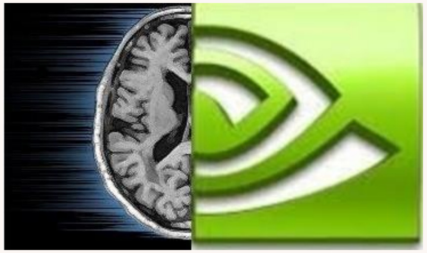
CREST UCL Centre for Medical Image Computing, 27th February 2019

Genetic Improvement for faster Medical Imaging GPU Software

W. B. Langdon

Computer Science, University College London

Simple blue example of Genetic Improvement
opencv_gp.tar.gz
RN/18/06



Improving 3D medical image registration CUDA software with genetic programming, Langdon, Marc Modat GECCO 2014

WIKIPEDIA
Genetic Improvement

Genetic Improvement of Software

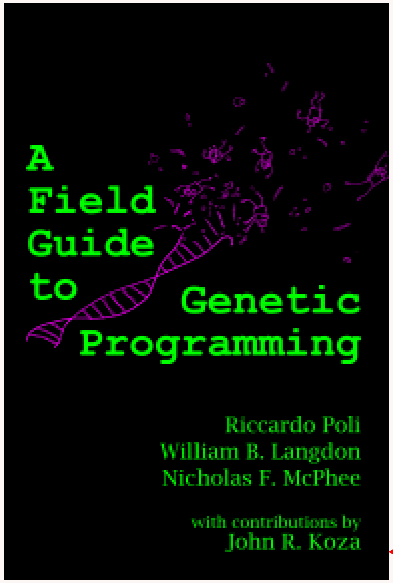
- What is Genetic Improvement
  - Genetic Programming (GP) on existing code
- What has Genetic Improvement done
  - Technology behind automatic bug fixing
  - Improvement of existing code: speedup, transplanting, program adaptation, parallel, mobile energy reduction
  - Genetic Improvement of Data CGIP, better predictions
- Faster CUDA 3D kernel for NiftyReg
- Conclusions

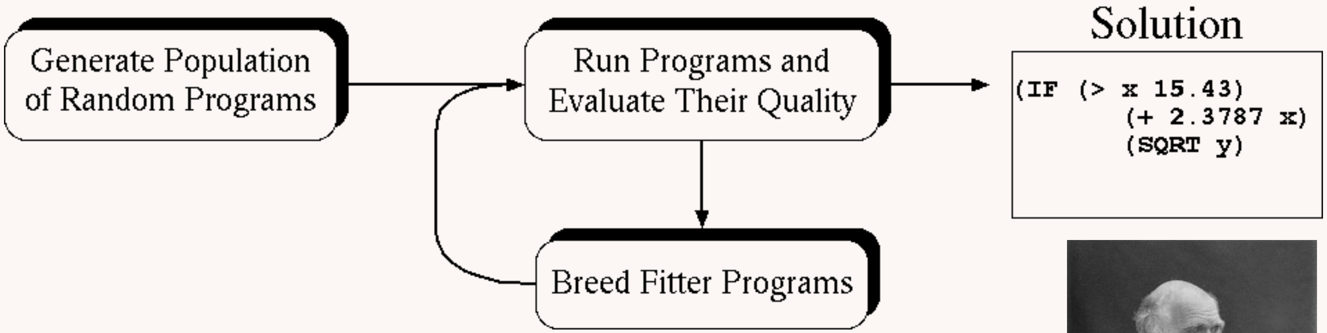
What is Genetic Improvement

Genetic Improvement

Use GP to evolve a population of computer programs

- Start with representation of human written code
- Programs’ fitness is determined by running them
- Better programs are selected to be parents
- New generation of programs are created by randomly combining above average parents or by mutation.
- Repeat generations until solution found.




Generate Population of Random Programs

Run Programs and Evaluate Their Quality

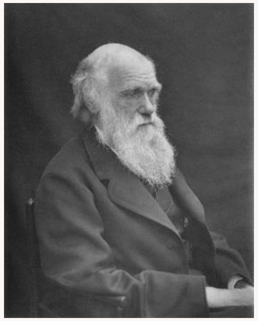
Breed Fitter Programs

Solution

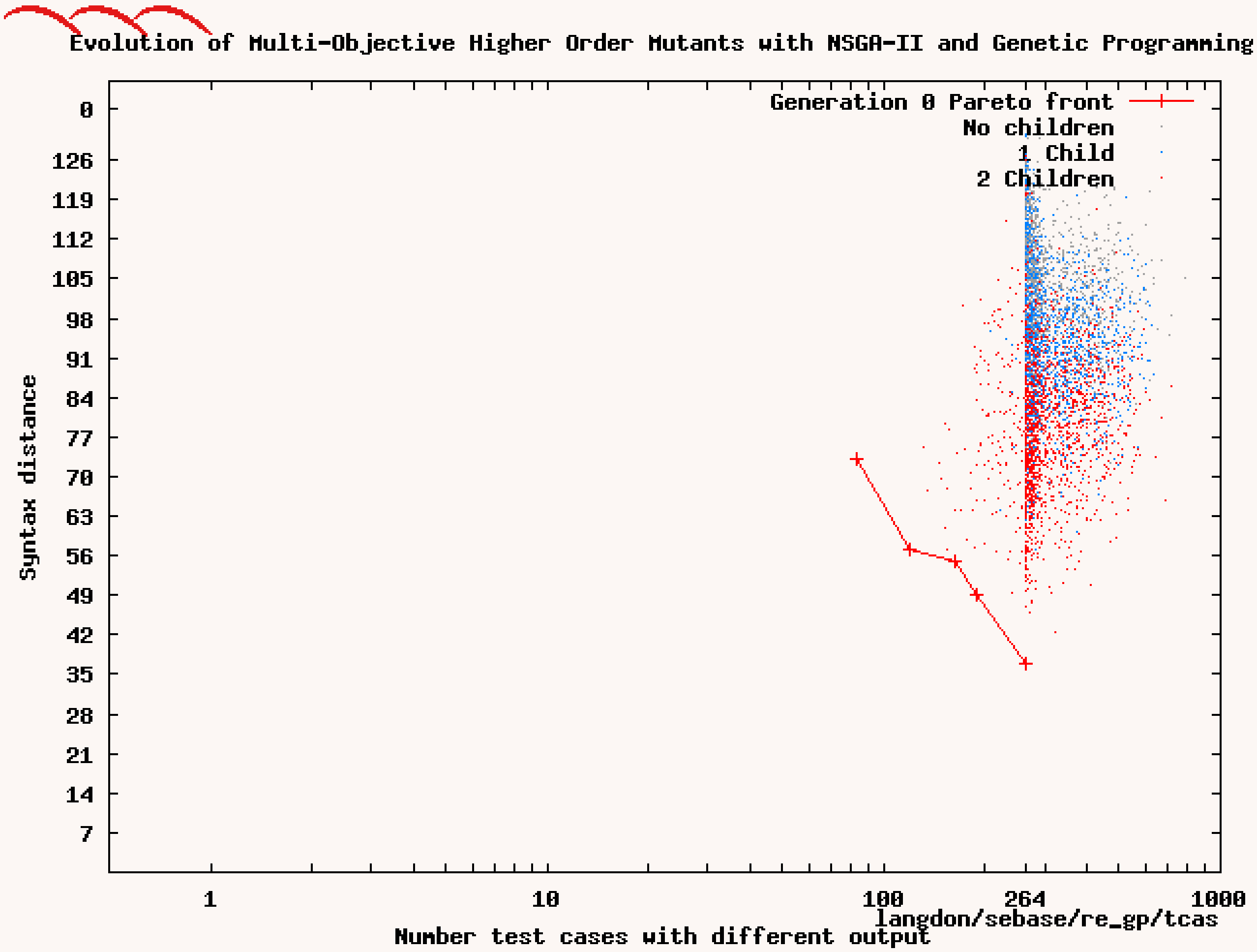
```
(IF (> x 15.43)
        (+ 2.3787 x)
        (SQRT y)
```

Free PDF

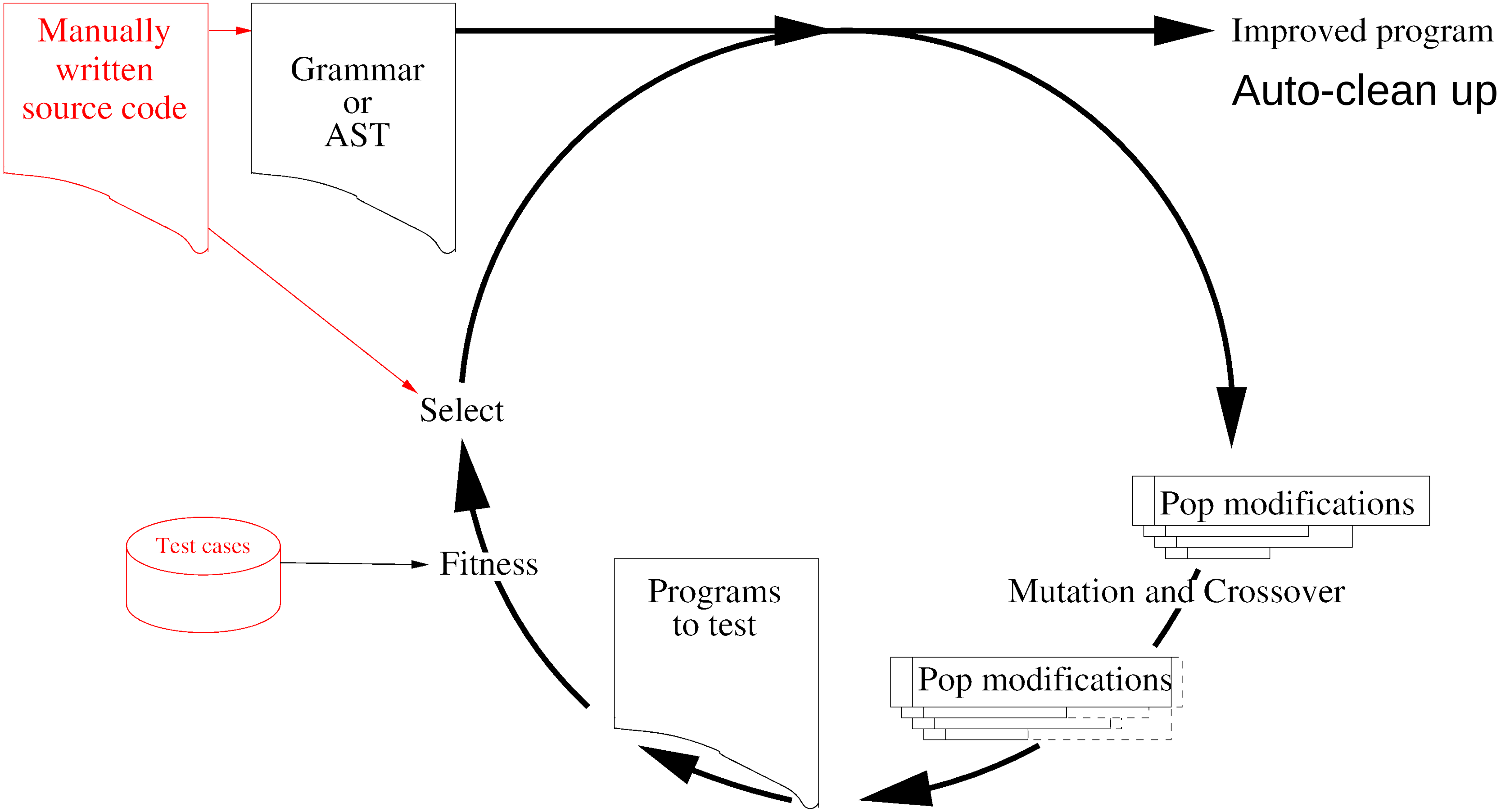
Free E-book kindle


Charles Darwin 1809-1882

Evolving population of programs



Typical GI Evolutionary Cycle



Many types of mutation.
Eg replace line of C++ code with another from the same file.

