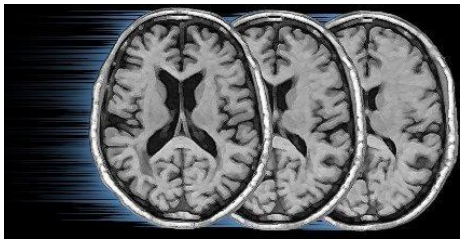
20 May 2014

CREST Annual Review
Genetic Improvement

- 2 Keynotes, 5 seminars, GECCO GP track
- Genetic Improvement (GIP) in CREST:
  - WBL,Shin, Justyna, Yue, Fan, Iman, Federica
- StereoCamera [EuroGP-2014]
- NiftyReg [GECCO-2014]

- Babel Pidgin (Yue) autocreate/merge countdown and bi-translate [SSBSE challenge]

Evolving Faster Nifty Reg 3D Medical Image Registration CUDA kernels



- What is NiftyReg?
  - UCL CMIC M.Modat sourceForge 16000 C++
- 3D Medical Images
  - Magnetic Resonance Imaging (MRI) brain scans 1mm resolution → 217^3=10,218,313 voxels
- Registration: NiftyReg nonlinear alignment of 3D images
- Graphics GPU parallel hardware
- CUDA allows C++ functions (kernels) to run in parallel

