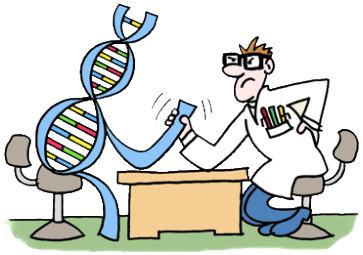
CREST

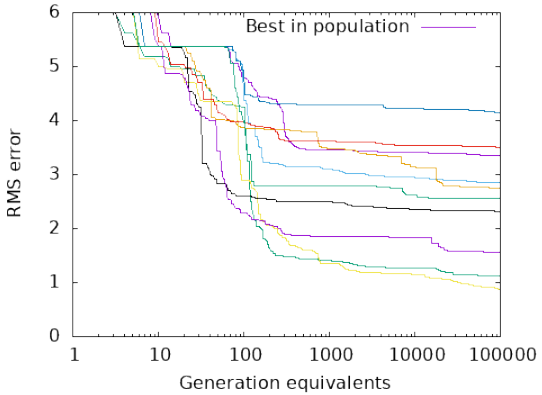
Improving Parallel Linear GP Interpreters

15th International Workshop on Genetic Improvement
Monday 13 April 2026



Humies $10000 prizes
Submit by **Friday 29 May**

W. B. Langdon, UCL



Genetic Improvement can automatically refine complex parallel C++ code that is difficult for human developers to optimise reliably.

Improving a Parallel C++ Intel SSE SIMD Linear Genetic Programming Interpreter, W.B. Langdon and Carol Hanna.

Improving a Parallel C++ Intel AVX-512 SIMD Linear Genetic Programming Interpreter, William B. Langdon, ArXiv 2512.09157.

Long Term Evolution Experiments with Linear Genetic Programming, W.B. Langdon.

```
for (int i=0;i<InstrLen;i++) loop
<  if(Instr[i][2]==div_op) {
>  if(Instr[i][2]>=div_op) {

In __m256i InstrReg16(const OP code, const retval reg[]) function
<       const __m256i mask     = _mm256_set1_epi32(255);
>       const __m256i mask     = _mm256_set1_epi32(-1);

In Interpret16 code for add sub mul using epi16
>    c = _mm256_mullo_epi16(a,b);
```

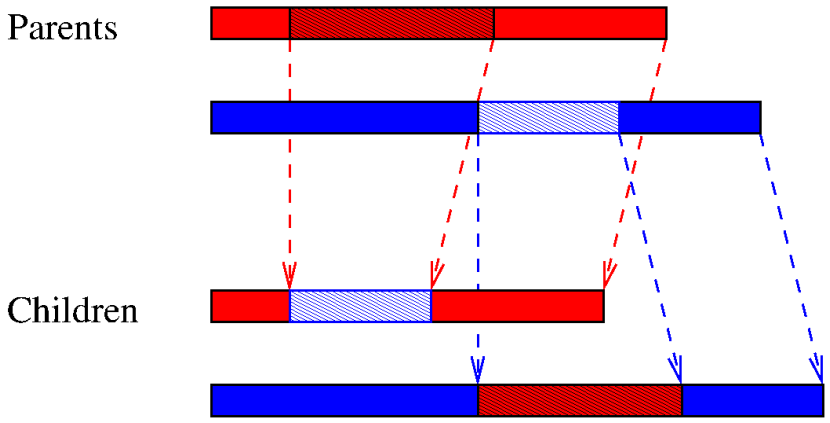
Best solution makes 3 Intel SIMD code changes: 1st ≥, 2nd and 3rd enable compiler optimisations

Improving a Parallel C++ Intel SSE SIMD Linear Genetic Programming Interpreter

- What is Linear Genetic Programming?
- What are Long Term Evolution Experiments?
- Long Term Evolution Experiments with Linear GP
  - W.B. Langdon, in Recent Advances in Linear Genetic Programming, Wolfgang Banzhaf and Ting Hu, Eds. 2026
- Need for speed
  - Intel SSE 256 bits
  - Intel AVX 512 bits
- Need for Genetic Improvement (Magpie)
- Linear Genetic Programming GPengine
  - https://github.com/wblangdon/GPengine
- All 100,000 generation runs ≤ 12 days