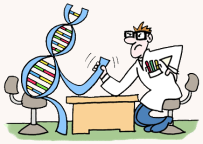
GI Automatic Coding

- Genetic Improvement does not start from zero
- Use existing system
  - ▪ Source of non-random code
  - – Use existing code as test “Oracle”. (Program is its own functional specification)
  - – Can always compare against previous version
  - – Easier to tell if better than if closer to poorly defined goal functionality.
- Testing scales (sort of). Hybrid with “proof” systems

What has
Genetic Improvement done

GP Automatic Bug Fixing (APR)

- Run code: example to reproduce bug, a few tests to show fixed code still works.
- Search for replacement C statement within program which fixes bug. Fault location tool
- Real bugs in real programs (mostly C/C++ or Java).
  - Multiple prizes and best papers, including:
  - 1st prize Human-Competitive [ICSE] Gold Humie



- In daily use: Iceland health clinic [GI-2017] Python Facebook SapFix Mark Harman

GI to Speed up human written programs

- Bowtie2, 70 times faster [IEEE TEVC 2015]
- GPGPU BarraCUDA [BioData Mining]
  - – In use since 2015. 4000 downloads from SF
  - – On real data speed up to 3 times (arXiv.org)
  - – Commercial use by Lab7 (in BioBuilds 2015)
  - – Ported by IBM to their Power8
  - – Cambridge Epigenetix

GTX 1080 21x faster than bwameth (twin core CPU)

Microsoft Azure GPU cloud

- RNAfold CUDA, AVX, improved predictions

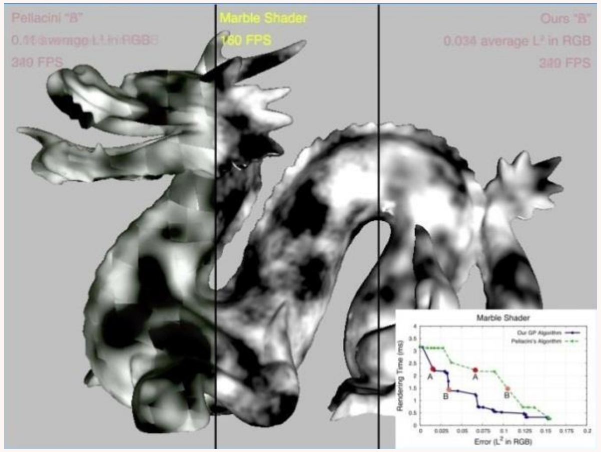
Genetic Improvement to Reduce Resource Consumption

- Energy reduction [GECCO 2015a,SSBSE] particularly for mobile computing [GI-2017]
- RAM memory reduction [GECCO 2015b]
- Reduce run time [pknotsRG,OpenCV, RNAfold, EuroGP2019]
- Choose better library [SSBSE-2017]
- Improve library [SSBSE 2014,2016]

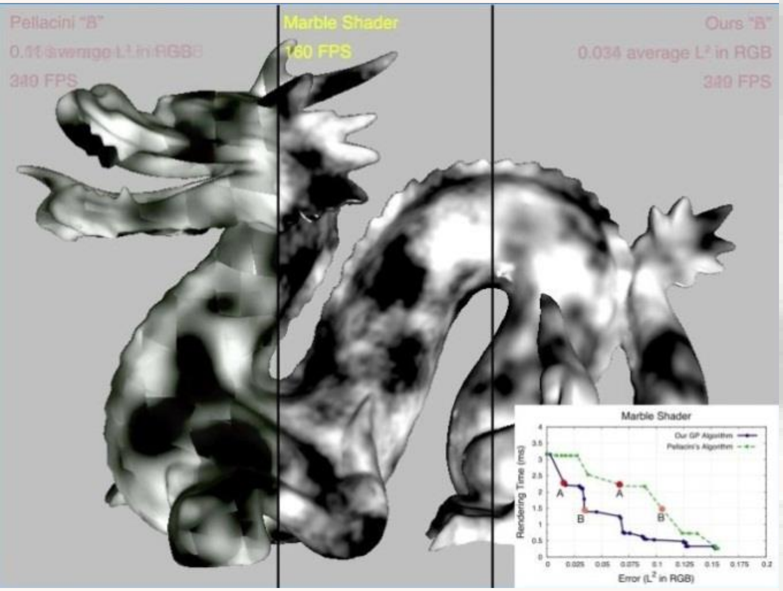
GI to Improve functionality

- Transplanting C++ [Marginean SSBSE’15, ISSTA’15]

  E.g. graph layout into Kate, H.264 into VLC, awarded Gold Humie, 26hours CPU v. 20days
- Autoporting
  – gzip to GPU [CEC 2010], RNAfold to SSE [GI-2017]
- Better RNA structure prediction
- Improving GPU shaders [2011]



GI Improving GPU shaders [2011]