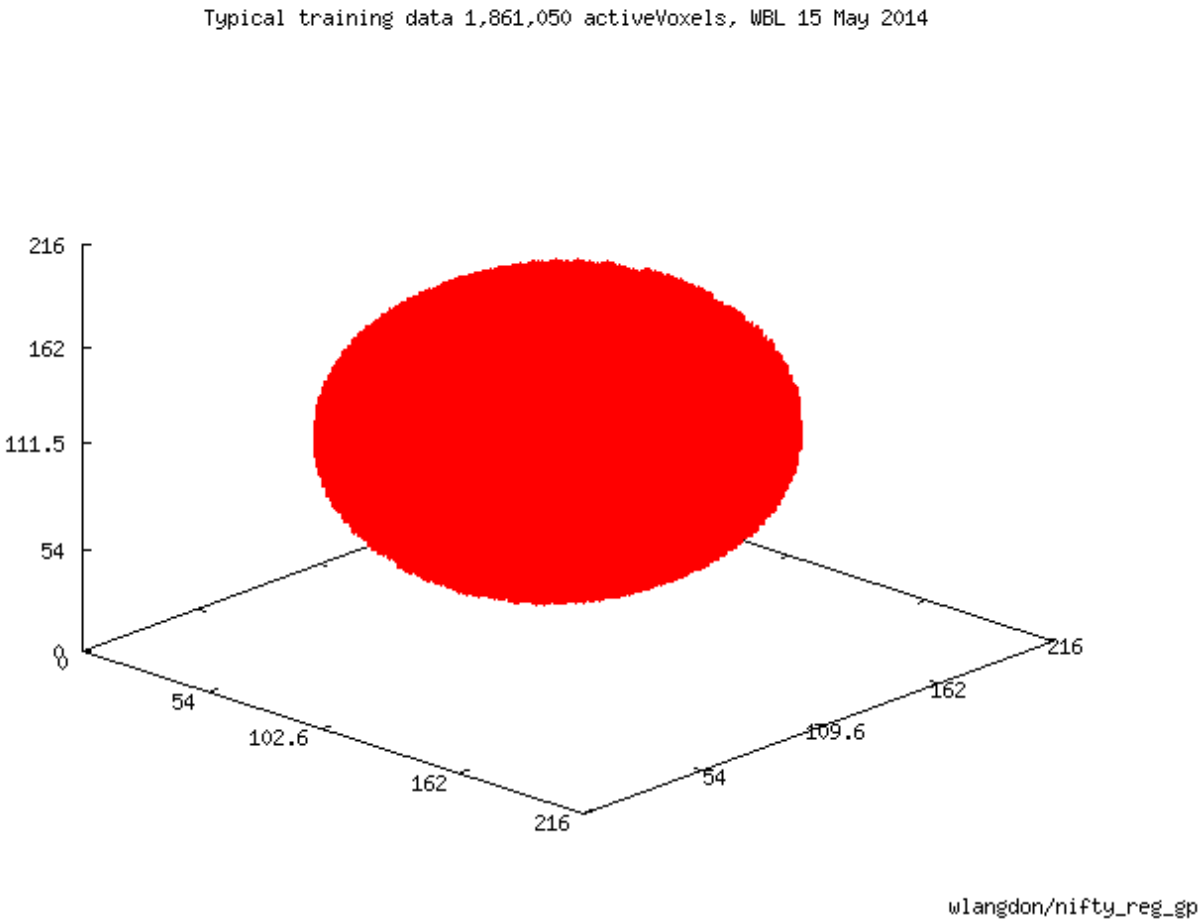
Nifty Reg

- Graphics hardware “ideal” for processing 2 and 3 dimensional images.
- NiftyReg partially converted to run in parallel on GPUs.
- Only part? Cluster easier to code(?) but not real-time.
- Aim to show GP can help with conversion of remainder or improvement of kernels.
- reg_bspline_getDeformationField() 97lines

reg_bspline_getDeformationField3D

- Chosen as used many times (≈100,000)
  70% GPU (GTX 295) time
- Need for accurate answers (stable derivatives).
- Active region (Brain) occupies only fraction of cube. List of active voxels.
- Kernel interpolates (using splines) displacement at each voxel from neighbouring control points.

Typical Active Part of Image

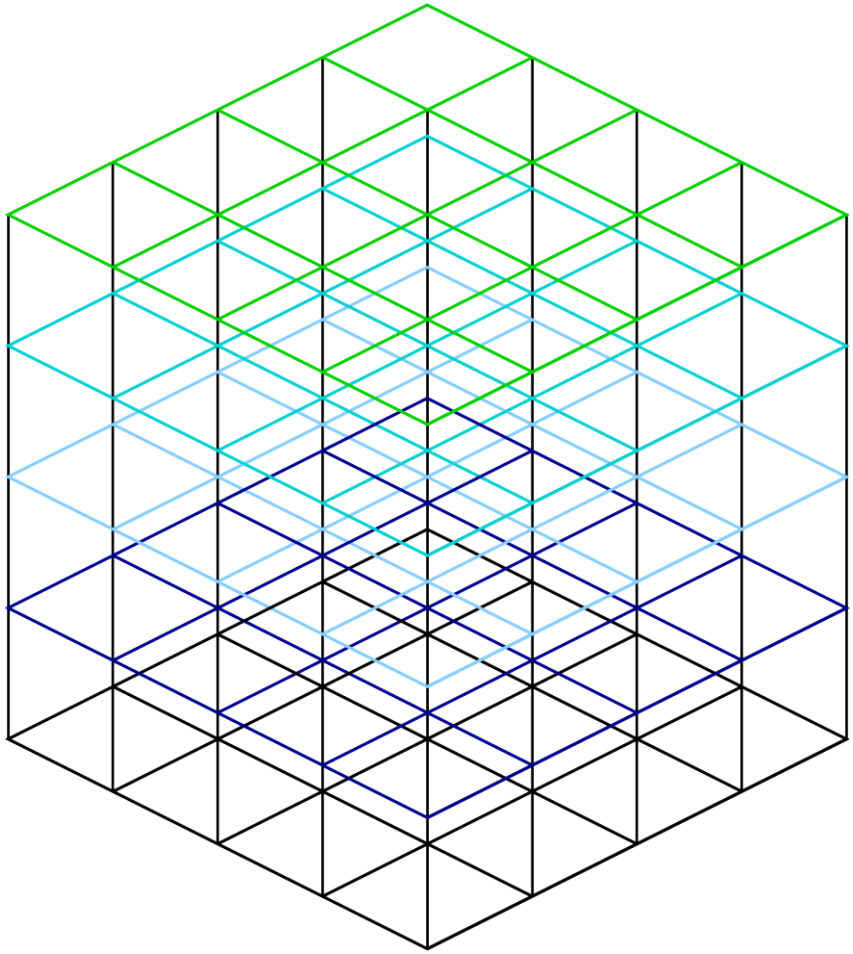


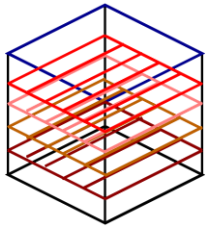
spline interpolation between 4×4×4 neighbours

Control points every 5th data point.