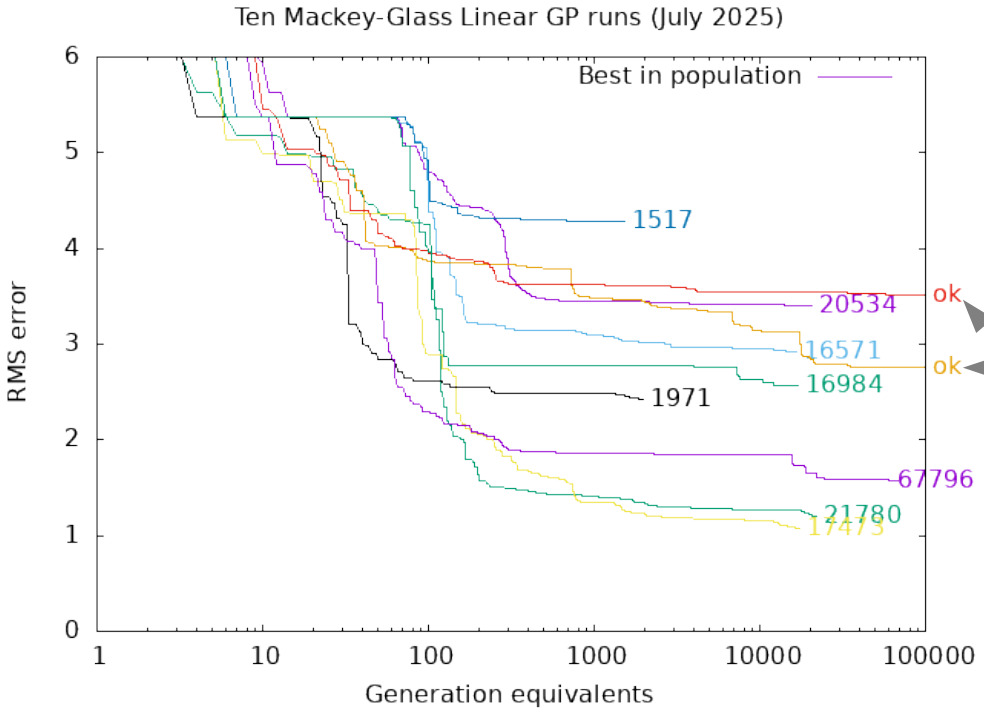
What is Linear Genetic Programming (GPengine)

- Here Linear GP has a population of 500 programs. Each is a variable length list of arithmetic operations on eight read-write registers (bytes). Eg R4=R1+119
- The operation are interpreted in order: start to end.
- Before execution the registers are loaded with the next test case. The program's output is collected from register 0 when it finishes
- No loop or branches. Max 4 million instructions. 1201 test cases.
- 4 programs are selected at random. Best two become parents for two children which replace the worst two.
- Children are created by crossover and mutation.



Long Term Evolution Experiments with Linear Genetic Programming

- In biology Long Term Evolution Experiment ran during whole of Rich Lenski's post doctoral career.
  - Shows bacteria continue to evolve even after 80,000 generations (38 years). HomoSapiens 9300 generations.
- Tree Genetic Programming run ≤ 1 million generations
- Goal Linear Genetic Programming 100,000 generations
- Reviewers complain not all runs reached 100,000



Due to time limits, only two runs finish

Need for Speed

- One GPengine run (worst case)
  - 100000 gens x pop=500 x 1201 tests x 4 million instructions = 240.2 10^{15} GP operations (7½ years, if 10^9 per second)
- Tried to run GPengine fitness in parallel using Intel 256 vector instructions. Manual coding proved hard and abandoned instead:
- GPengine C++ interpreter modified to use multiple treads.
- 24 threads still not enough.
- SSE 256 bit allows 32 test cases in parallel
  - SSE 256 initially available. Magpie optimised.
- AVX 512 bit instructions allows 64 test cases in parallel
  - AVX 512 not just longer but more complex
  - Magpie used again. Similar result.
  - Genetic Improvement result used in AVX version of GPengine
- Longest run 12 days AVX 512 and 19 cores

Need for Genetic Improvement

- Typically major performance bottle neck is fitness evaluation
- Run interpreter in parallel across 1201 fitness cases, not enough
- AVX 512 each instruction may process data for 64 fitness cases
  - with multi-core 1201/64 => 19 threads
- What is best way to work round single instruction multiple data?
  - No protected division
  - No byte level multiply
- Hand code best guess, use Genetic Improvement to optimise
- Three GI experiments (each on interpreter only):
  - SSE 256 (byte)
  - AVX 512 (multiple data types, i.e. byte, 16 bit and 32 bit)
  - AVX 512 (best type only, ie byte)
- Manually apply best GI solution to GPengine

