CREST

Grow and Graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation

Genetic Improvement workshop GECCO 2015

12th July 2015

W. B. Langdon

Department of Computer Science

UCL

Grow and Graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation

W. B. Langdon
Department of Computer Science
UCL

GGGP and CUDA pknotsRG

- CUDA pknots

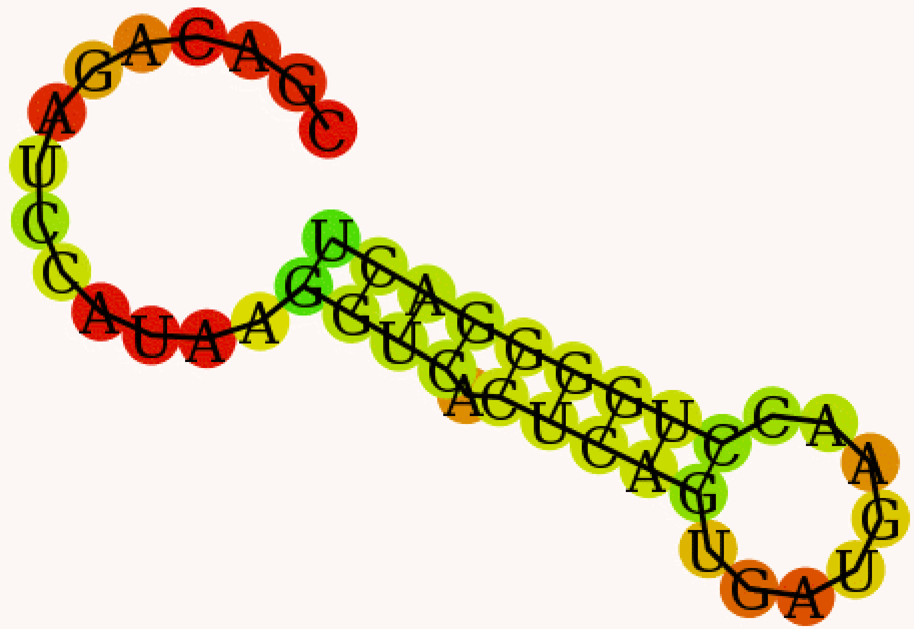
  Calculate shape of RNA molecules

  11 000 lines of C and CUDA code
- Grow and Graft Genetic Programming
- 10 thousand fold speedup

pknots:
RNA sequence → folding

Input ⟶ CGACAGAUCCAUAAGGUCACUCAGUGAUGAACCUGGGGACU

Output ↗ ..............((((.(((((........)))))))))

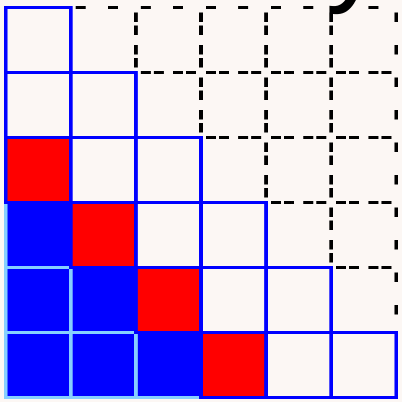


free energy of -9.30 kcal/mol

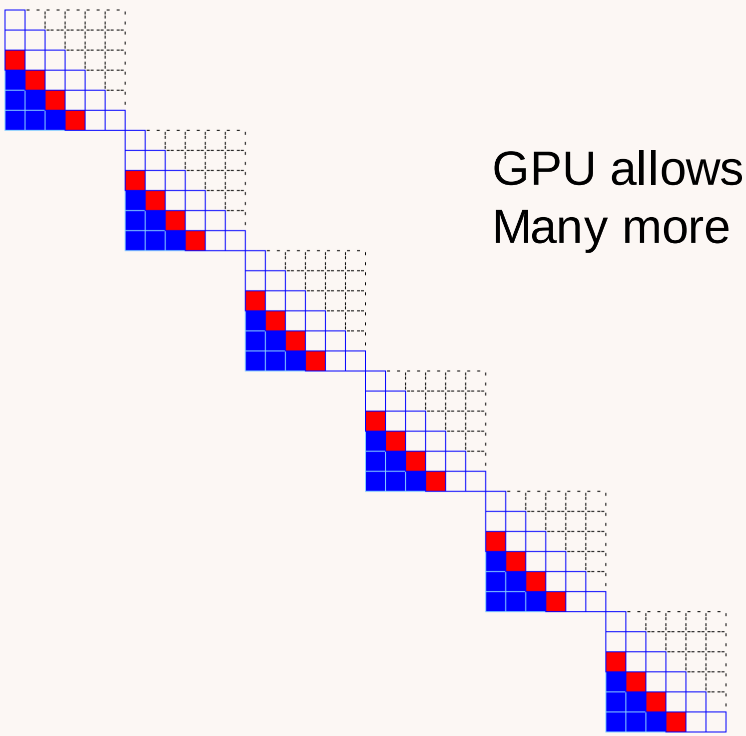
Dynamic programming

- pknots uses dynamic programming to find minimum energy of folding
- One molecule at a time
- Not enough parallelism
- Run Dynamic Programming algorithm on 200,000 matrices in parallel
- GGGP convert CUDA from 1 to n matrices