47^3=103,823 control points

All 5^3=125 data points in each control cube have same control point neighbours





reg_bspline_getDeformationField3D

- For each active voxel (≈10^6)
  - Calculate its x,y,z displacement by non-linear B spline (cubic) interpolation from 64 neighbouring control points
- Approx 600 flops per voxel.
  - Re-use limited by register/shared memory.
- Read voxel list and control points displacement from global memory (via textures)
- Write answer δx,δy,δz to global memory

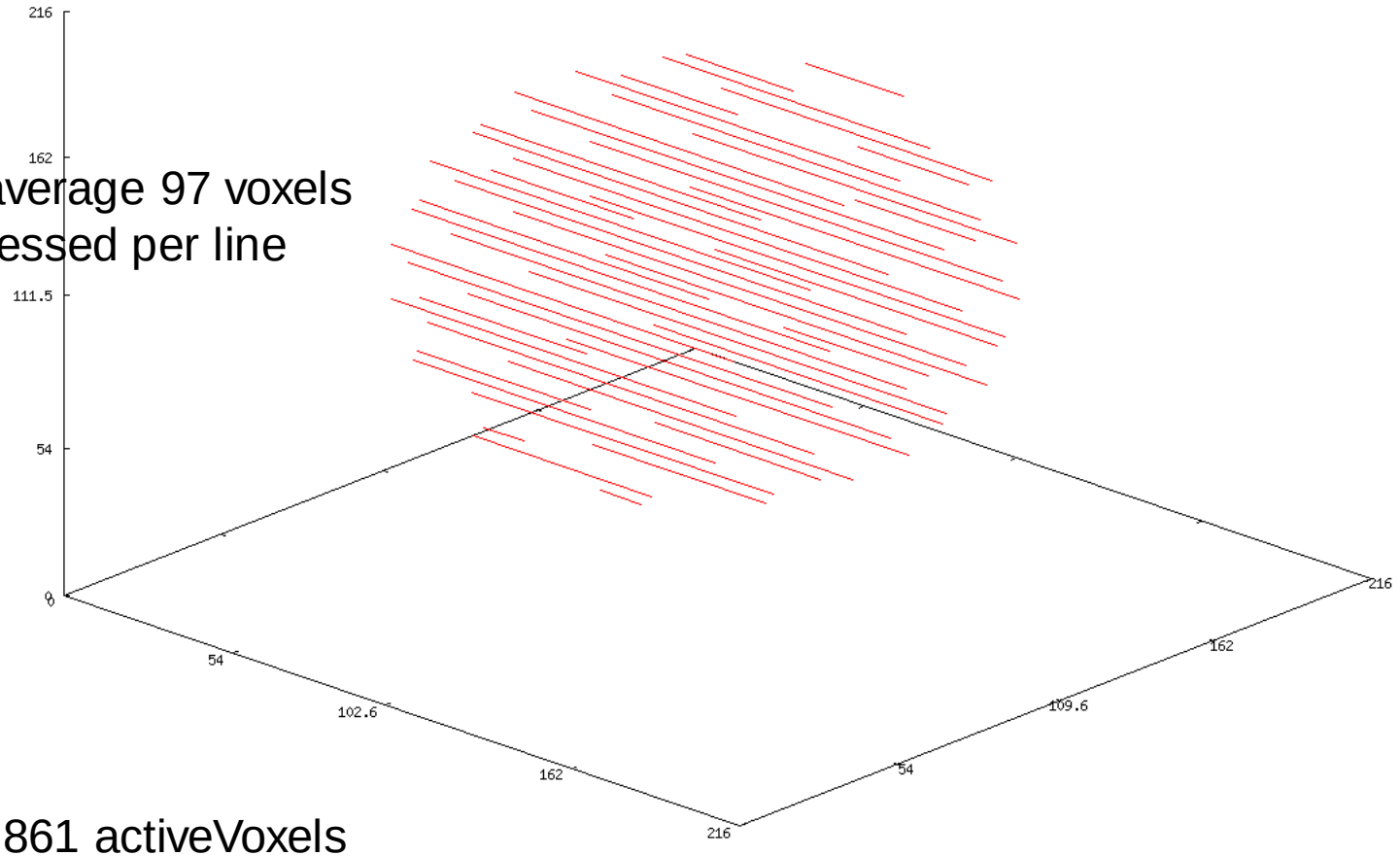
Improve Kernel

- Confusion with 1st example…
- Fixed control grid spline coefficients (20) need be calculate once and then stored.
  - Leave to GP how to store
- Huge reduction in computation. So kernel faster but now I/O bound.
  - Process 25 active voxels which share same control points together.