Magpie

https://github.com/bloa/magpie

- Magpie tutorials GI@ICSE 2024, 2025, 2026
- Multiple experiments (similar best solutions):
  - SSE 256 bit
  - AVX 512 bit, three types: byte, short int, 32bit int
  - AVX best type (unsigned char)
- Automatically convert C++ to XML
- Magpie mutates XML and then converts to C++, compiles, runs
- Magpie's scenario file
  - Local search
  - max_steps = 100000 (< 22 hours)
  - All applicable XML edits
  - (3 types experiment only include regsize {8,16,32}[8] avx.param)
- 5 runs for each experiment

GPengine interpreter

- A few weeks of manual effort to create SSE 256 bit and then AVX similar effort for 512 SIMD version of interpreter (64 x bytes).
- Support: 8 bit + - * protected division, eight read-write registers
- No 8 bit multiply.
- No integer division:
  - Use float division, protect v. divide by zero, convert back to int?
  - Use 256x256 look up table
- Genetic Improvement to choose/tune many sequential and AVX options
- Extract human code (eval.cpp). srcml => XML eval.cpp.xml
- Magpie updates XML, converts to mutated C++, tries to compile. Fitness from running mutant on test cases.
- Magpie reduces run time (measured by Linux perf)

Multiple XML Ingredients plus eval.cpp.xml

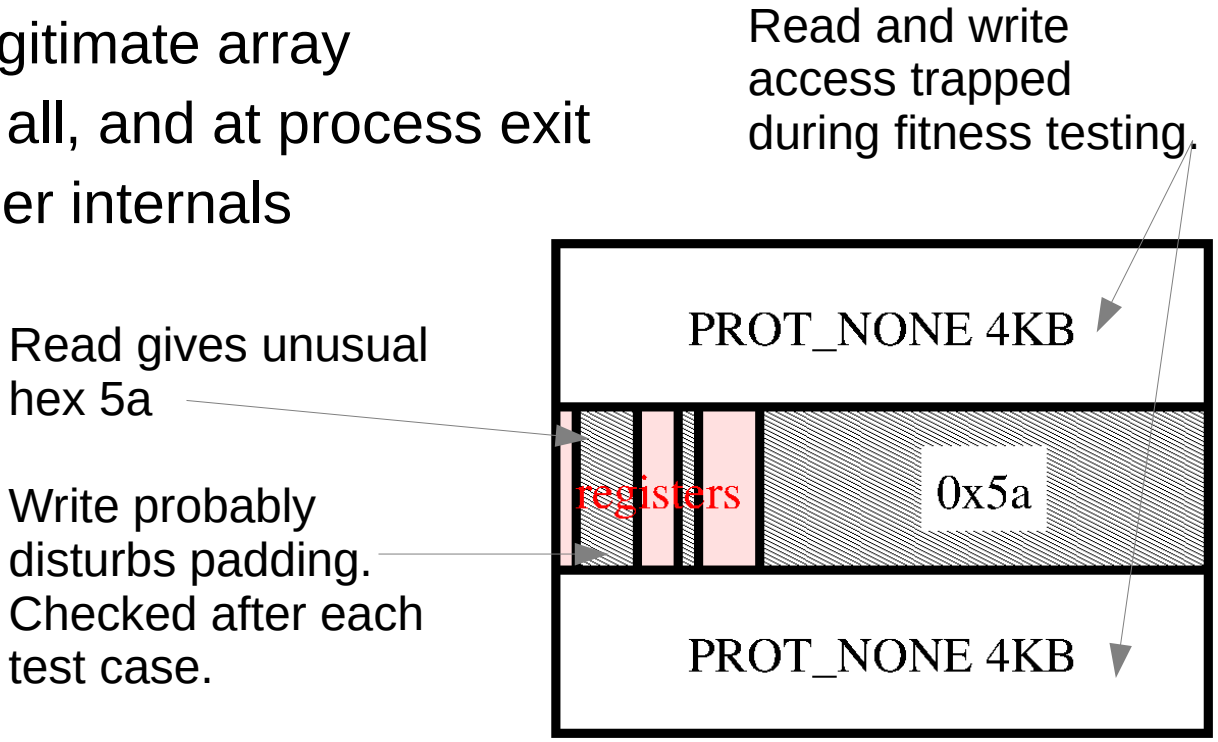
- Tried to help evolution by also giving it access to
  - History of manual changes diffs.cpp.xml
  - Intel documentation for AVX library IntrinsicsGuide.cpp.xml
- Magpie can incorporate XML from multiple sources but
  - often failed to compile, eg identifier mismatches
  - never included in evolved solution

