



# Interpreting a genetic programming population on an nVidia Tesla

[W. B. Langdon](#)

CREST lab,  
Department of Computer Science



# Introduction

- General Purpose use of GPU (GPGPU) and why we care
- Evolutionary algorithms on graphics hardware designed for a single program operating on many data in parallel
- Simultaneously running  $\frac{1}{4}$  million programs
- Actual speed 212 billion GP ops /second
- Lessons

# Why Interest in Graphics Cards

- Speed
  - 800 0.8Ghz CPUs
  - Even with multi-threading off-chip memory bottleneck means difficult to keep CPUs busy
- Future speed
  - Faster than Moore's law
  - nVidia and AMD/ATI claim doubling 12months

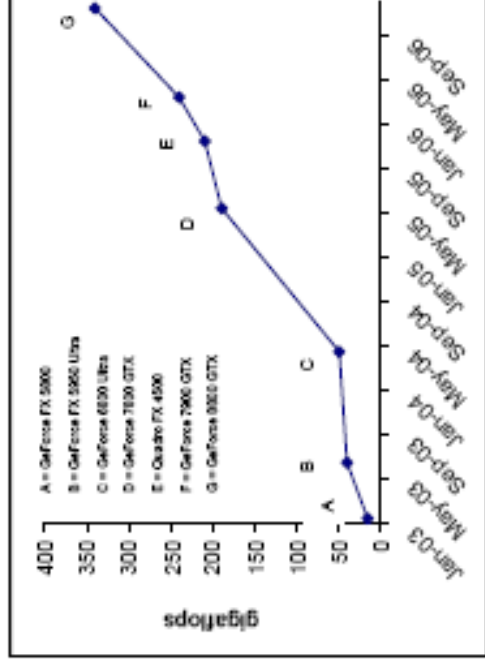


Figure 1-2. Floating-Point Operations/Second on the GPU

NVIDIA Tesla: GPU Compute Tech Brief, Version 1.0.0

# General Purpose GPU Software Options

Most software aimed at graphics. Interest in using them (and CELL processors, XBox, PS3, game consoles) for general purpose computing.

- Microsoft Research windows/DirectX [2007]
- BrookGPU stanford.edu
- GPU specific assemblers
- nVidia CUDA [EuroGP 2008]
- nVidia Cg [GECCO 2007]
- PeakStream
- Sh no longer active. Replaced by RapidMind [EuroGP 2008]
- OpenCL

# Artificial Intelligence on GPU

- Genetic Programming
  - Bioinformatics (breast cancer, gene scanning h/w)
  - Image processing/Graphics
  - Data Mining
  - Financial
- Genetic Algorithms
- Estimation of Distribution Algorithms
- Particle Swarms
- Ant Systems
- Cellular Automata
- Robotic simulations (eg PhysX)
- Commercial = GAMES!

# Missing, Future GPGPU

- **Untimely death of tools.**
  - Tools no longer supported
  - Tools superseded or become more commercial
  - Hardware rapid turn over
- **AMD/ATI**
- **OpenCL**
- **Complete GP on GPU (small gain?)**
- **Applications with fitness evaluation on GPU**

# Experiments

- Interprets 57 billion GP primitives per second (212 sustained peak)
- How?
  - each int contains 32 Booleans
  - randomised test case selection
  - simple CUDA reverse polish interpreter
- 20 mux solved (<< 1500 days)
- 37 mux solved. 137 billion test cases