Dynamic programming



(n+1) by (n+1) matrix.
Only lower half used.
Active front can be calculated in parallel
(needs 1 to n+1 threads)



GPU allows many matrices to be calculated in parallel
Many more GPU threads can be used.

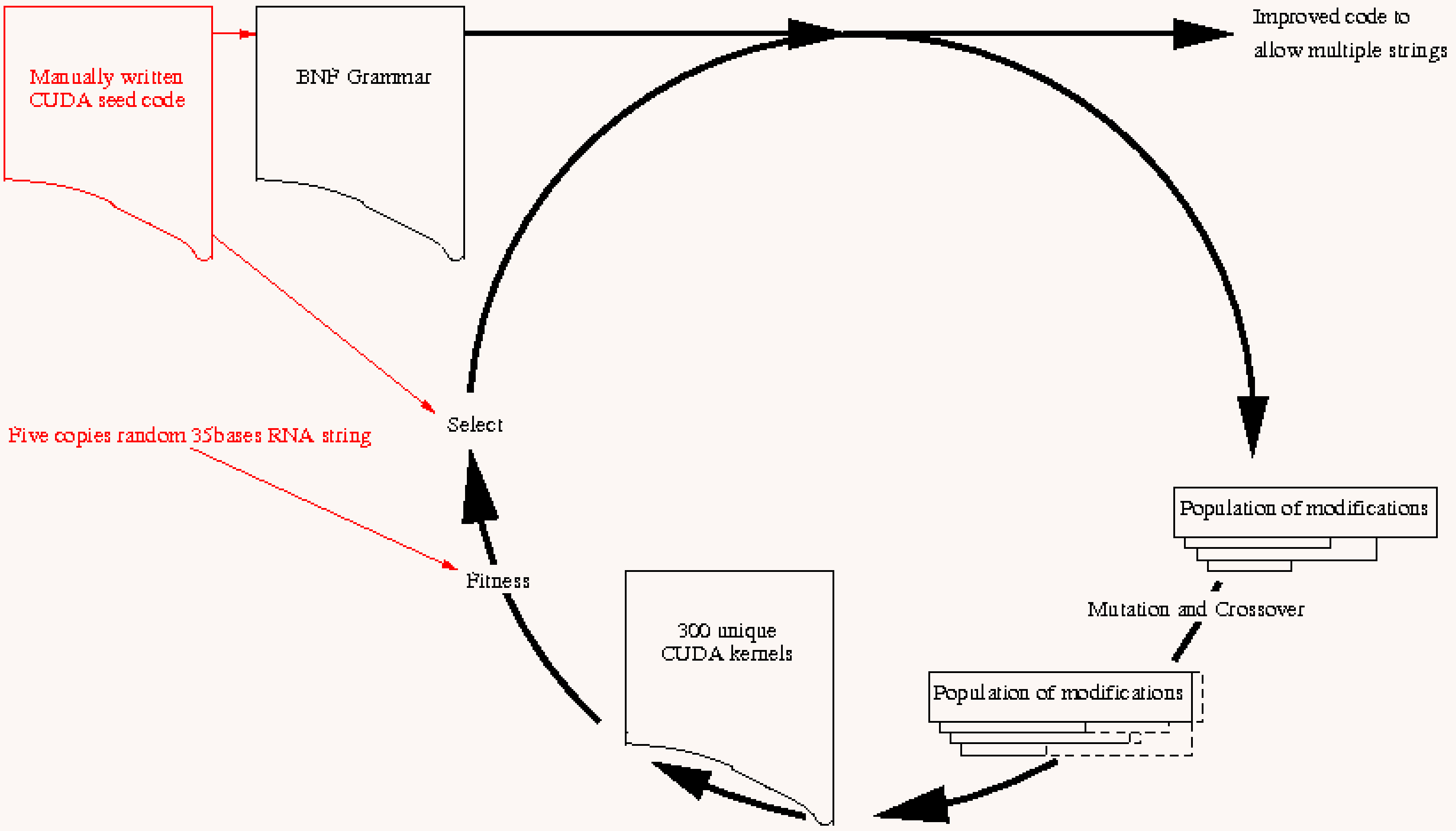
Evolving pknotsRG kernel

- Convert manual CUDA code into grammar
- Grammar used to control code modification
- GP manipulates patches
  - Small movement/deletion of existing code
  - New program source is syntactically correct
  - all mutants compile
  - No loops so kernels always terminate
- GP continues despite runtime errors
- Fitness by testing and comparing against outputs and speed of original code