Fitness: Compilation

- GCC 11.5.0 C++ compiler error message gives error messages: did you mean xyz?
  - Automatic replacement of failing code with suggested replacement gave another compilation error. Abandoned.
  - Some evolved code would pass -O3 but fail without it. So:
  - Fitness run without -O3. If ok compile with -O3 and time it.
- AVX512 example for unsigned char (ok without -O3)
  - g++ -c -fmax-errors=1 -fpermissive -fconcepts -march=skylake-avx512 -DAVX **eval_h.cpp** -O3 -o eval_h_8.o -Dregtype8 -Dregtype=uint8_t
  - g++ -g -o eval_perf.exe eval_h_8.o eval_perf_8.o
  - eval_h.cpp wrapper for #include "**eval.cpp**"
  - eval_perf.cpp main(), test cases, sandbox, collect timing
- Check object file eval_h_8.o for equivalent mutations

Fitness: Sandbox mprotect

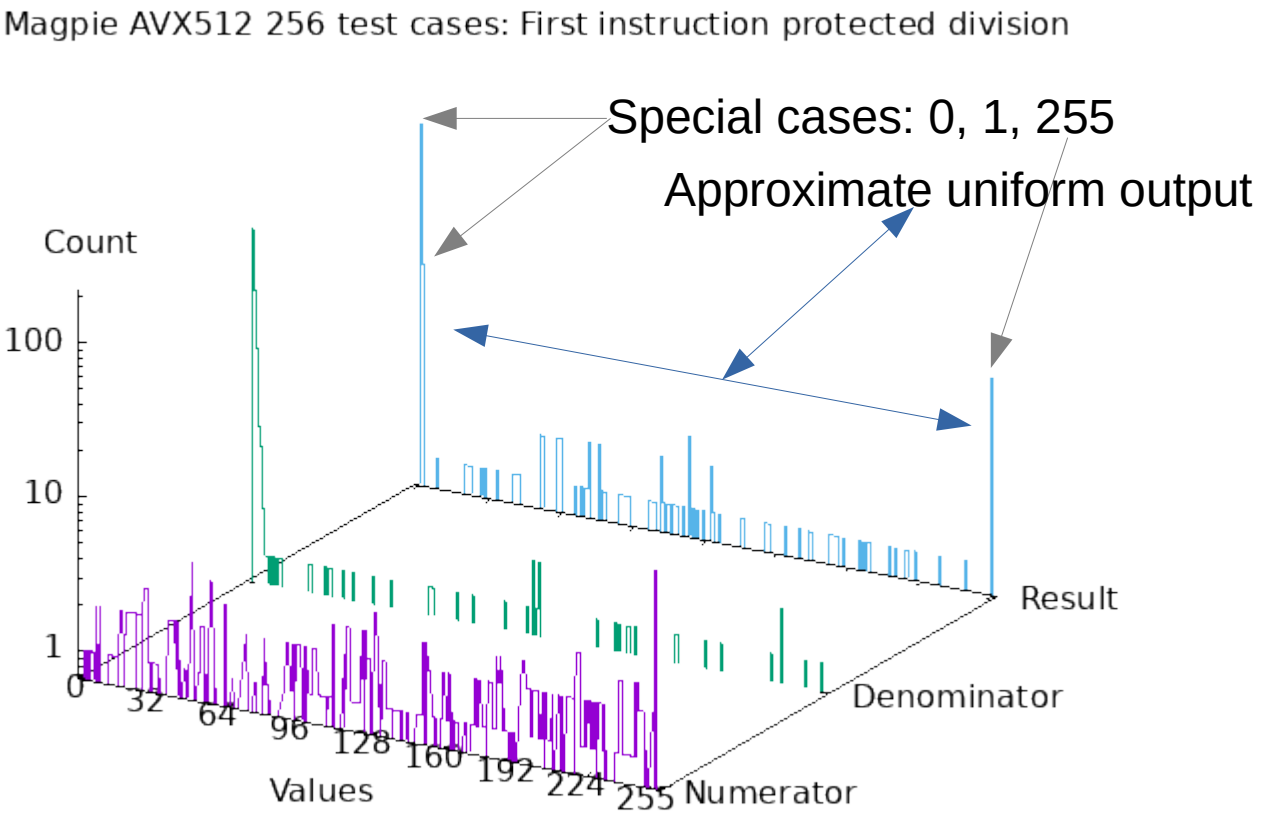
- In C++ access outside array is undefined
- GISMOE had sometimes wrapped arrays in bound checking
  - Bound checking typically changes timing
- Linux mprotect (hardware, no run time overhead)
  - Write protect read-only data
  - Protect “near” legitimate array
  - Tricky to protect all, and at process exit
  - Assumes compiler internals