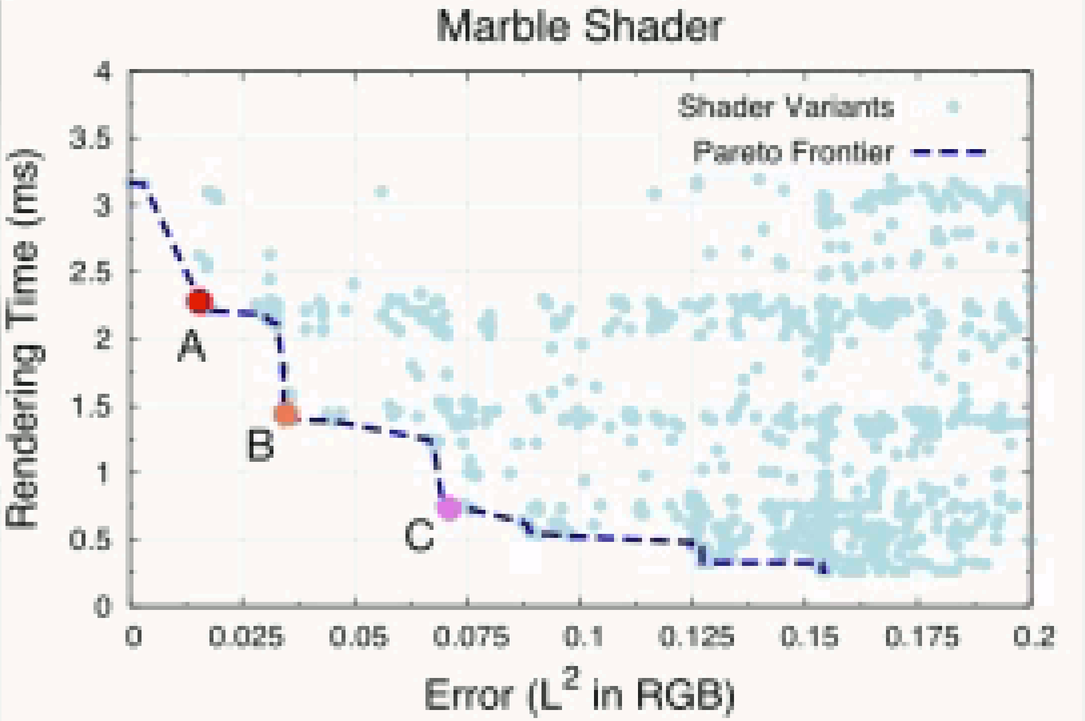


Fig 1. number of core papers on genetic improvement

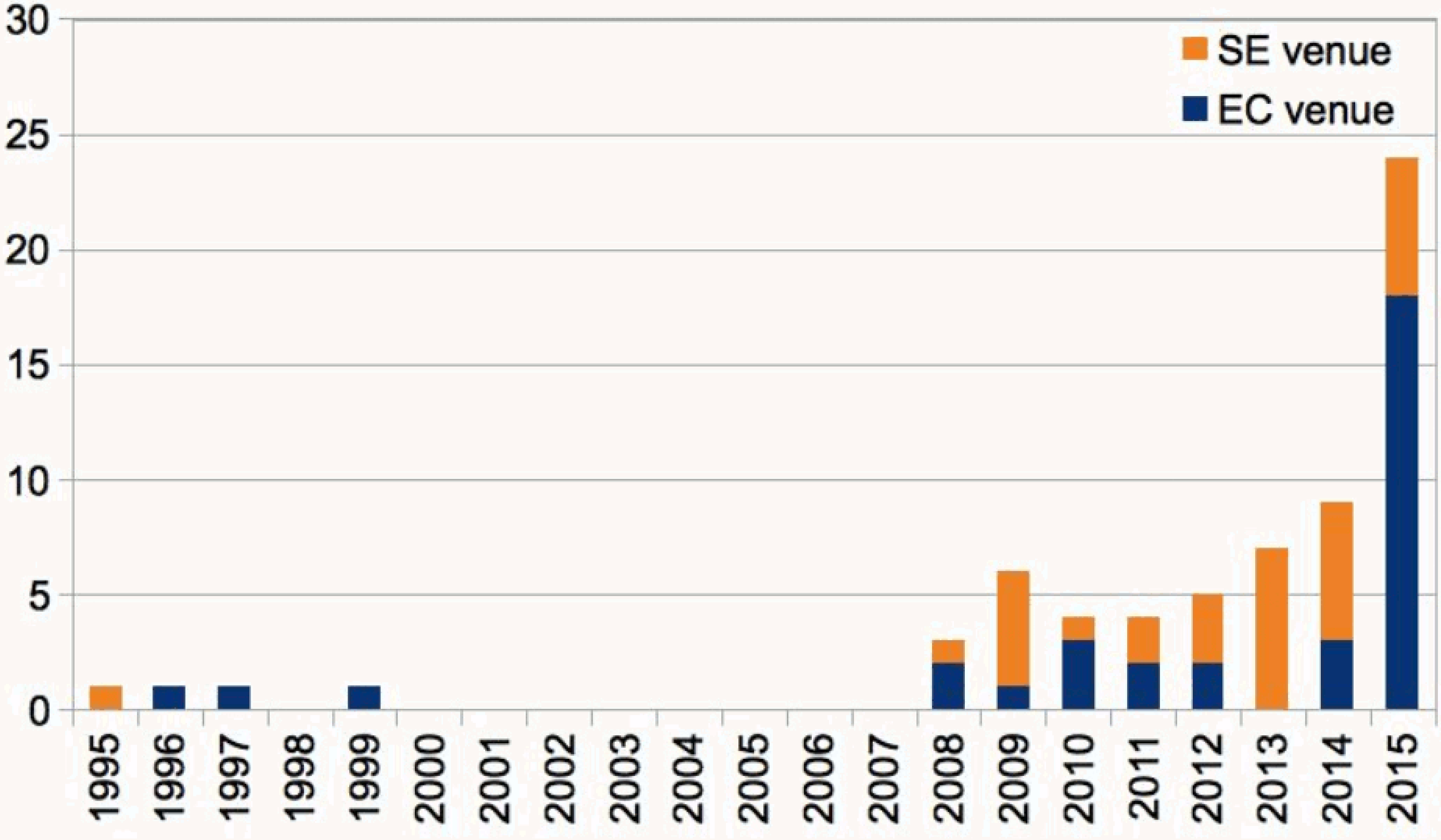
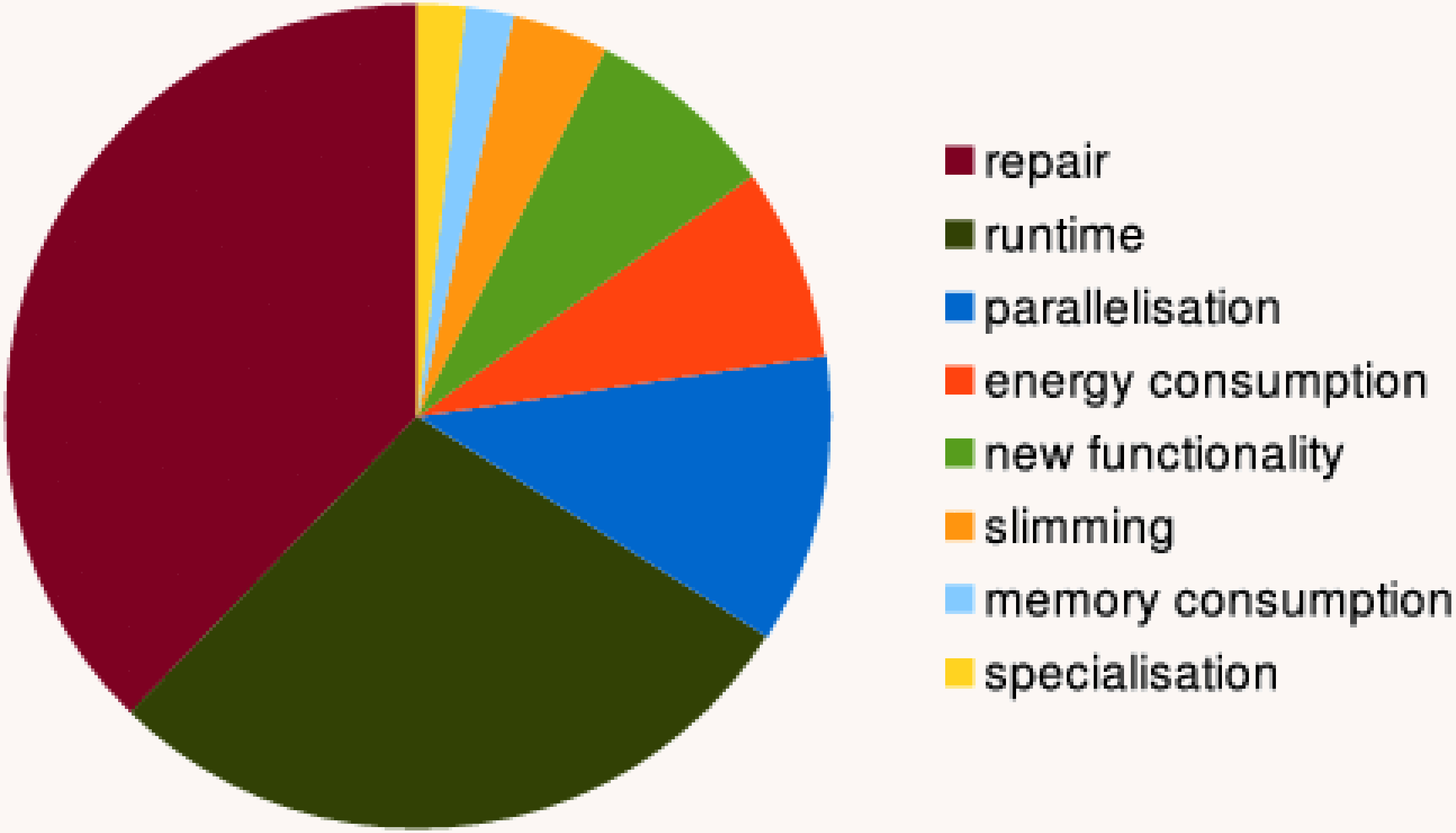


Fig 3. software applications of empirical studies in core papers on genetic improvement



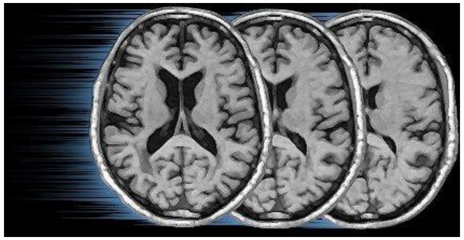
Maintaining Embedded Constants

- EuroGP 2018
  - – RNAfold 7000 lines of code 50000 numbers
  - – net 20% better prediction of RNA structures
  - – In ViennaRNA package since 2.4.7
- SSBSE-2018 sqrt converted to cube root
- RN/18/05 generate $\log_2$ from open source maths framework

Improving 3D Medical Image Registration CUDA Software with Genetic Programming

- NiftyReg reg_spline_getDeformationField3D
- Pre and Post GP tuning of key GPU code
- GP BNF grammar
- Mutation, crossover gives new kernel code
- Fitness: compile, run on random example
- Results: it works, where next?

Evolving Faster NiftyReg 3D Medical Image Registration CUDA kernels



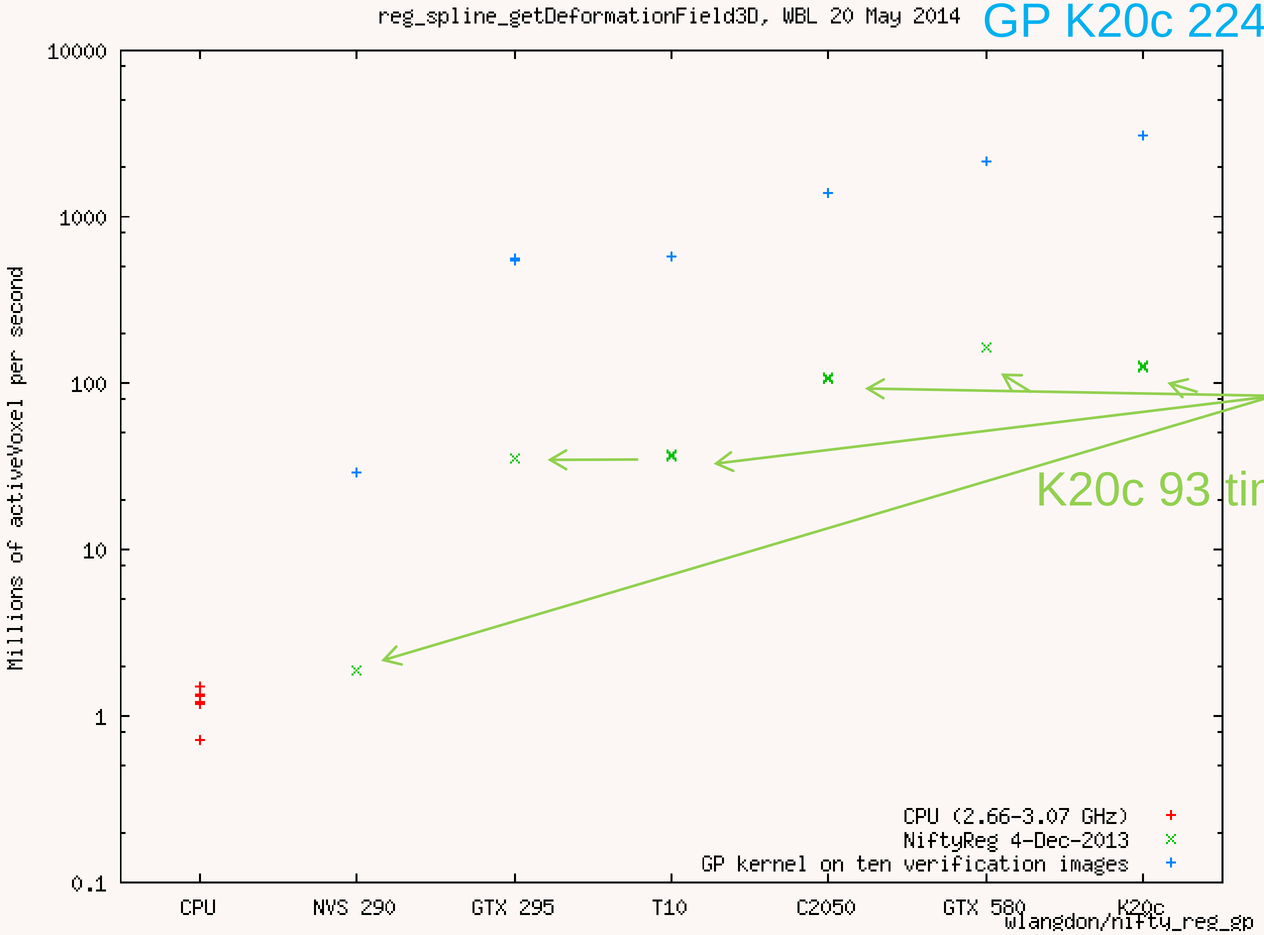
- What is NiftyReg?
  - UCL CMIC M.Modat (2009) sourceForge 16000 C++
- 3D Medical Images
  - Magnetic Resonance Imaging (MRI) brain scans 1mm resolution → 217^3=10,218,313 voxels
- Registration: NiftyReg nonlinear alignment of 3D images
- Graphics GPU parallel hardware
- CUDA allows C++ functions (kernels) to run in parallel

reg_bspline_getDeformationField3D

- Existing CUDA code 97 lines
- 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.