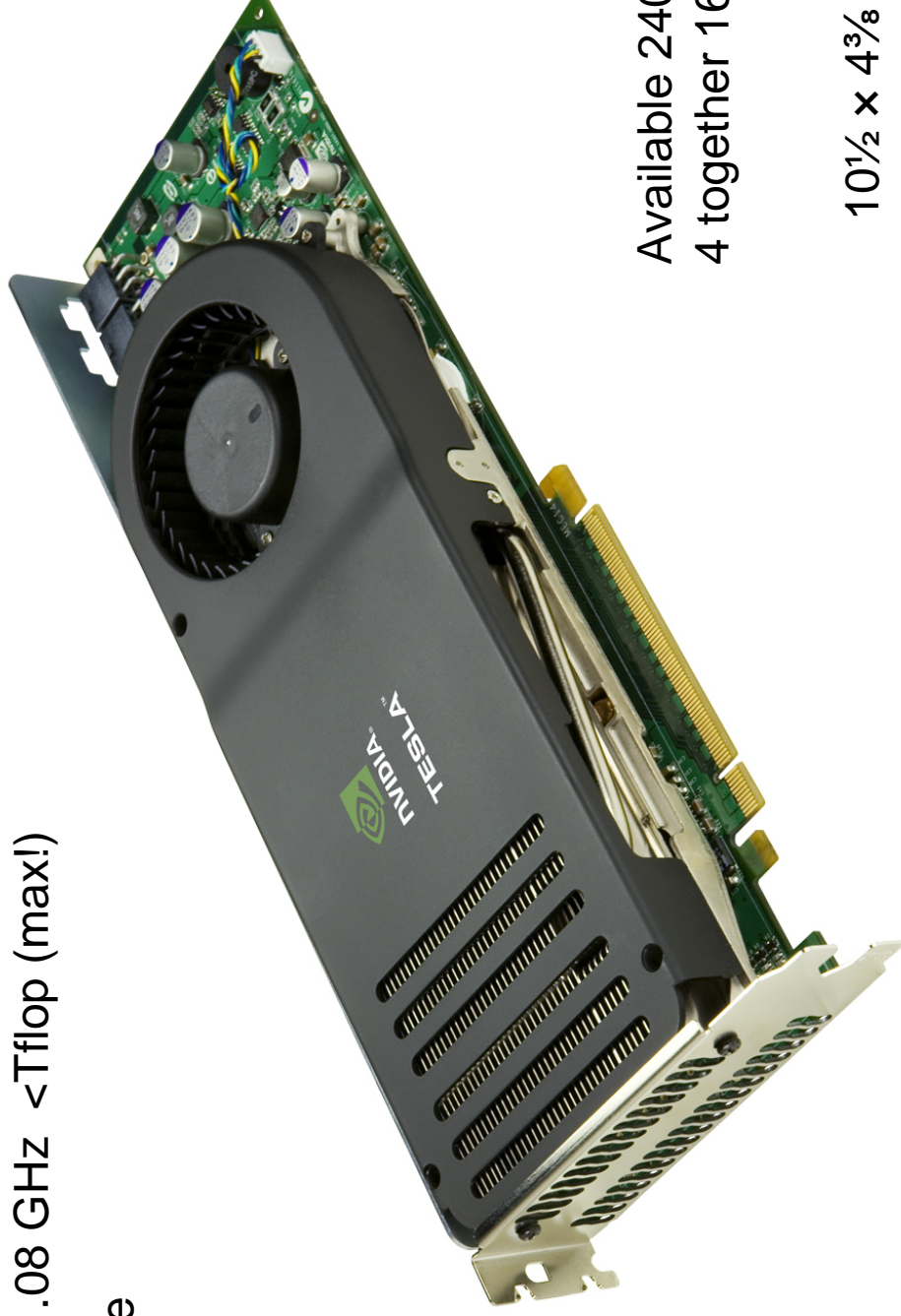


# early nVidia t10p Tesla

192 Stream Processors

Clock 1.08 GHz <Tflop (max!)