Fitness Test Cases

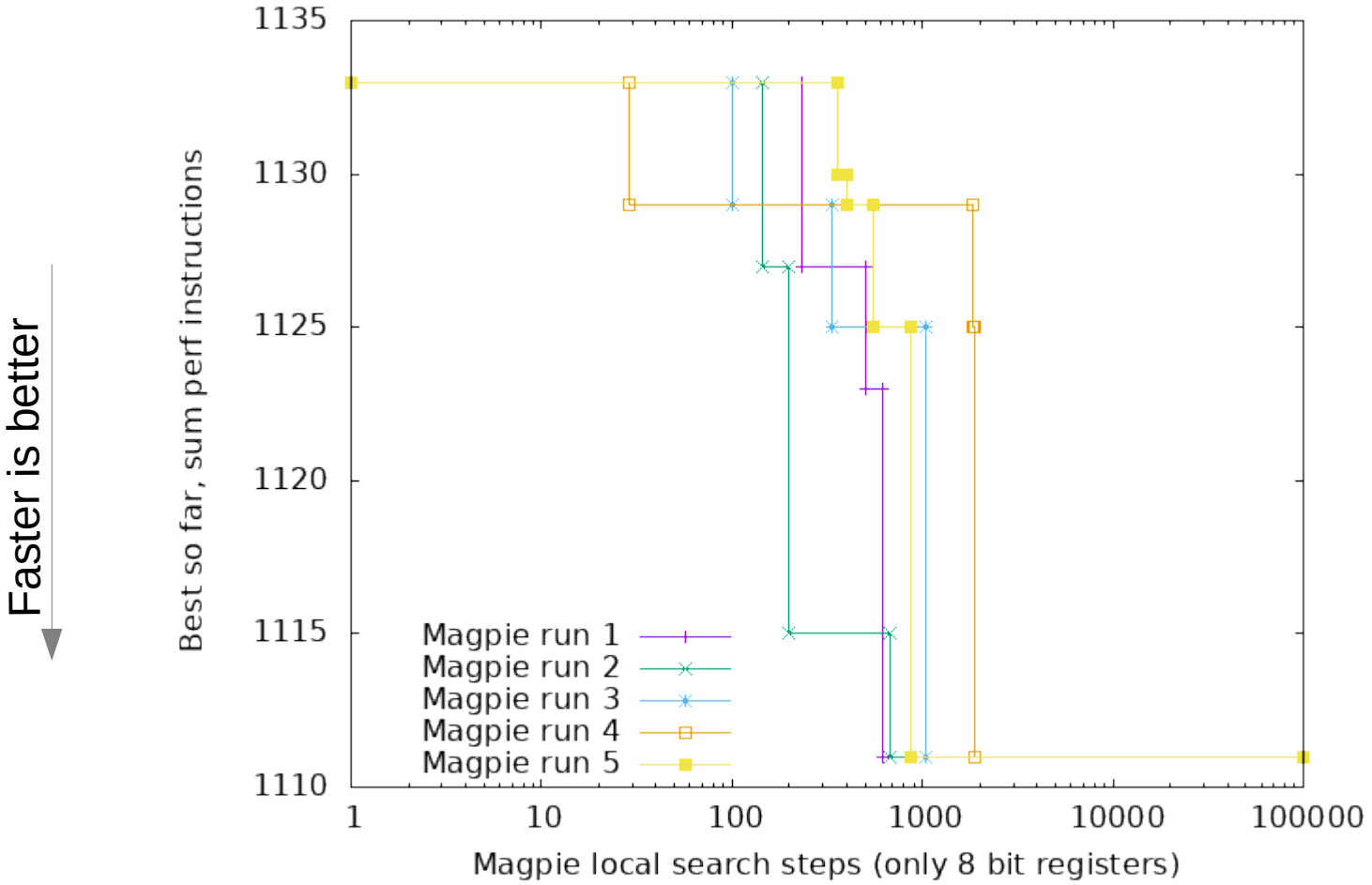
- Four small programs to be interpreted created at random each with 64 random test case to processed in parallel.
- If mutant passes all 4x64, fitness is speed (ie sum perf instruction counts)
- Ensure programs only read data that has been written to
- Ensure at end output register R0 is set
- Ensure add, sub, multiply, protected division are all used
- Protect div most complicated, so especial care with it, randomly test edge cases, uniform *output.*
  - Programs start with protected division
  - 50% divide by zero
  - 25% 0, 1 or 255
  - 25% uniform output (2..127)
- Otherwise randomly choose registers or constants

256 Fitness Test Cases



Result = Numerator / Denominator

Magpie Runs



AVX 512
unsigned char

Five runs find same solution.