CPU v GPU

Note: Log vertical scale



GP K20c 2243 times CPU

Improved Kernels

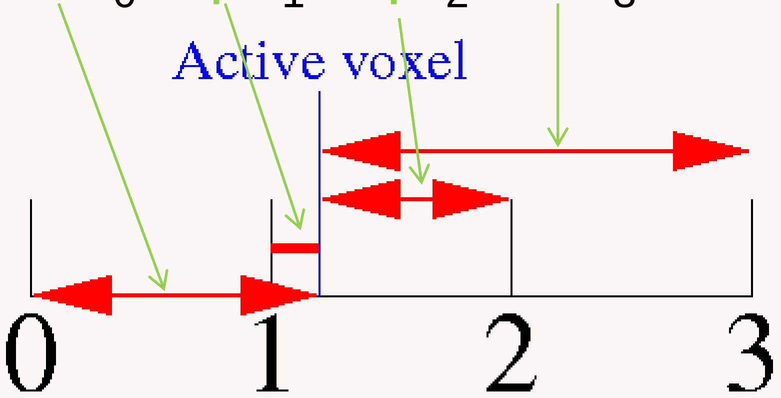
Original kernel

K20c 93 times CPU

Spline Interpolation

In one dimension displacement is linear combination of displacement at four neighbouring control points:

Displacement = $\alpha d_0 + \beta d_1 + \gamma d_2 + \delta d_3$



Spline coefficients α β γ δ given by cubic polynomial of distance from voxel to each control point 0,1,2,3.

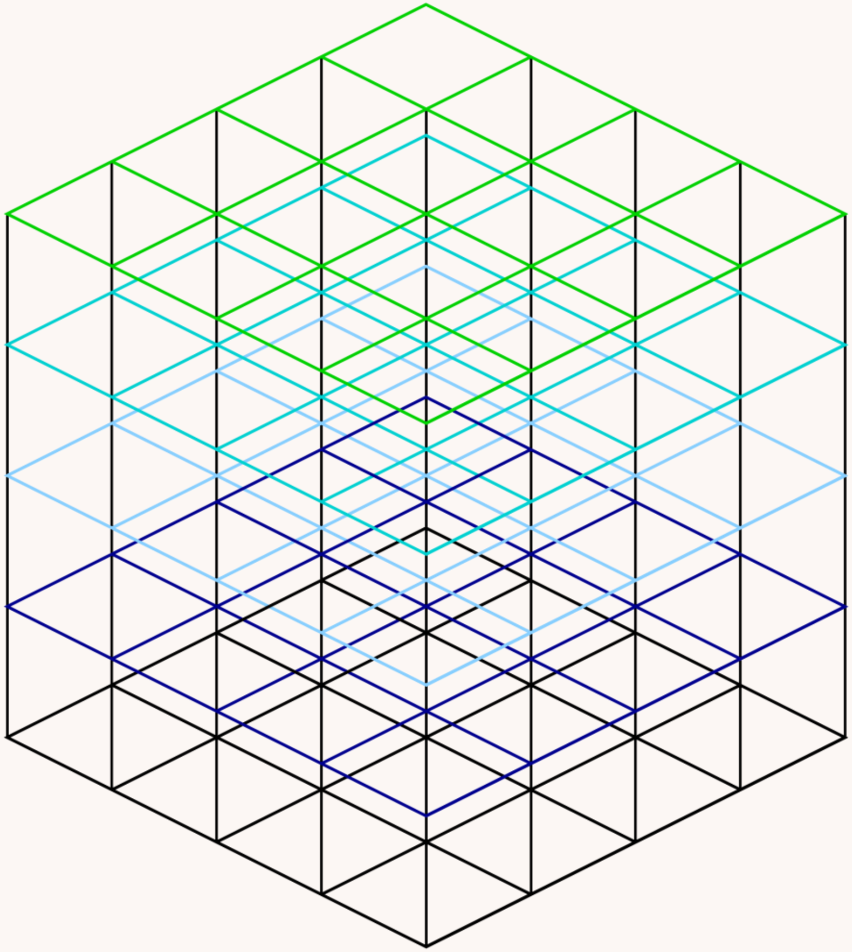
In 3D have 64 neighbours, so sum 64 terms.
If control points are five times unit distance, there are only 4×5=20 coefficients which can be precalculated.

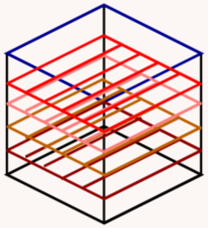
spline interpolation between 4×4×4=64 neighbours

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
- Approximately 600 flops per voxel.
  - Re-use limited by register/shared memory.
- Read voxel list and control points displacement from global memory (via texture cache)
- Write answer δx,δy,δz to global memory

Manual Improved Kernel

- Fixed control grid spline coefficients (20) need be calculate once and then stored.
- GPU has multiple types of memory:
  - Global large off chip, 2 level cache, GPU dependent
  - “Local” large off chip, shares cache with global
  - “Textures” as global but read only proprietary cache (depends on GPU).
  - “Constant” on chip 64K read only cache, contention between threads, GPU dependent
  - “shared” on chip 16-48K, configurable, GPU dependent
  - Registers fast, limited, GPU dependent
- Leave to GP to decide how to store coefficients

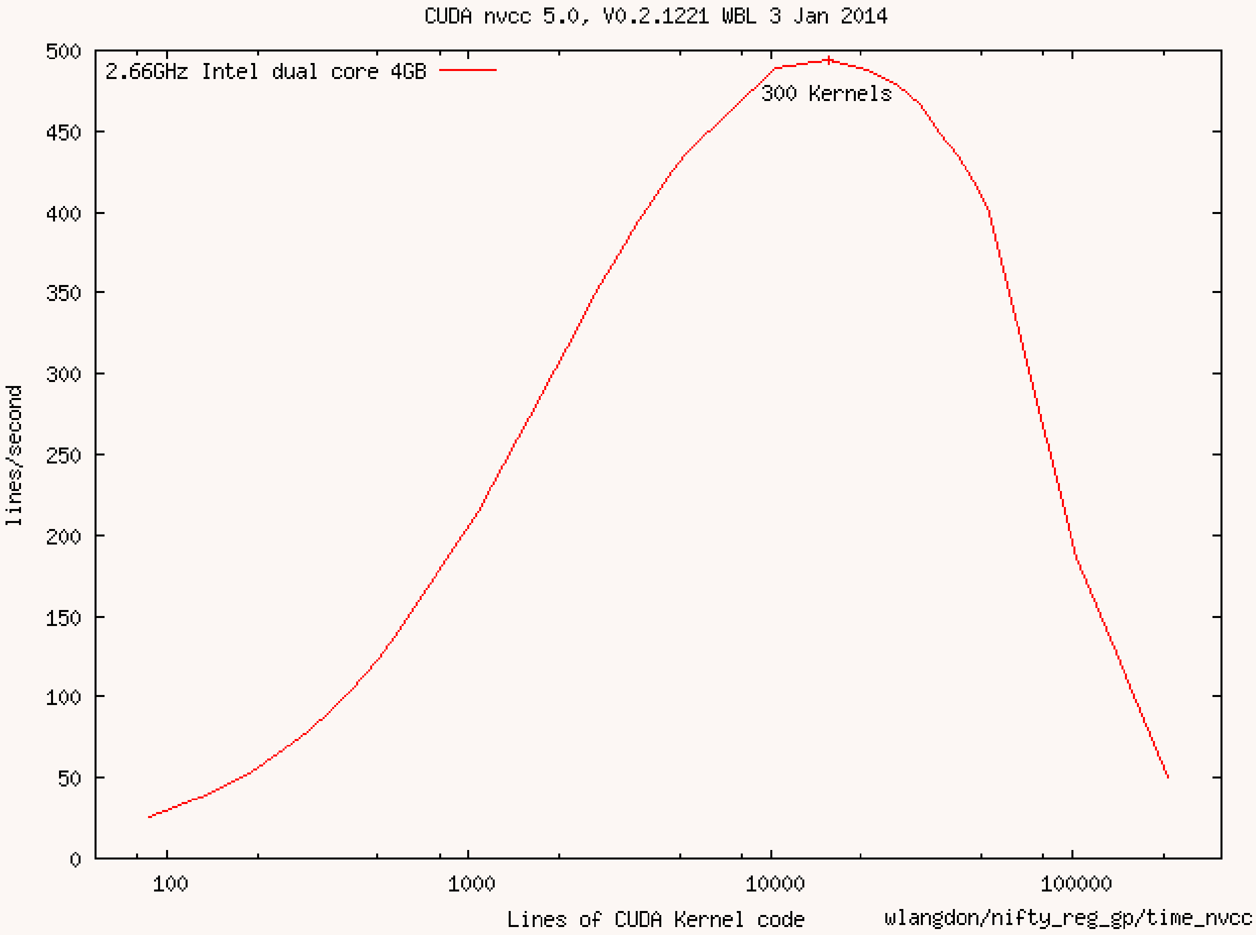
Six Types of nVidia GPUs Parallel Graphics Hardware

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

Before GP

- Easy to auto-tune key parameters:
  - – Number of threads, compiler GPU -arch
- 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