Preparing for Evolution

- GP fitness testing framework
  - Generate and compile 300 unique mutants
    - Whole population in one source file
    - All mutants compile
    - Run and measure speed of 300 kernels
    - Reset GPU following run time errors
  - For each kernel check 5×36 answers

Evolving pknotsRG



CREST

BNF Grammar

```
 const int gggp_I =
gggp_thread
/
gggp_x
;
```

CUDA lines 54-57 (initialise gggp_I)
(Variables beginning gggp are part of seed)

```
<Kkernel_bnf.cu_54>         ::= "const int gggp_I =\n"
<Kkernel_bnf.cu_55>         ::= <gggpint_Kkernel_bnf.cu_55> "\n"
<gggpint_Kkernel_bnf.cu_55>::= "gggp_thread"
<Kkernel_bnf.cu_56>         ::= "/\n"
<Kkernel_bnf.cu_57>         ::= <gggpint_Kkernel_bnf.cu_57> "\n"
<gggpint_Kkernel_bnf.cu_57>::= "gggp_x"
<Kkernel_bnf.cu_58>         ::= ";\n"
```

Fragment of Grammar (Total 104 rules)

Representation

- variable length list of grammar patches.
  - no size limit, so search space is infinite
- tree like 2pt crossover.
- mutation adds one randomly chosen grammar change
- 3 possible grammar changes:
  - Delete line of source code
  - Replace with line of GPU code (same type)
  - Insert a copy of another line of kernel code

Example Mutating Grammar

```
const int gggp_I =  gggp_thread/gggp_x;
```

```
<gggpint_Kkernel_bnf.cu_57>        ::=    "gggp_x"
<gggpint_Kkernel_bnf.cu_128>       ::=    "gggp_y"
```

2 lines from grammar

```
<gggpint_Kkernel_bnf.cu_57><gggpint_Kkernel_bnf.cu_128>
```

Fragment of list of mutations

Says replace line 57 by copy of line 128

```
const int gggp_I =  gggp_thread/gggp_y;
```

New code

Testing kernel variants

- Apply 300 GP patches (plus original)
- Compile specifically for GPU in use.
- Run on 5 random RNA sequences
  - 35 bases long enough to fold, at least $O(n^3)$
  - copies of fixed sequence sufficient
- Calculate time taken and check 5×35 intermediate answers.
- Only those returning correct answers quicker than manual code can breed.