Evolved AVX512 unsigned char patch

- 5 runs found same patch: 3 independent changes.
- InstrArg32 and InstrReg32 disable mask to force 16bits to 8bits.
- ≥ ok as div_op last legal opcode.

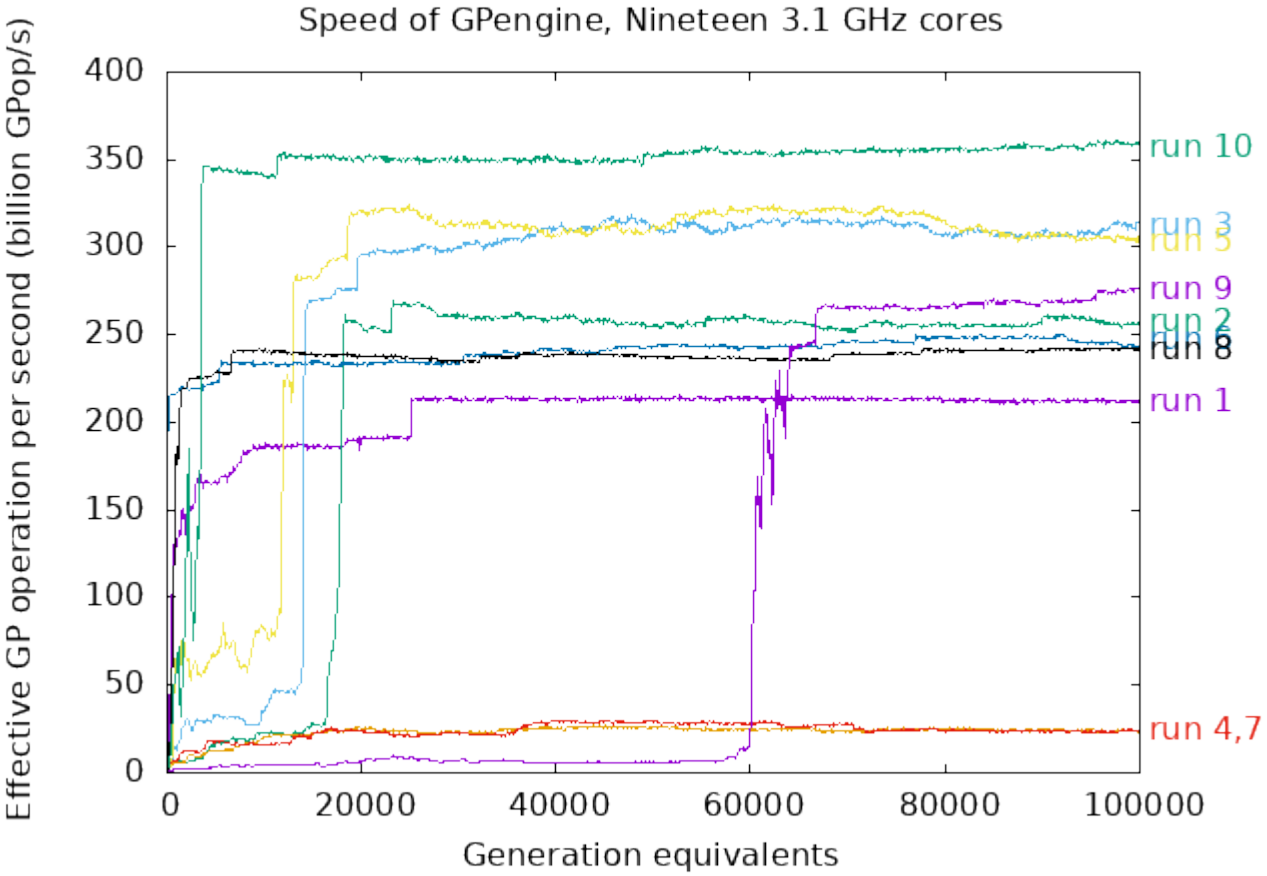
```
In __m512i InstrArg32(const OP code, const uint8_t registers[], const int j)
        const __m256i a    = _mm256_loadu_epi8(&registers[x*npar+j]);
        const __m512i aa   = _mm512_cvtepi8_epi16(a);
        const __m512i zero = _mm512_set1_epi32(0);
-       const __mmask64 k  = 0xaaaaaaaaaaaaaaaa;
+       const __mmask64 k  = 0;
        return _mm512_mask_blend_epi8(k, aa, zero);

In __m512i InstrReg32(const OP code, const uint8_t registers[], const int j)
        const __m256i a    = _mm256_loadu_epi8(&registers[x*npar+j]);
        const __m512i aa   = _mm512_cvtepi8_epi16(a);
        const __m512i zero = _mm512_set1_epi32(0);
-       const __mmask64 k  = 0xaaaaaaaaaaaaaaaa;
+       const __mmask64 k  = 0;
        return _mm512_mask_blend_epi8(k, aa, zero);

 void Interpret64(const int InstrLen, const instr *Instr, regtype registers[
NumVar*npar], const uint32_t div32[256*256]) {
 for (int i=0;i<InstrLen;i++) {
-  if(Instr[i][2]==div_op) {
+  if(Instr[i][2]>=div_op) {
```