Original Kernel

Voxels processed in x-order
so caches may reload at
end of line

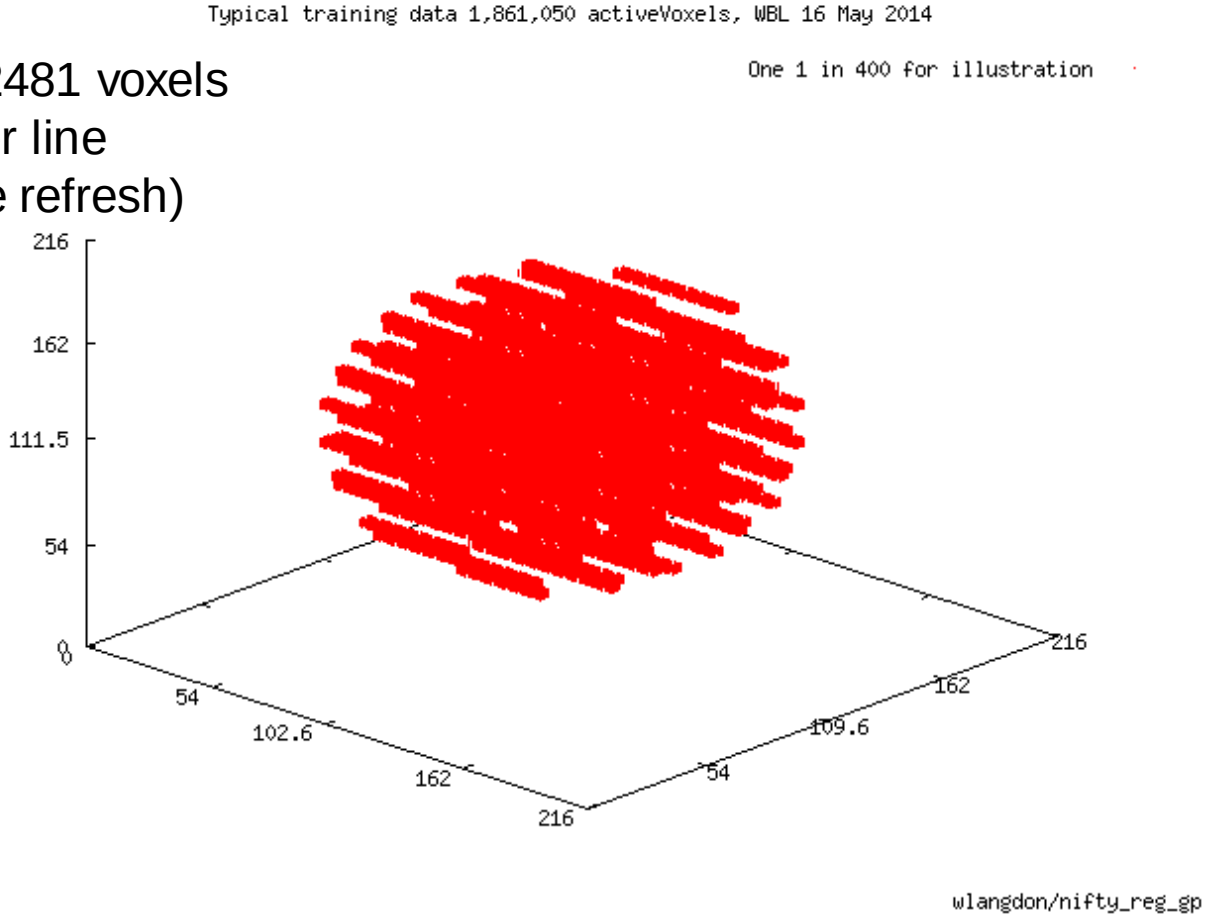
On average 97 voxels
processed per line



1,718,861 activeVoxels
To reduce clutter only one 1 in 400 plotted

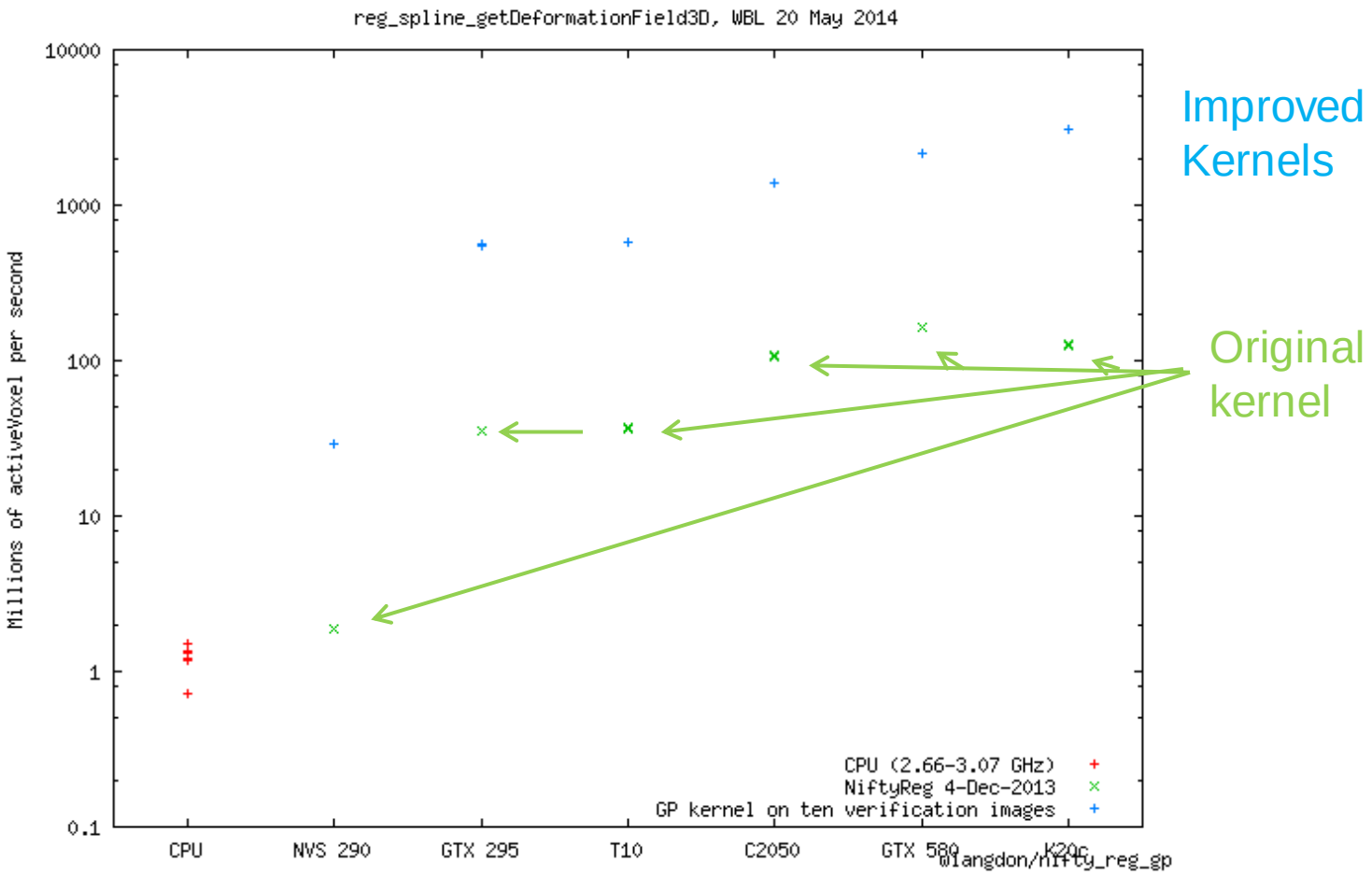
Improved kernel

On average 2481 voxels processed per line (before cache refresh)



1,861,050 activeVoxels
To reduce clutter only one 1 in 400 plotted

CPU v GPU


reg_spline_getDeformationField3D, WBL 20 May 2014
Millions of activeVoxel per second
10000
1000
100
10
1
0.1
CPU
NVS 290
GTX 295
T10
C2050
GTX 580
K20c
CPU (2.66-3.07 GHz)
NiftyReg 4-Dec-2013
GP kernel on ten verification images
wlangdon/nifty_reg_gp
Improved Kernels
Original kernel

GP Automatic Coding

- Target open source system in use and being actively updated at UCL. Hope for take up by developers
- Chose NiftyReg
- GPU already give 15× speedup. We get another 25-120× (up to 1825 overall)
- Tailor existing system for specific use:
  - Images of 217^3, Dense region of interest,
  - Control points spacing = 5
  - 6 different GPUs (16 to 2496 cores)