Breeding kernel variants

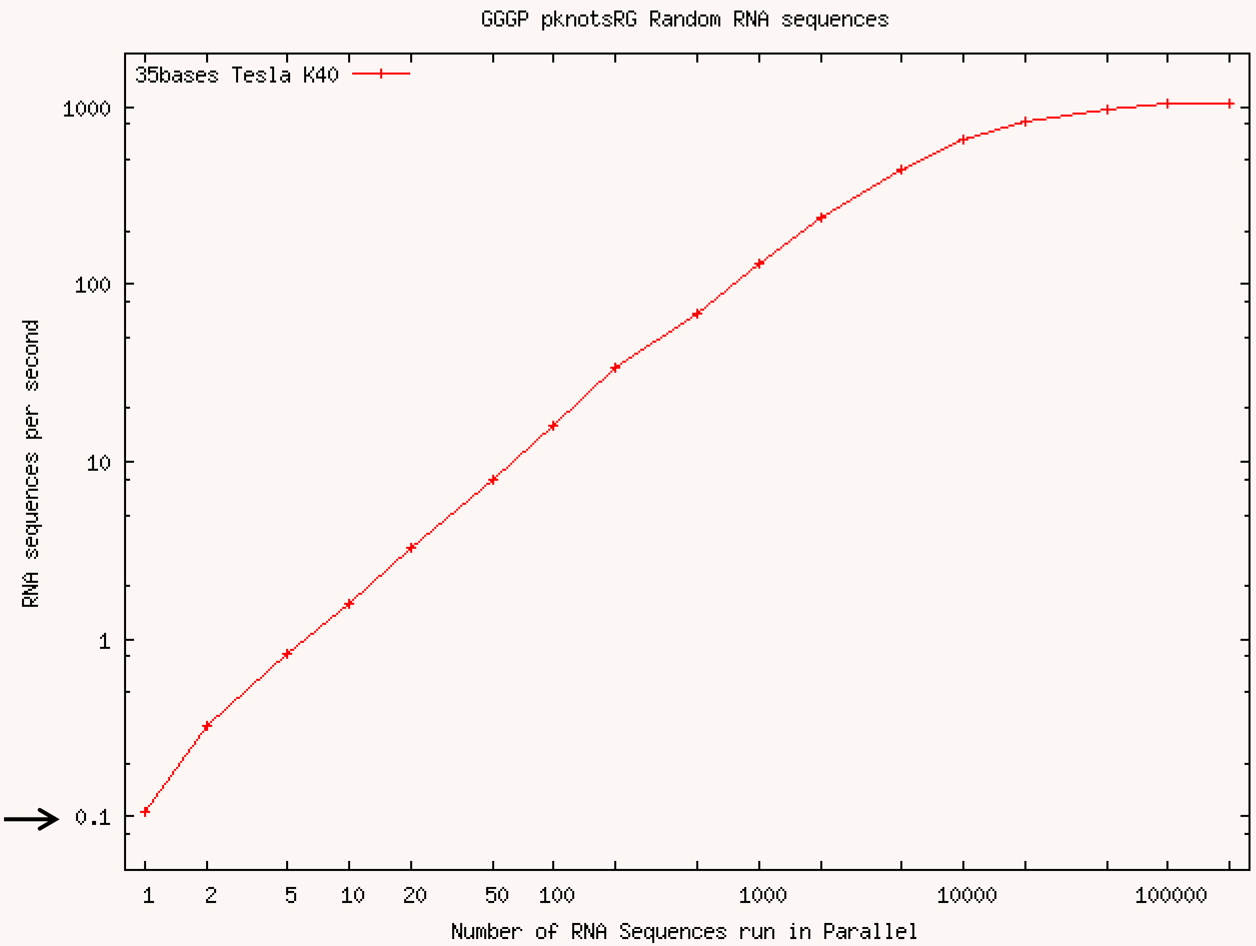
- Only mutants returning correct answers faster than manual code can breed.
- Choose fastest 150 to be parents.
- Mutate, crossover: 2 children per parent.
- Repeat 50 generations.
- Actually 2nd variant (with `diag`) solutions found in first generation.

Chosen Solution

```
const int gggp_I =  gggp_thread/gggp_x;
const int gggp_I =  gggp_thread/gggp_y;
```

- Several solutions in first random population. This one chosen as clearest
- Fixes error in manual seed code.
  - `gggp_I` indicates which matrix a thread is working on.
  - `x` is size of individual matrix.
  - Error is size of active front changes during run. `x` gives location of active front but this is also length of active front and so replacing `x` by `y` gives correct value for `gggp_I`

GGGP CUDA pknotsRG speed



Original →

Up to 10000 times faster

GP extrapolates from 5 examples to hundreds of thousands and from 35 bases to hundreds of bases.

Conclusions

- Genetic programming and human work together.
- On 11 000 lines of C/CUDA code
- Gives spectacular speed up
- ftp.cs.ucl.ac.uk/genetic/gp-code/pknotsGI.tar.gz

Talk GP3 Tuesday 11:35 Patio 2

END

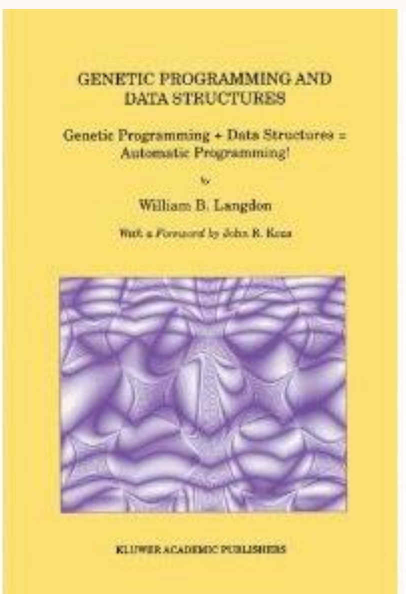
http://www.cs.ucl.ac.uk/staff/W.Langdon/ http://www.epsrc.ac.uk/