AVX 512 version GPengine

- Apply evolved change to GPengine.cpp :
  - Make interpreter use unsigned char
  - Replace == by >=
  - In InstrArg32() and InstrReg32() remove masking 16 bit to 8 bits
    - Mask was included as AVX does not support equivalent byte operation.
    - Mask removal ok as following AVX multiply code works with 16 bit values and already forces them 8 bit before output.
- GPengine.cpp used in 100000 generation runs and gives the same answers.
- GPengine interpreter peak speed 3.5 billion GP operations per second per thread (3.8 GHz desktop).
- Mean effective speed for whole program 23 to 349 billion GP operations per second (3.1 GHz server) 19 threads.

GPengine 3.1GHz AMD EPYC 9554

- Evolved interpreter peak speed 3.5 billion GPop/sec per thread.
- Sustained effective speed (include overheads, exclude unused ops) 23 to 349 billion GP/sec

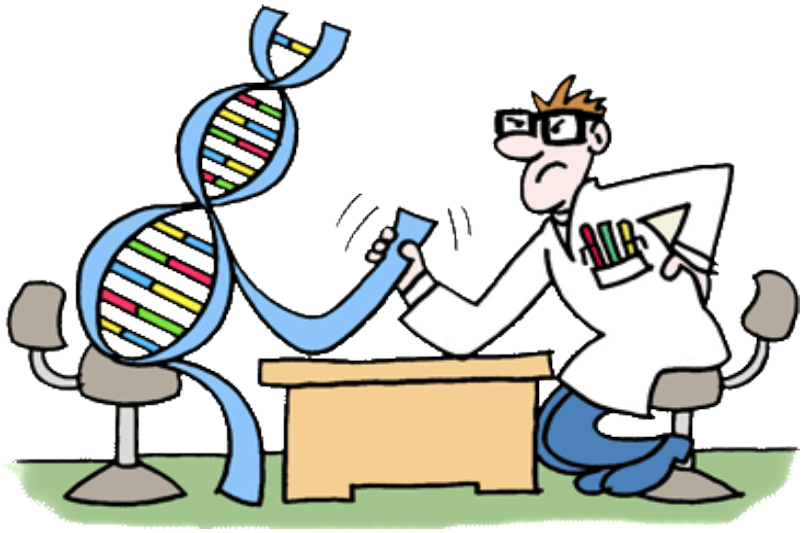


Conclusions

- Magpie can automatically refine complex code that is difficult for people to tune (weeks).
- It produced small, compact, non-trivial, comprehensible, correct and reusable patches (< 1 day).
- Linux mprotect can help sandbox mutated C++ code.
- Linux perf instruction count gave stable fitness measurement.
- The three independent Genetic Improvement patches were applied with no change in functionality.
- GIed GPengine completed all runs before deadline.

```
for (int i=0;i<InstrLen;i++) loop
<  if(Instr[i][2]==div_op) {
>  if(Instr[i][2]>=div_op) {

In __m256i InstrReg16(const OP code, const retval reg[]) function
<       const __m256i mask     = _mm256_set1_epi32(255);
>       const __m256i mask     = _mm256_set1_epi32(-1);

In Interpret16 code for add sub mul using epi16
>    c = _mm256_mullo_epi16(a,b);
```

Code on https://github.com/wblangdon/GPengine_eval_AVX512 etc.