Compile Whole Population



Note Log x scale

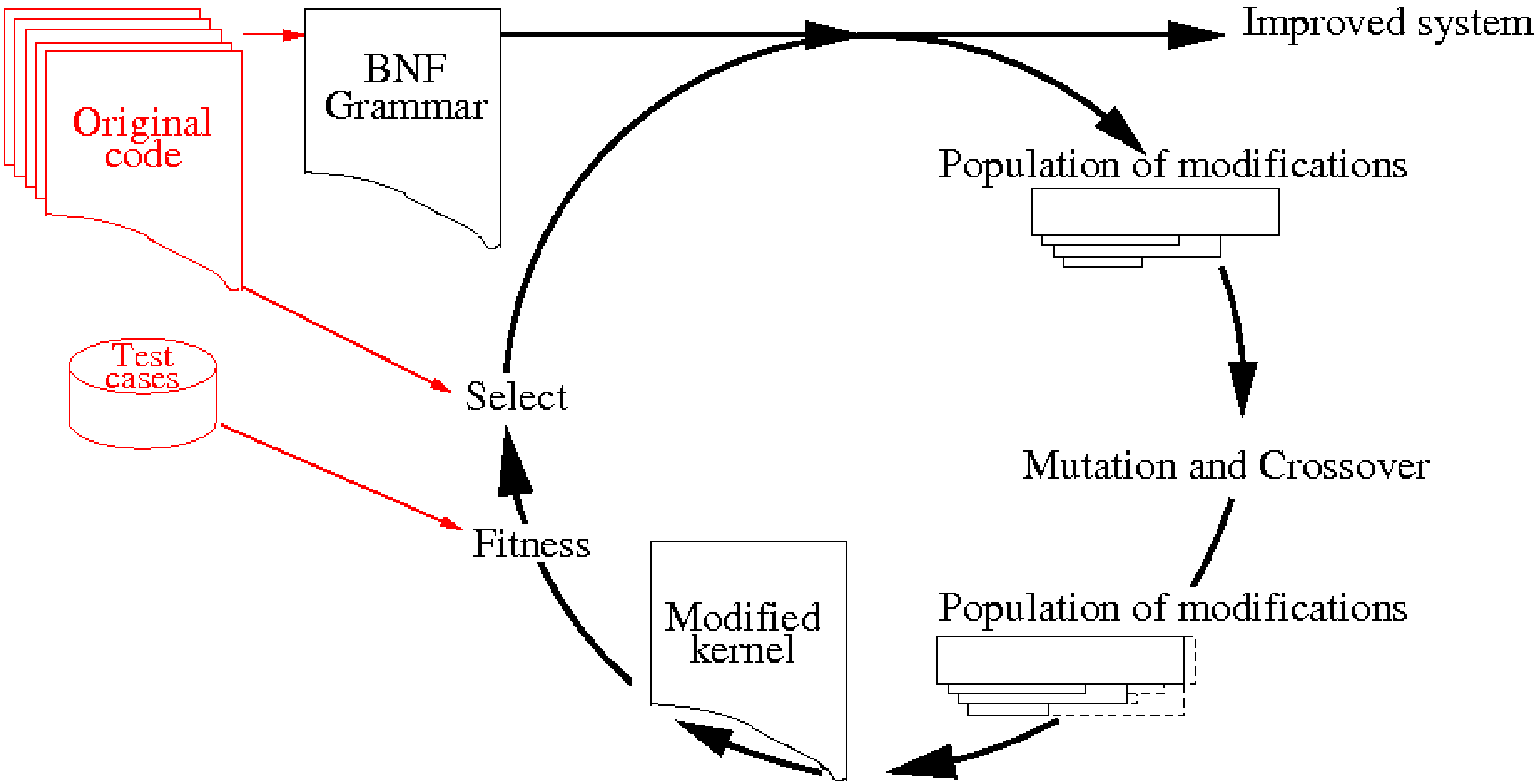
Compiling 300 kernels together is 19.3 times faster than running the compiler once for each.

Pre and Post Evolution Tuning
1.number parallel threads per block
2.compiler –arch code generation

1.CUDA Block_size parallel thread per block
During development 32
tune → 64 or 128
After GP tune →128/512