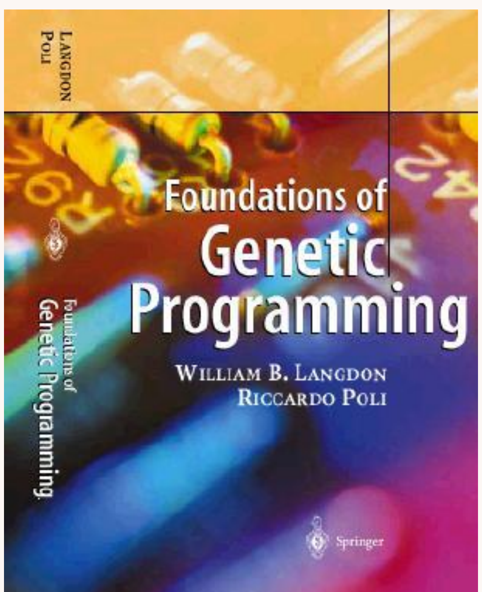
Genetic Improvement

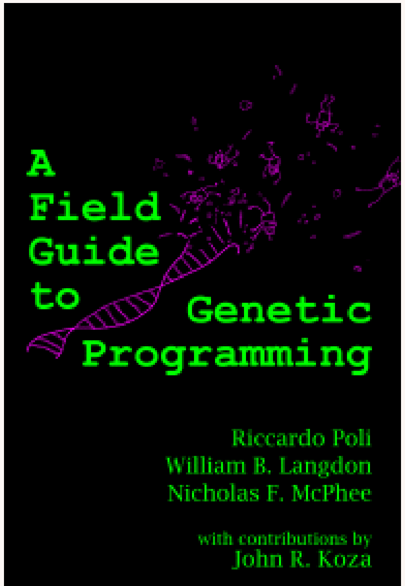
[W. B. Langdon]