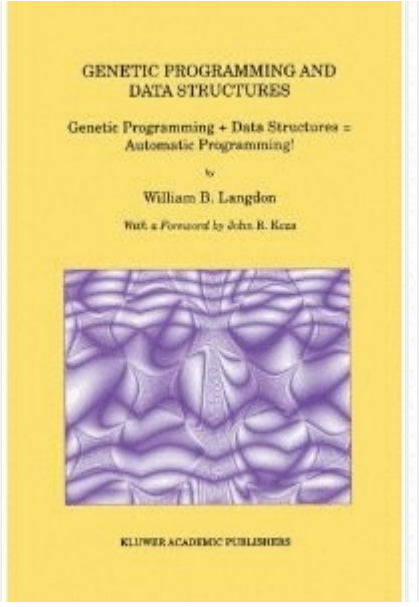


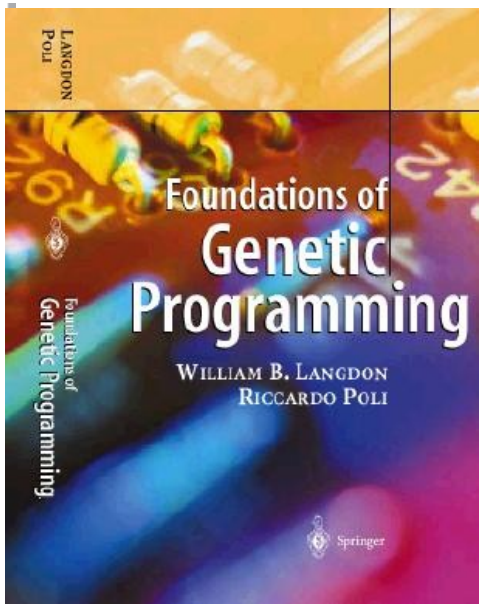
Human-Competitive results $10,000 prizes
https://www.human-competitive.org

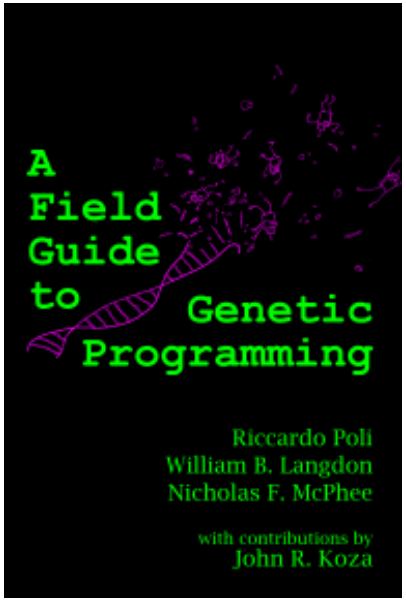
Email your entry to goodman@msu.edu
by **Friday 29 May**

Genetic Programming

W. B. Langdon







Magpie AVX512 Parameters

```
[magpie]
show_cmd_progress = False

[magpie.log]
color_output = False

[srcml]
rename =
    stmt: break continue decl_stmt do
expr_stmt for goto if return switch while
    number: literal_number
focus = block stmt operator_comp
operator_arith number
internodes =
process_pseudo_blocks = False
process_literals = True
process_operators = True
```

```
[software]
path = examples/code/avx
target_files =
    eval.cpp.xml
ingredient_files =
    eval_diffs.cpp.xml
    diffs.cpp.xml
    IntrinsicsGuide.cpp.xml
fitness = outputcompile_cmd = tcsh ./
compile.bat
test_cmd =
run_cmd  = tcsh ./fit.bat

[search]
max_steps = 100000
possible_edits =
    SrcmlArithmeticOperatorSetting
    SrcmlComparisonOperatorSetting
    SrcmlNumericSetting
    SrcmlRelativeNumericSetting
    SrcmlStmtDeletion
    SrcmlStmtInsertion
    SrcmlStmtReplacement
    XmlNodeDeletion<stmt>
    XmlNodeInsertion<stmt,block>
    XmlNodeReplacement<stmt>
    XmlNodeReplacement<number>
```

Hardware

- Training

  AVX512 Intel 24 Core i7-9800X CPU 3.80GHz
- Production
  - AVX512 AMD EPYC 9554 256 Core Processor 3.1 GHz

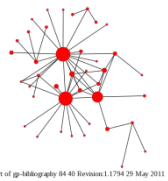
The Genetic Programming Bibliography

http://gpbib.cs.ucl.ac.uk/

18476 references, 19000 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

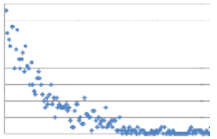


Co-authorship community.
Downloads

Downloads by day

A personalised list of every author's GP publications.

blog



Your papers

Text search

Googling GP bibliography, eg:
Development and learning site:gpbib.cs.ucl.ac.uk