Six Types of nVidia GPUs Parallel Graphics Hardware

Name	year		MP	Cores	Clock
Quadro NVS 290	2007	1.1	2 × 8	16	0.92 GHz
GeForce GTX 295	2009	1.3	30 × 8	240	1.24 GHz
Tesla T10	2009	1.3	30 × 8	240	1.30 GHz
Tesla C2050	2010	2.0	14 × 32	448	1.15 GHz
GeForce GTX 580	2010	2.0	16 × 32	512	1.54 GHz
Tesla K20c	2012	3.5	13 × 192	2496	0.71 GHz

Evolving Kernel

- Convert source code to BNF grammar
- Grammar used to control modifications to code
- Genetic programming manipulates patches
  - Copy/delete/insert lines of existing code
  - Patch is small
  - New kernel source is syntactically correct
  - Essentially no compilation errors

Before GP

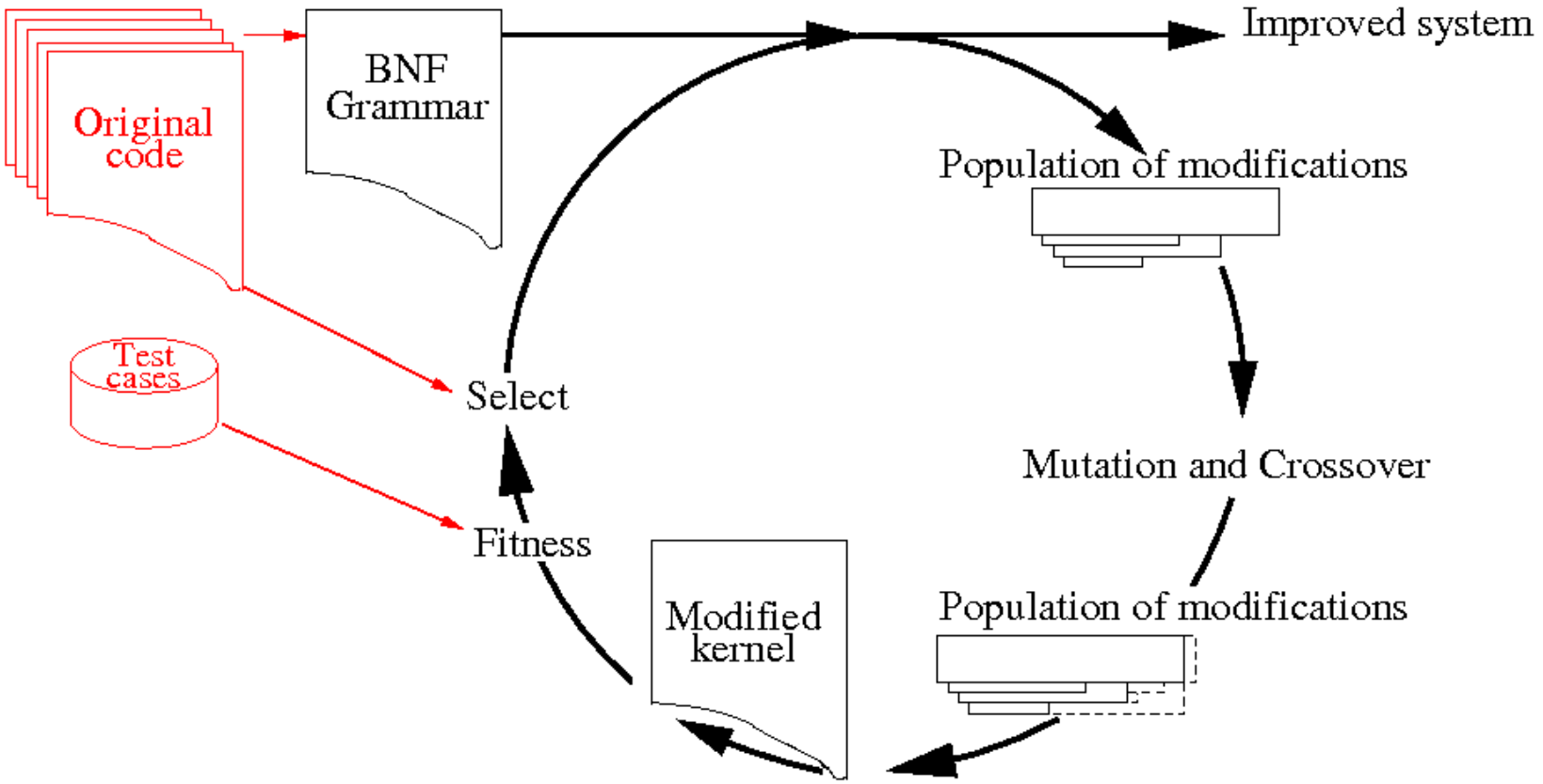
- Earlier work (StereoCamera) suggested
  - 2 Objectives: low error and fast, too different
  - Easy to auto-tune key parameters: block_size
- Therefore:
  - Single-objective GP: go faster with zero error
  - Pre and post tune 2 key parameters
  - GP optimises code (variable length)
    - Whole population (300) compiled together