CREST

Department of Computer Science



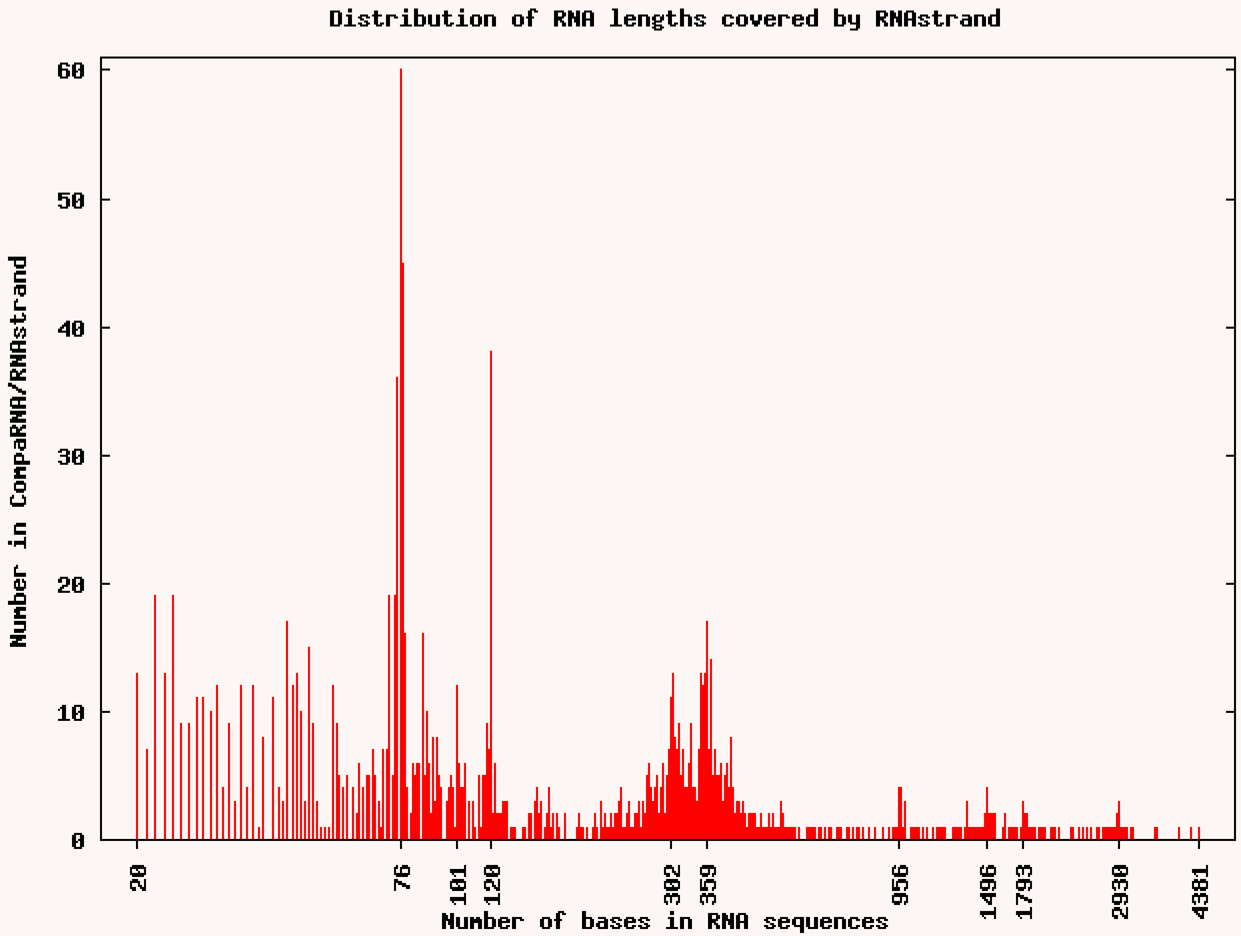




GPUs

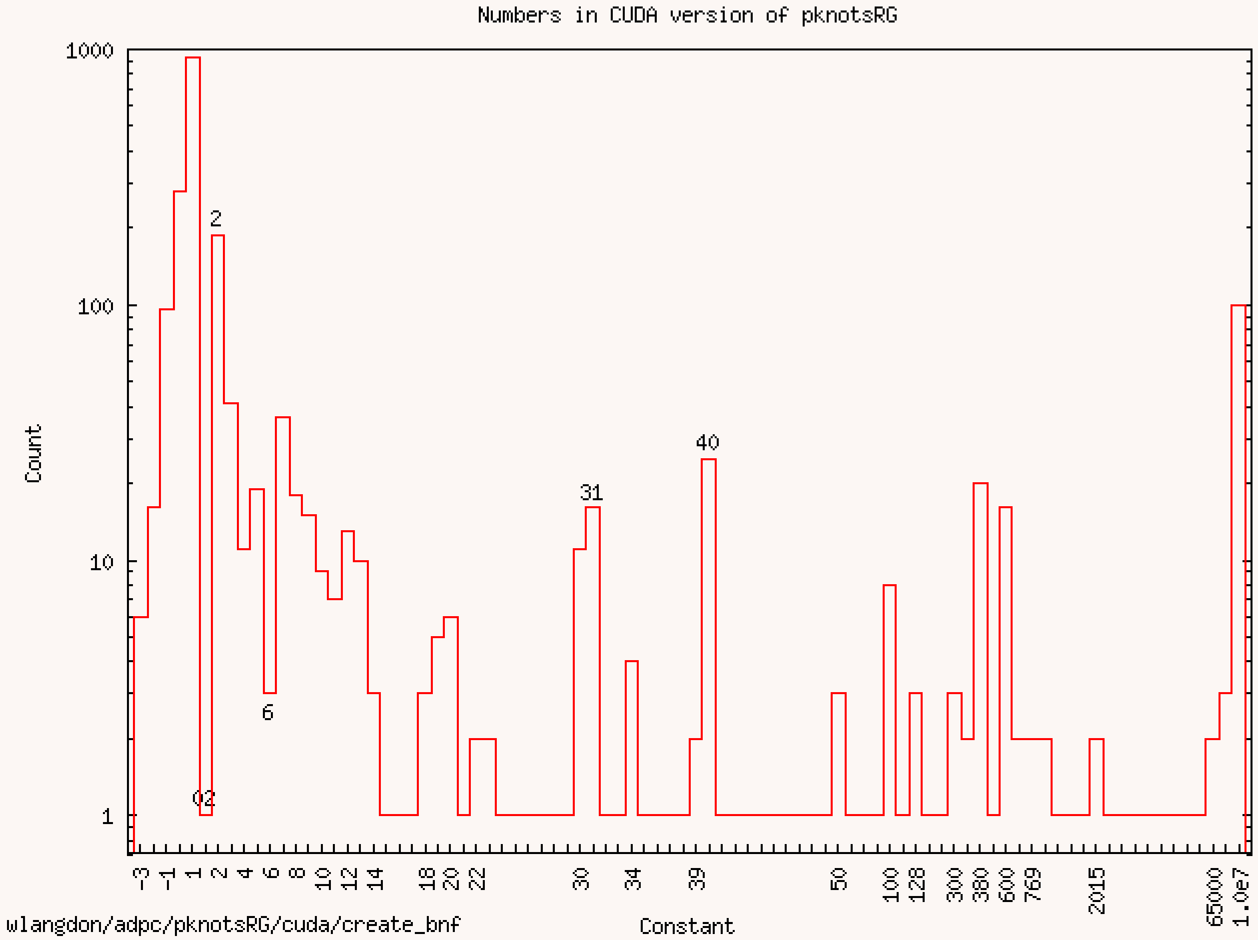
GPU	Total cores	clock	Bandwidth Giga Bytes/sec
Tesla K20	2496	0.71 GHz	140
Tesla K40	2880	0.88 GHz	180