2. Compiler code -arch sm_10
After GP tune → sm_10, sm_11 or sm_13

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.
  - no size limit, so search space is infinite
- crossover: child takes parts of 2 parents
- mutation add 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)
```

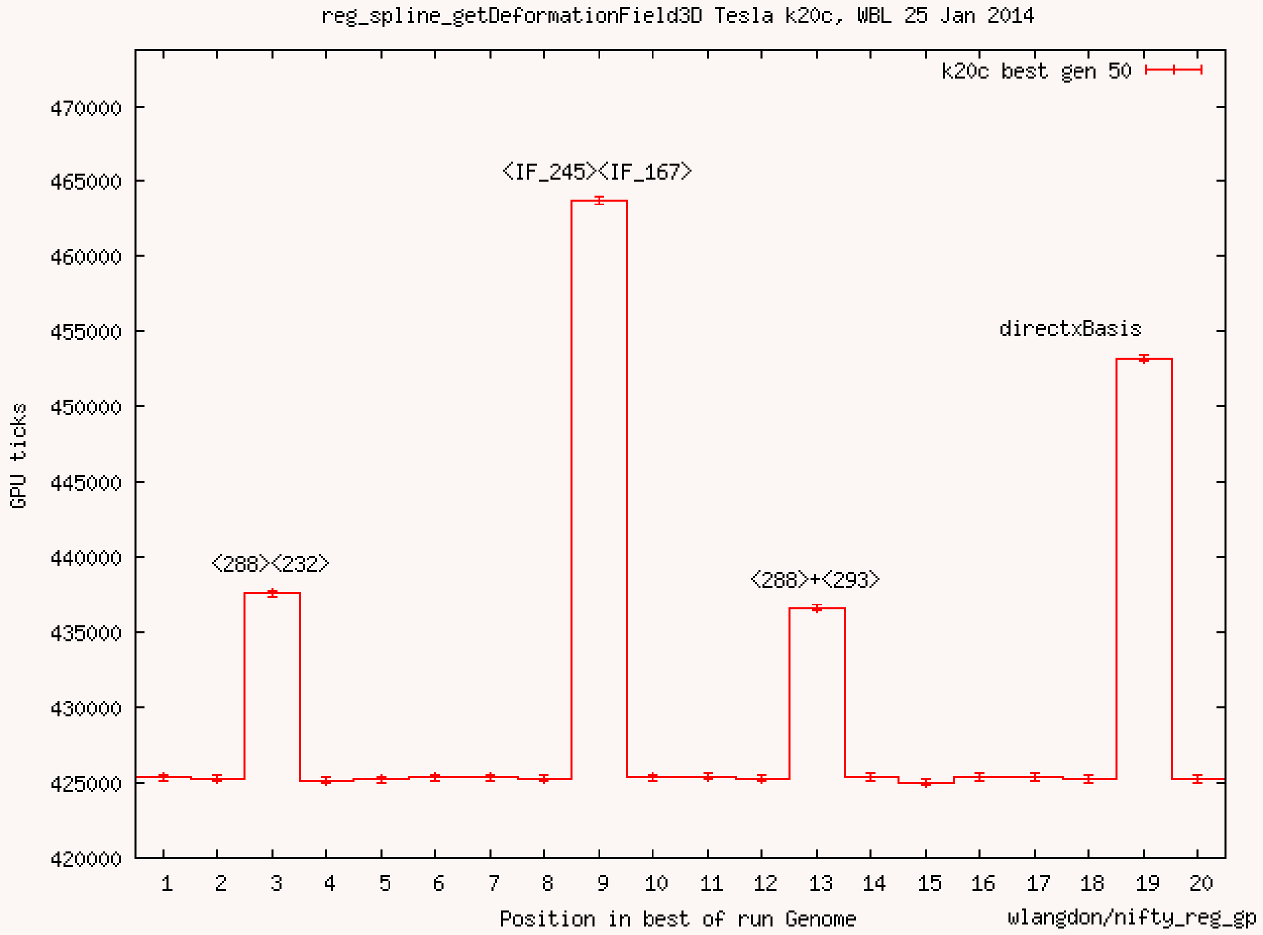
New code

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

Bloat Removal

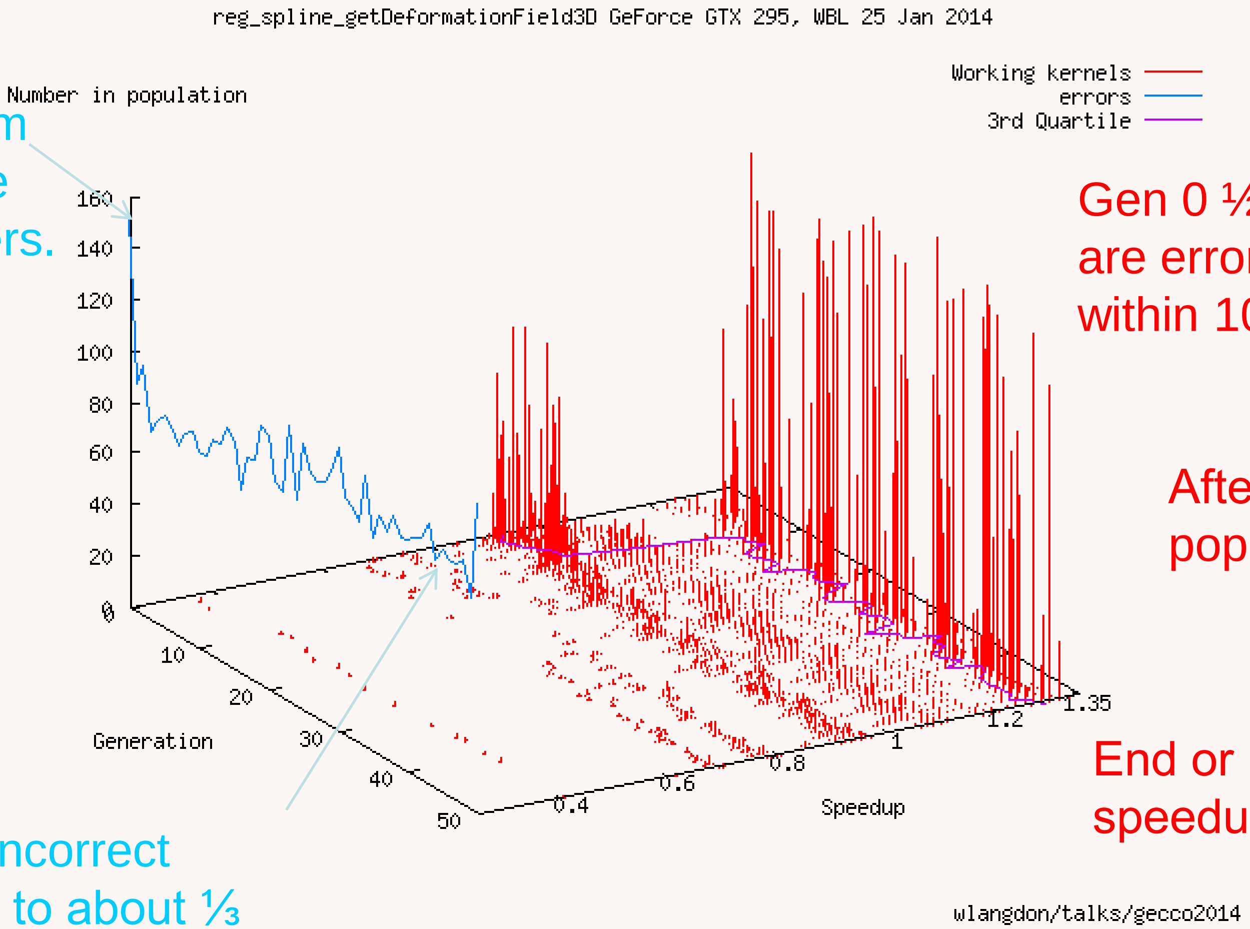


Fitness effect of each gene evolved by GP tested one at a time. Only important genes kept.

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

Evolution of kernel population



wlangdon/talks/gecco2014

Gen 0 ½ random kernels produce incorrect answers.

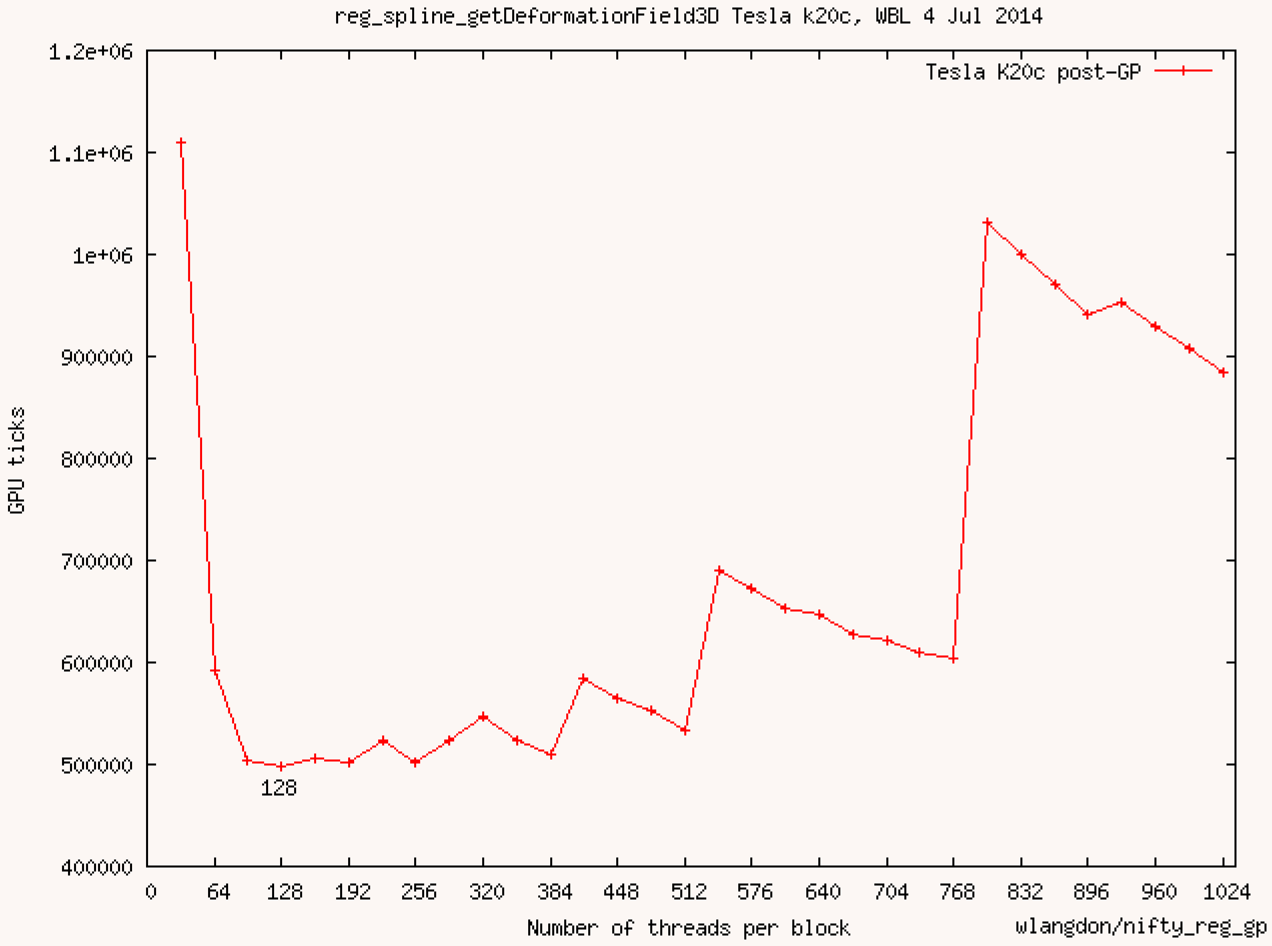
Gen 0 ½ population are error free and within 10%

After gen7 ≥1/3 pop are faster

End or run ≥½ pop speedup ≥28%

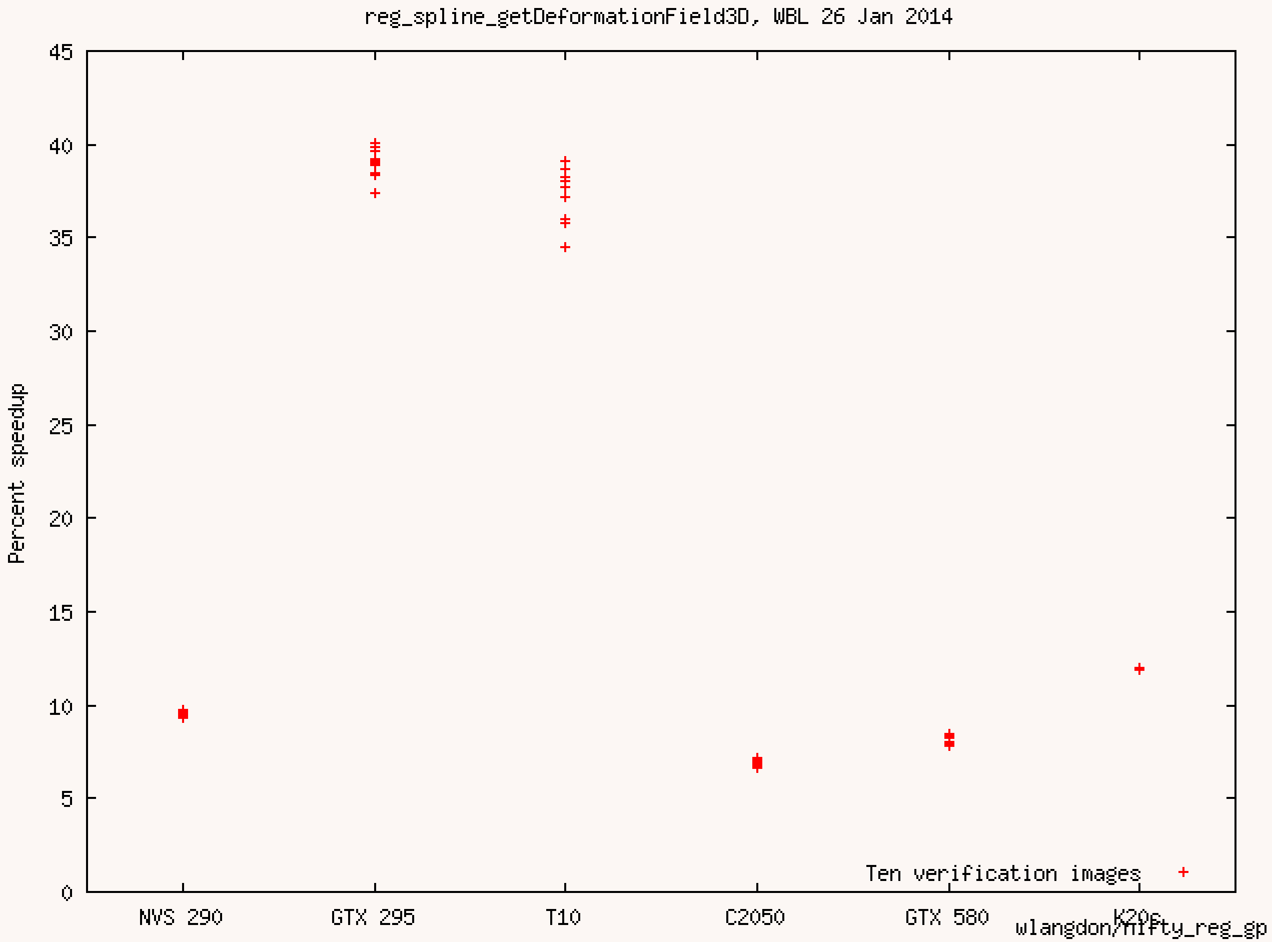
Fraction of incorrect kernels falls to about ⅓

Post Evolution Auto-tune



Compile and run GP kernel with all credible block_size and chose fastest

NiftyReg Results



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

Tesla K20c
NiftyReg Code changes

Remove CUDA code	New CUDA code
	#define directxBasis 1
if((threadIdx.x & 31) < 16)	if(1)

displacement=make_float4(
0.0f,0.0f,0.0f,0.0f);
	displacement.y +=
	tempDisplacement(c,b).y * basis;
	nodeAnte.z =
	(int)floorf((float)z/gridVoxelSpacing.z);

directxBasis means pre-calculated X-spline co-efficients are read from texture memory not calculated.

16 idle threads exactly duplicate 16 others.

Two genes <288><232> <288>+<293> safe but rely on optimising compiler to remove unneeded code.

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.

Six impossible things before breakfast

- To have impact do something considered impossible.
- If you believe software is fragile you will not only be wrong but shut out the possibility of mutating it into something better.
- Genetic Improvement has repeatedly shown mutation need not be disastrous and can lead to great things.

Conclusions

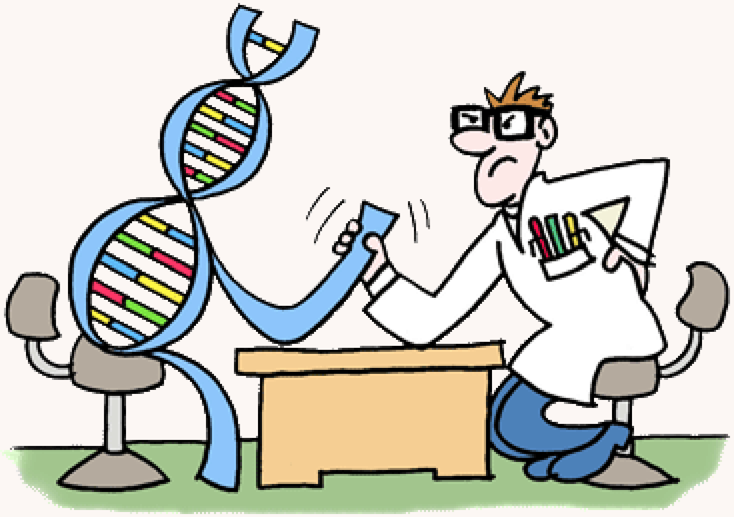
- Genetic Improvement (GI) applies Darwinian survival of the fitness to software
  - bugfixing, software transplanting, performance improvement, faster answers or better answers. Eg BarraCUDA RNAfold data,SSE+AVX,CUDA
- NiftyReg, GP working on top of manual improvements. Up to 2234 times faster CPU
- Software is not fragile

break it, bend it, Evolve it

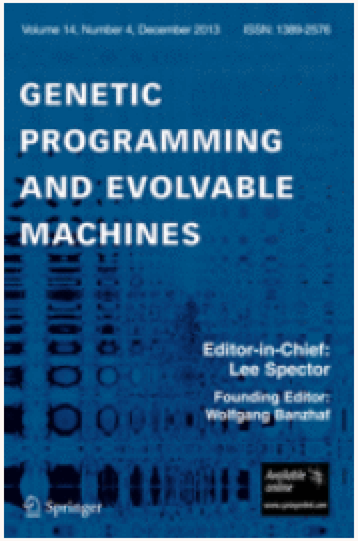
GI 2🍁19

GI 2019, Montreal, ICSE-2019 workshop, 28th May

WIKIPEDIA
Genetic Improvement



Humies: Human-Competitive
$10,000 prizes. Finals in Prague



Volume 14, Number 4, December 2013 ISSN: 1389-2576

GENETIC PROGRAMMING AND EVOLVABLE MACHINES

Editor-in-Chief:
Lee Spector

Founding Editor:
Wolfgang Banzhaf

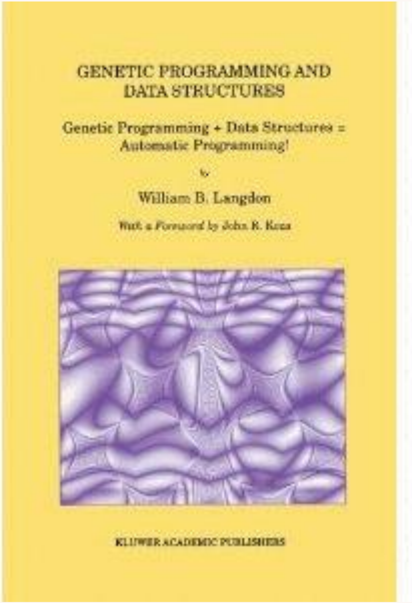
Springer

GI special issue

W. B. Langdon

CREST

Department of Computer Science



GENETIC PROGRAMMING AND DATA STRUCTURES

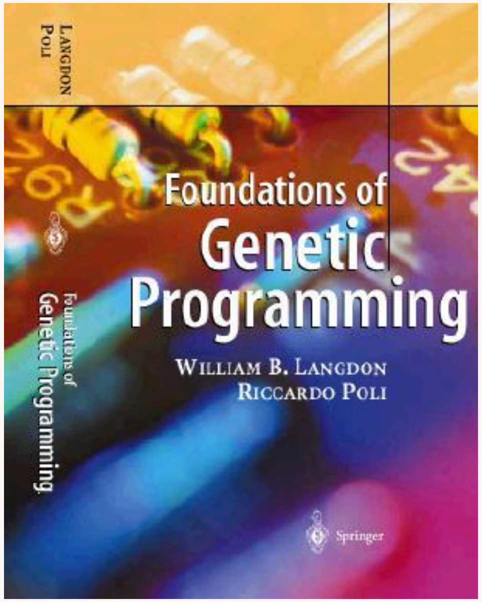