Pre and Post Evolution Tuning block_size –arch option

Block_size
During development 32
tune → 64 or 128
After GP tune →128/512

-arch none
After GP tune → sm_11,sm13 or none

GP Evolving Patches to CUDA



BNF Grammar for code changes

```
if(tid<c_ActiveVoxelNumber) {
```

Line 167 kernel.cu

```
<Kkernel.cu_167>     ::=    " if" <IF_Kkernel.cu_167> " {\n
<IF_Kkernel.cu_167>  ::=    "(tid<c_ActiveVoxelNumber)"
```

```
//Set answer in global memory
positionField[tid2]=displacement;
```

Line 298 kernel.cu

```
<Kkernel.cu\_298>    ::=    "" <_Kkernel.cu_298> "\n"
<_Kkernel.cu_298>    ::=    "positionField[tid2]=displacement;"
```

Two Grammar Fragments (Total 254 rules)

BNF Grammar fragment example parameter

Replace variable c_UseBSpline with constant

```
<Kkernel.cu_17>      ::=    <def_Kkernel.cu_17>
<def_Kkernel.cu_17>  ::=    "#define c_UseBSpline 1\n"
```

In original kernel variable can be either true or false. However it is always true in case of interest.

Using constant rather than variable avoids

- passing it from host PC to GPU
- storing on GPU
- and allows compiler to optimise statements like if(1)…

Grammar Rule Types

- Type indicated by rule name
- Replace rule only by another of same type
- 25 statement (eg assignment, **Not** declaration)
- 4 IF
- No `for`, but 14 `#pragma` unroll
- 8 CUDA types, 6 parameter macro `#define`

Representation

- variable length list of grammar patches.
- 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
- Mutation movements controlled so no variable moved out of scope. All kernels compile.
- No changes to for loops. All loops terminate

Example Mutating Grammar

```
<IF_Kkernel.cu_167> ::=    "(tid<c_ActiveVoxelNumber)"
<IF_Kkernel.cu_245> ::=    "((threadIdx.x & 31) < 16)"
```

2 lines from grammar

```
<IF_Kkernel.cu_245><IF_Kkernel.cu_167>
```

Fragment of list of mutations
Says replace line 245 by line 167

`if((threadIdx.x & 31) < 16)`	Original code
`if(tid<c_ActiveVoxelNumber)`	New code

Original code caused ½ threads to stop. New condition known always to be true. All threads execute. Avoids divergence and pairs of threads each produce identical answer. Final write discards one answer from each pair.