Lengths of RNA in CompaRNA

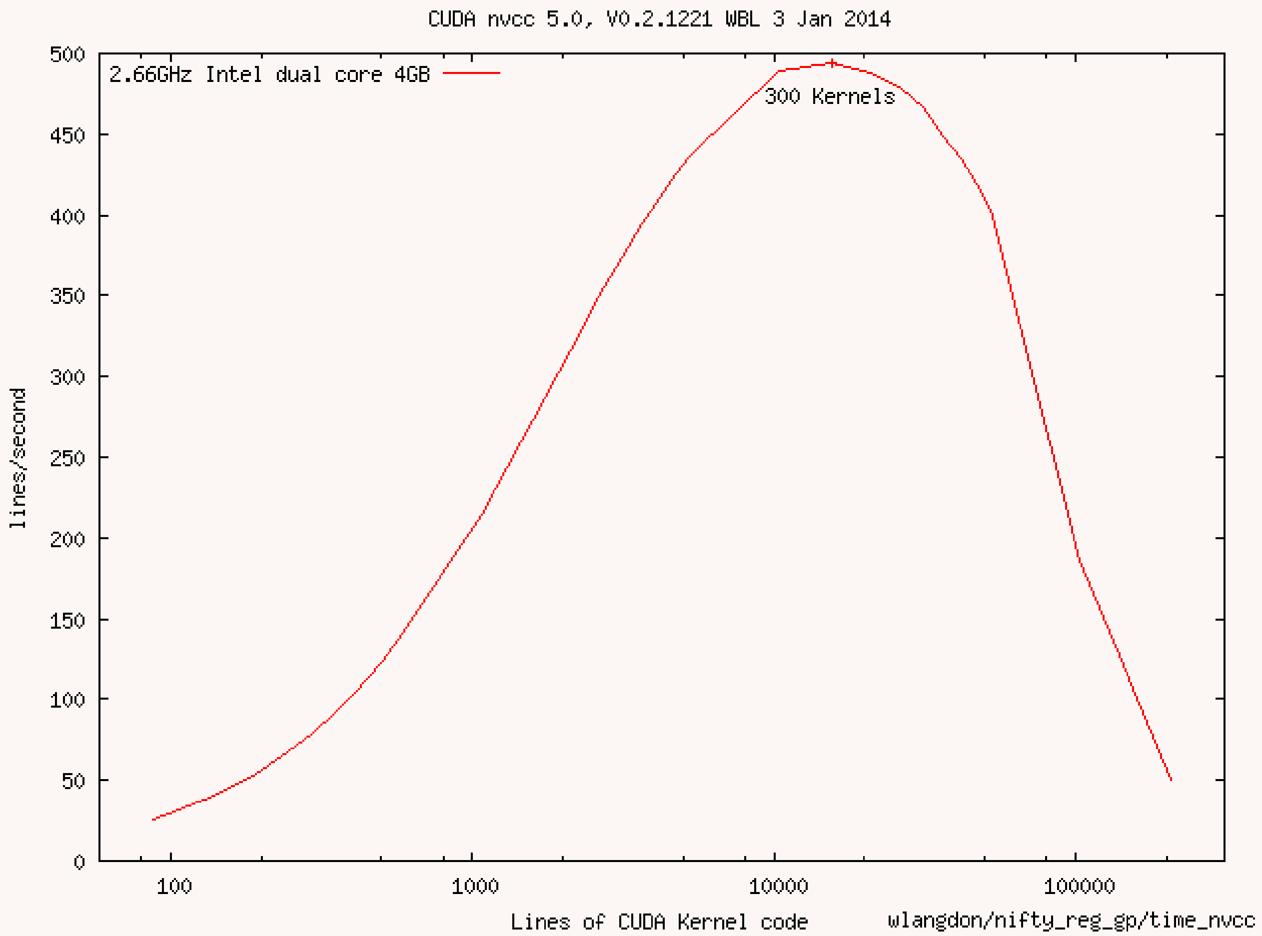


Number of bases in RNA sequences

Constants in pknotsRG sources



Compile Whole Population



Note Log x scale

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

4 Types of grammar rule

- Type indicated by rule name
- Replace rule only by another of same type
- 77 fixed, 27 variable.
- 17 int variables
- 6 small int constants

  0

  1

  2
- 3 optconst
  - potential compiler efficiency gains by declaring arguments as const
- 1 CUDA special __restrict__

GP Evolution Parameters

- Pop 300, 50 generations
- 50% crossover: two point crossover on variable length list of code patches
- 50% mutation:add a random patch to variable list.
- Truncation selection