Genetic Programming + Data Structures = Automatic Programming!

by

William B. Langdon

With a Foreword by John R. Koza

KLUWER ACADEMIC PUBLISHERS

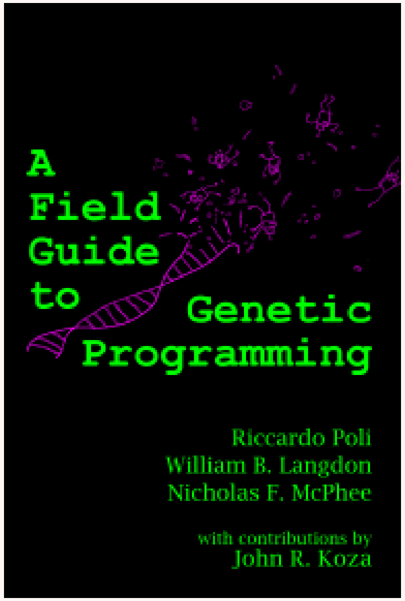


LANGDON POLI

Foundations of Genetic Programming

WILLIAM B. LANGDON
RICCARDO POLI

Springer



A Field Guide to Genetic Programming

Riccardo Poli
William B. Langdon
Nicholas F. McPhee

with contributions by
John R. Koza

Where is Genetic Improvement Headed

- Automatic coding
- Automatic testing (oracles)
- code transplant, data transplant

Goals of Genetic Improvement

- Totally automatic programming still distant
- Intermediate GI as stepping stone to higher level programming. Programmer says what needs to be done by test cases.
- Program assembled from existing(perhaps open source) code or more automated bag-of-parts software product lines (mashups)
- Automatic customisation, per user versions, many (30184 [Monperrus,2017]) version computing

GI Automatic Coding

- Genetic Improvement may also allow us to trade improvement in one aspect against loss in another.
  - E.g. reduce accuracy but faster execution
  - (Can sometimes improve both)
- Customise per user (dreaming smart phone)
- Predict what user will want to next.
  - E.g. yesterday read news page at 8:30 so today load it into cache before they reach underground tube station.

Automatic Testing

- Hardware Improvement: Tetsuya Higuchi Analogue EHW chip for mobile phones
- Software quality continues to be dominated by the cost of manual effort
- Existing test suites are often run automatically
- Evolution can automatically create test cases (goal: code coverage) but still lacks knowledge of the correct answer (known as the test oracle problem).

Automatic Oracle Generation

- Current automatic oracles are crude:
  - did the program terminate? Did it crash?
- Given huge number of existing open source test suites [SBST 2017], can Machine Learning:
  - infer the answer expected of a test case?
  - could Machine Learning get close or give plausible answers?
  - Reject non-plausible answers?

Demo count blue pixels

- Assumes Unix, tsch gcc
- http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/opencv_gp.tar.gz
- gunzip -c ftp/gp-code/opencv_gp.tar.gz | tar xvf –
- README.txt

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?

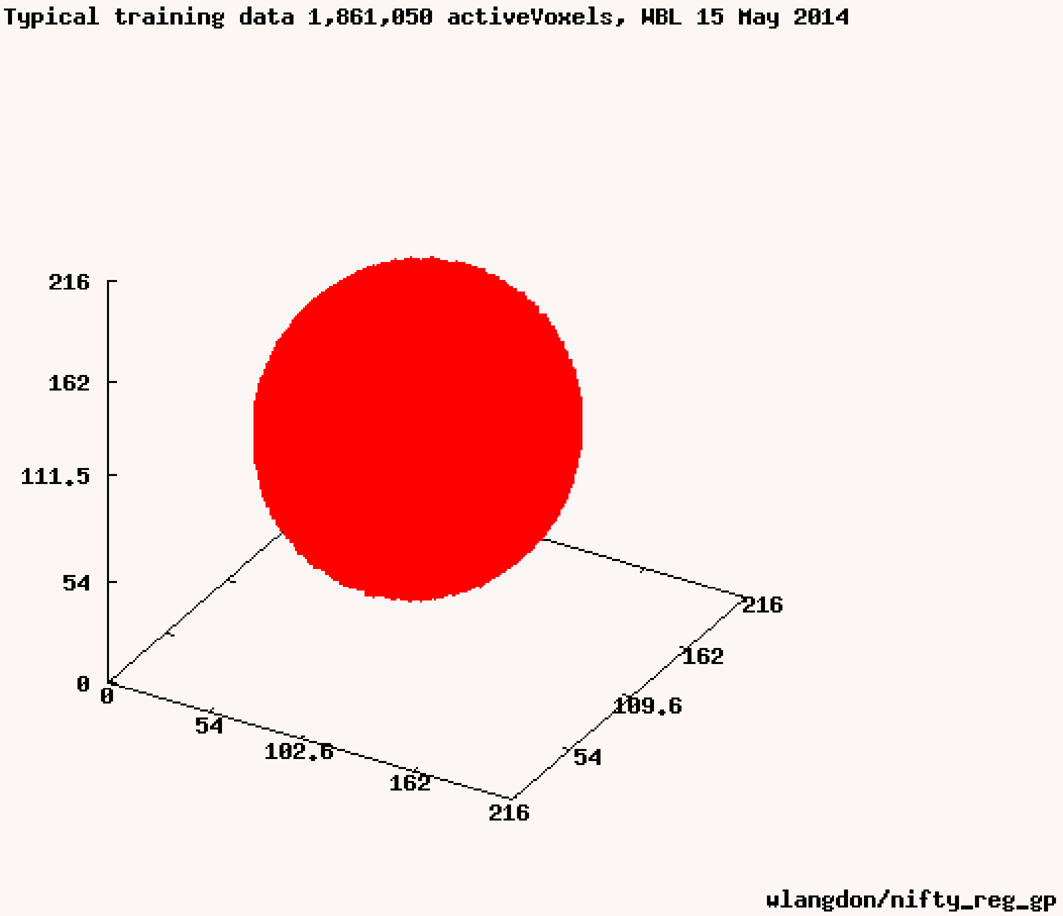
GP Automatic Coding

- Target open source system in use and being actively updated at UCL.
- Chose NiftyReg
- GPU already give 15× speedup or more. We get another 25-120× (up to 2243×CPU)
- 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)

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
  - No compilation errors. Loops terminate
    - Scoping rules. Restrict changes to loops and loop variables

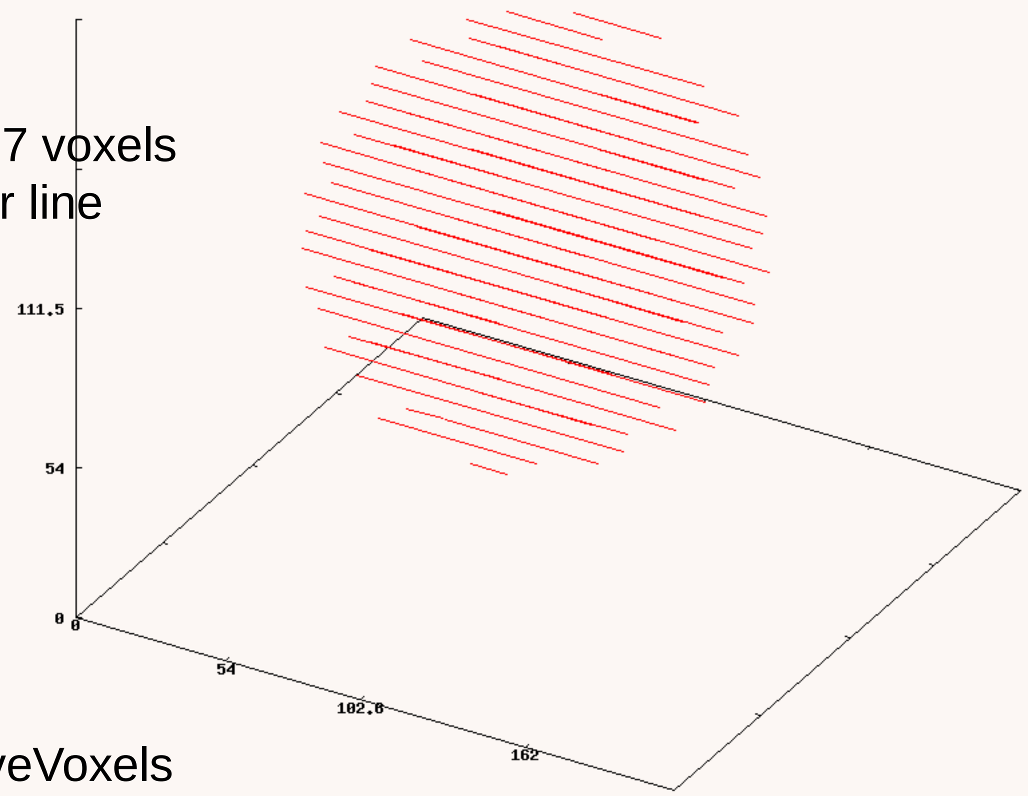
Typical Active Part of Image



Original Kernel

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

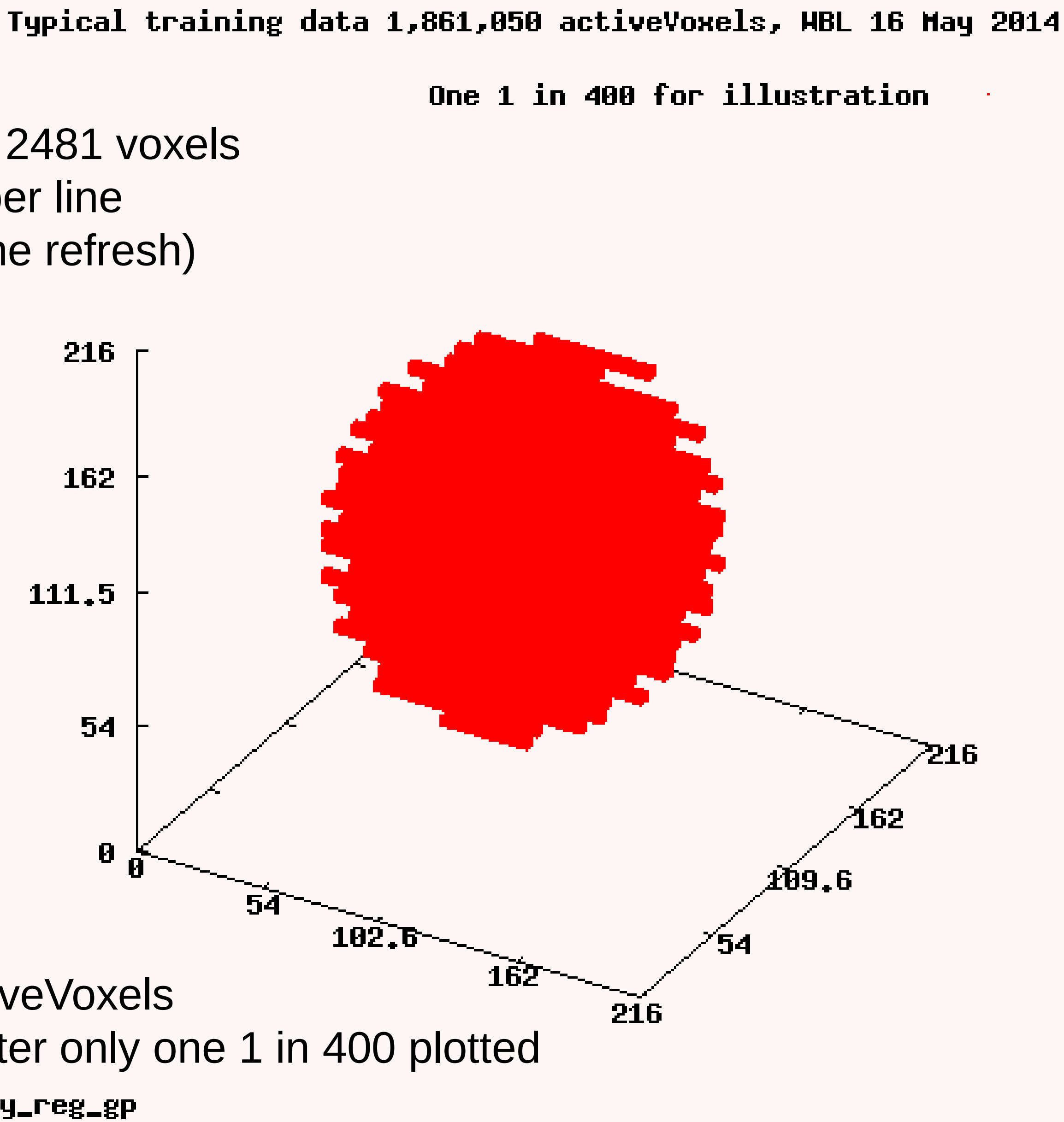
On average 97 voxels
processed per line



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