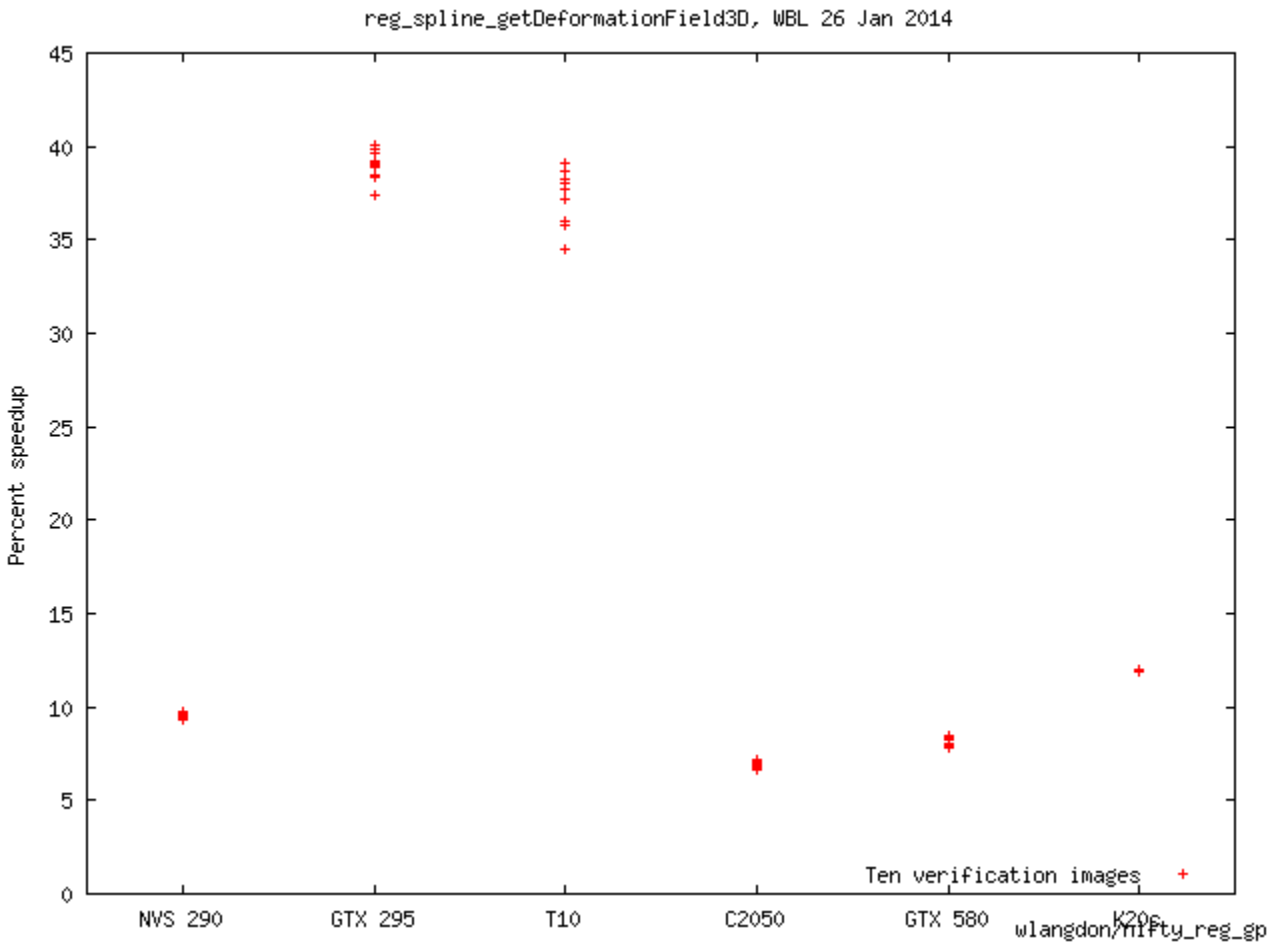
Fitness

- Run patched Kernel on 1 example image (≈1.6million random test cases)
  - All compile, run and terminate
  - Compare results with original answer
  - Sort population by
    - Error (actually only selected zero error)
    - Kernel GPU clock ticks (minimise)
  - Select top half of population.
- Mutate, crossover to give 2 children per parent.
- Repeat 50 generations
- Remove bloat
- Automatic tune again

Results

- Optimised code run on 16,816,875 test cases. Error essentially only floating point noise. Ie error always < 0.000107
- New kernels work for **all**. **Always** faster.
- Speed up depends on GPU

Nifty Reg Results



Speedup of CUDA kernel after optimisation by GP, bloat removal and with optimal block size and -arch compared to hand written kernel with default block size (192) and no -arch.
Unseen data.

GP can Improve Software

- Existing code provides
  1. It is its own defacto specification
  2. High quality starting code
  3. Framework for both:
     - Functional fitness: does evolve code give right answers? (unlimited number of test cases)
     - Performance: how fast, how much power, how reliable,…
- Evolution has tuned code for six very different graphics hardware.

END

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

Discussion Points

- Where next?
  - 3D images for more types Brain NMR
  - Port/improve other UCL CMIC software
- Code is not so fragile
- Build from existing code (source, assembler, binary)
- fitness: compare patched code v. original
  - Gives same or better answers?
  - Runs faster? Uses less power? More reliable?
- ftp://ftp.cs.ucl.ac.uk/genetic/gp-code/niftycuda.tar.gz

The Genetic Programming Bibliography

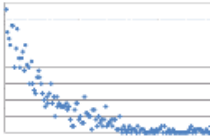
http://www.cs.bham.ac.uk/~wbl/biblio/

9505 references and 9206 online publications

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

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