Conclusions

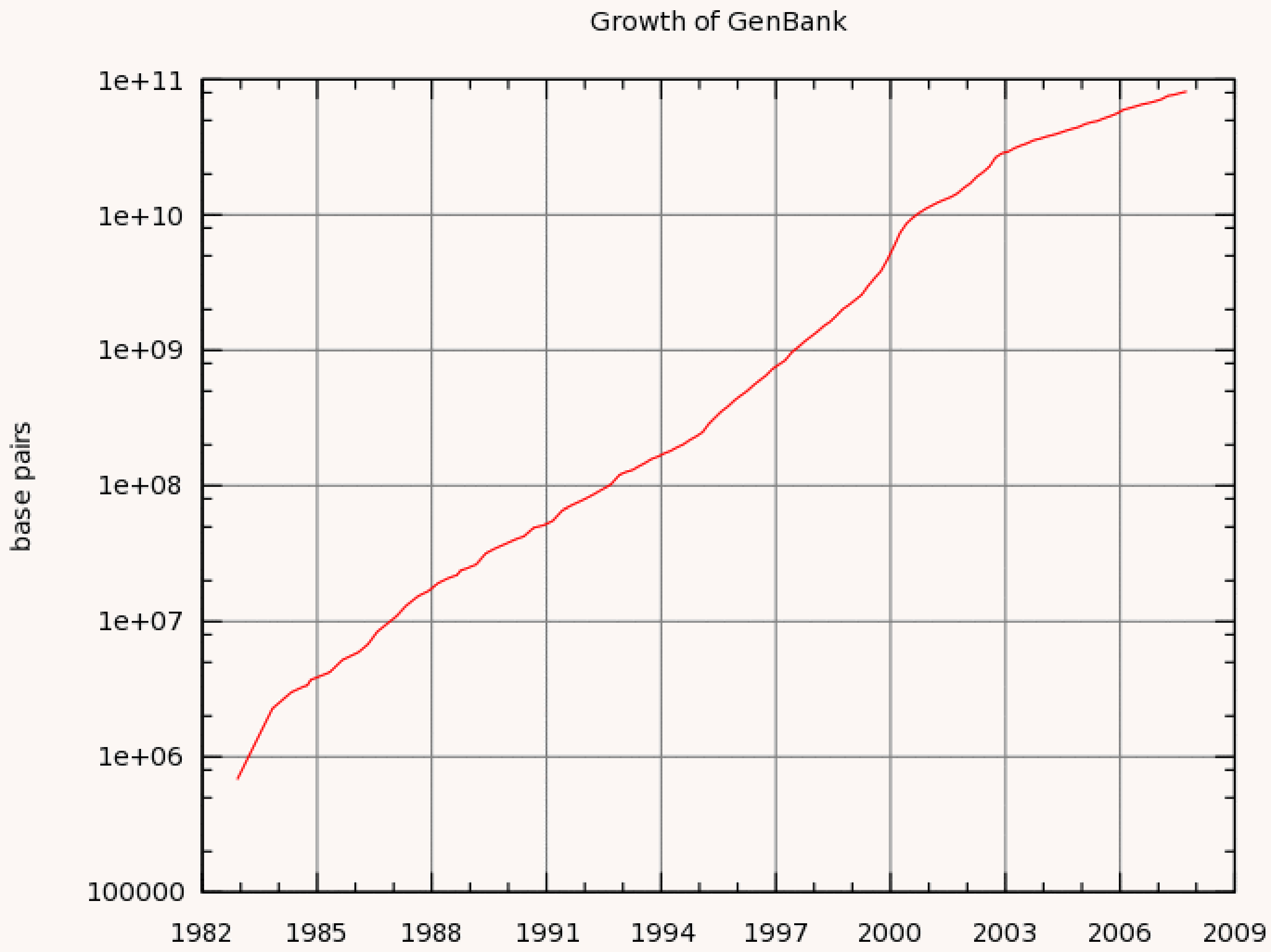
- Genetic programming can automatically re-engineer source code. E.g.
  - hash algorithm
  - Random numbers which take less power, etc.
  - mini-SAT (Humie award)
- fix bugs ($>10^6$ lines of code, 16 programs)
- create new code in a new environment (graphics card) for existing program,gzip WCCI '10
- 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

GP Automatic Coding

- Use existing code as test “Oracle”. (Program is its own functional specification)

“Moore’s Law” in Sequences



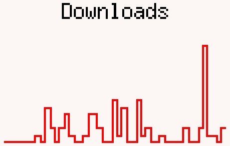
The Genetic Programming Bibliography

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

10318 references

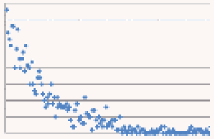
RSS Support available through the Collection of CS Bibliographies. XML RSS

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