Manual code changes

- Specialise to fixed (5) control point spacing
- Package coefficient __device__ function() so GP can use or replace by storing pre-calculated values.
- Expose kernel launch parameters for auto-tuner.
- grammar automatically created except for variable scope limits

CUDA Grammar Types

- `#pragma unroll`
- `__restrict__`
- `__launch_bounds__`

• `c_UseBSpline`	true
• `c_controlPointVoxelSpacing`	5
• `constantBasis`	Pre-calculate
• `BasisA`	Array index order
• `directxBasis`	Pre-calculate x
• `RemX`	Save x%5

GP Evolution Parameters

- Pop 300, 50 generations
- 50% 2pt crossover
- 50% mutation (3 types delete, replace, insert)
- Truncation selection
- 1 test example, reselected every generation
- 1.5 hours
- Unique initial population (≈hence 300)

Tesla K20c StereoCamera Code changes

Remove CUDA code	New CUDA code
`int * __restrict__ disparityMinSSD,`	
`volatile extern __attribute__ ((shared)) int col_ssd[];`	`extern __attribute__ ((shared)) int col_ssd[];`
`volatile int* const reduce_ssd = &col_ssd[(64 )*2 -64];`	`int* const reduce_ssd = &col_ssd[(64 )*2 -64];`
	`#pragma unroll 11`
`if(X < width && Y < height)`	`if(dblockIdx==0)`
`__syncthreads();`	
	`#pragma unroll 3`

Parameter `disparityMinSSD` no longer needed as made shared (ie not global)
All `volatile` removed
Two `#pragma` inserted
`if()` replaced
`__syncthreads()` removed

GP and Software

- Genetic programming can automatically re-engineer source code. E.g.
  - hash algorithm
  - Random numbers which take less power, etc.
  - mini-SAT

EuroGP 2014

- fix bugs (5 10^6 lines of code, 16 programs)
- create new code in a new environment (GPU) for existing program, gzip WCCI 2010
- 70 speed up 50000 lines of code IEEE TEC
- 7 times speed up for stereoKernel GPU

EuroGP 2014

3D NMR Brain scans GECCO 2014

GP Automatic Coding

- Show a machine optimising existing human written code to trade-off functional and non-functional properties.
  - E.g. performance versus:

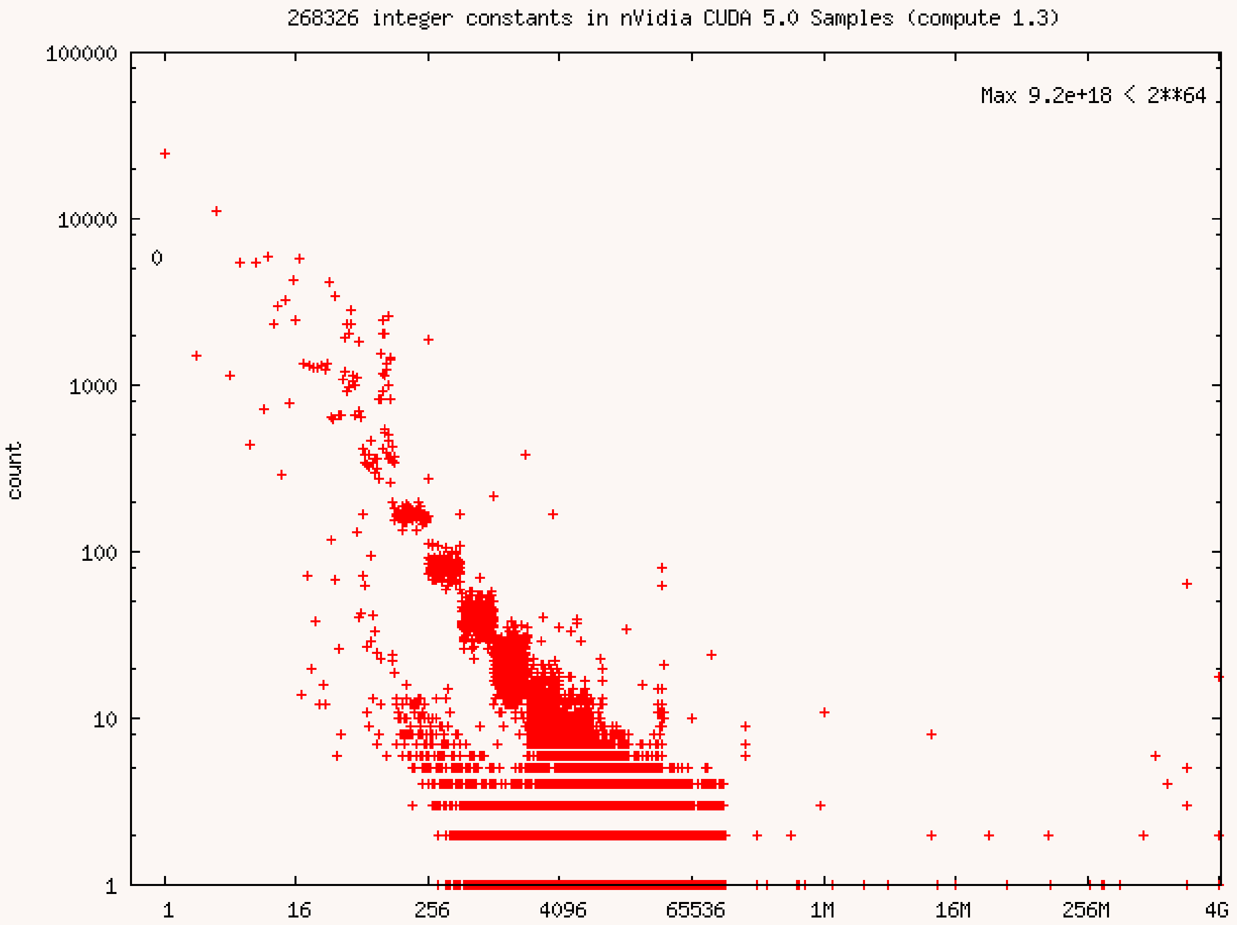
    Speed or memory or battery life.
- Trade off may be specific to particular use. For another use case re-optimise
- Use existing code as test “Oracle”. (Program is its own functional specification)

When to Automatically Improve Software

- Genetic programming as tool. GP tries many possible options. Leave software designer to choose between best.
- Port and optimise to new environment, eg desktop→phone (3D stereovision)

CREST

What's my favourite number?



The Genetic Programming Bibliography

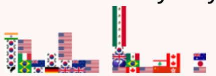
http://www.cs.bham.ac.uk/~wbl/biblio/

12825 references, 11000 authors

Make sure it has all of your papers!
E.g. email W.Langdon@cs.ucl.ac.uk or use | Add to It | web link

RSS Support available through the Collection of CS Bibliographies.

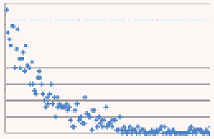
Downloads by day



Downloads

A personalised list of every author's GP publications.

blog



Your papers

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