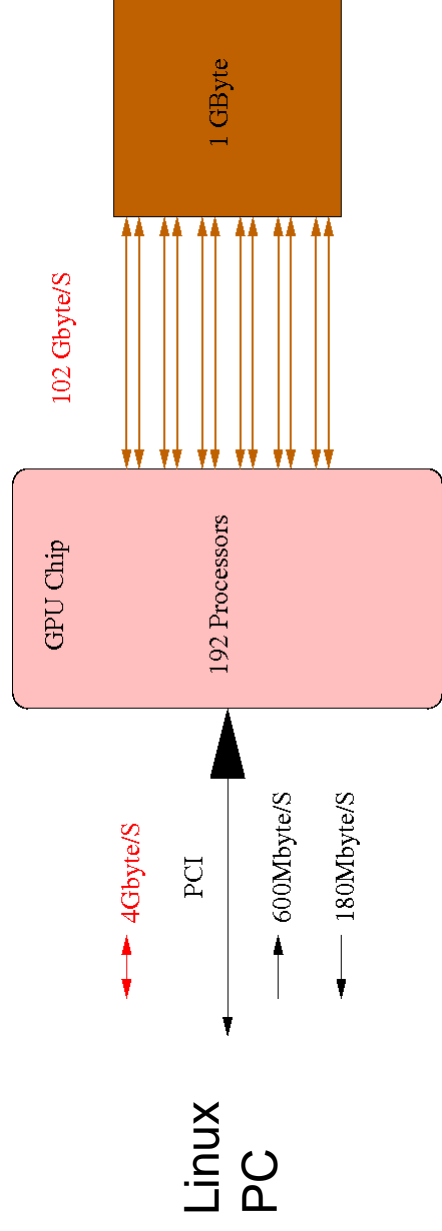
1 GByte



Available 240 1.5GHz  
4 together 16 GBytes

10<sup>1</sup>/<sub>2</sub> x 4<sup>3</sup>/<sub>8</sub> inches

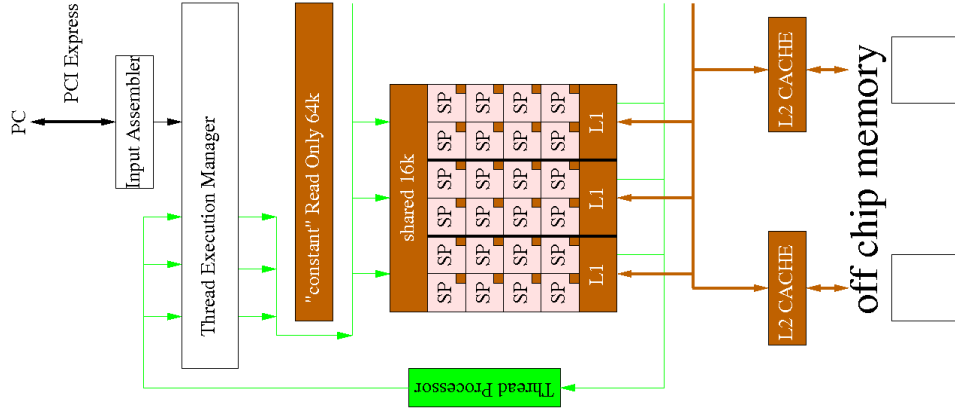
# Tesla chip connections



Memory, GPU chip, etc. All on one card

# CUDA data memory heirarchy

Linux PC



Each stream processor has its own registers

24 SP share 16k. Read/write contention delays threads.

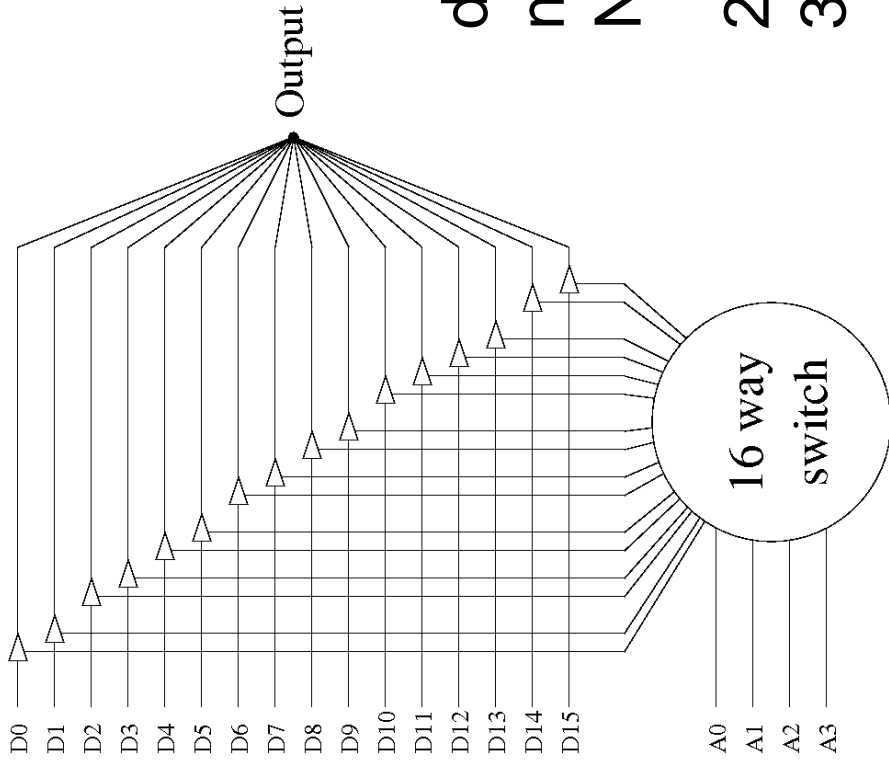
64k can be read by all 8 (10) blocks of SP without contention delays.

Both CUDA “local” and “global” variables are off chip. Latency hundred times more than on chip. Must have thousands of threads to keep SP busy.

Programmer responsible for dividing memory between threads and synchronisation.

Role of caches unclear.

# Boolean Multiplexor



$$d = 2^a$$

$$n = a + d$$

$$\text{Num test cases} = 2^n$$

20-mux 1 million test cases

37-mux 137 10<sup>9</sup> tests

# Submachine Code GP

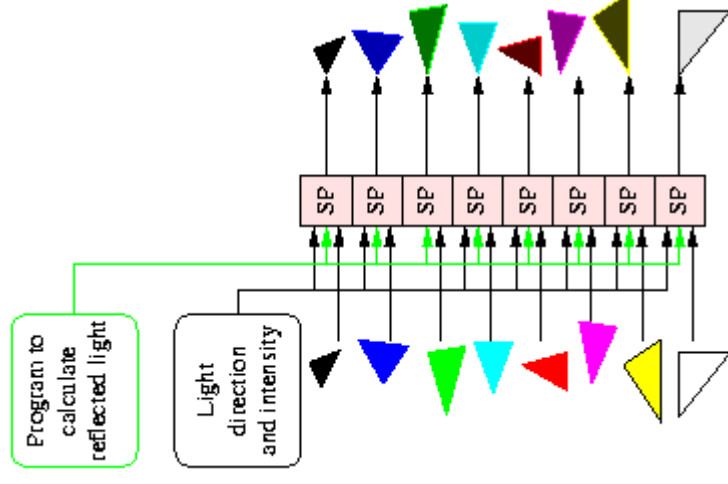
- int contains 32 bits. Treat each as Boolean.
- 32 Boolean (and or, nor, if, not etc) done simultaneously.
- Load 32 test cases
  - D0 = 01010101...
  - D1 = 00110011...
  - D<sub>N</sub> = 00000000... or 11111111...
- Check 32 answers simultaneously
- CPU speed up 24 fold (32) 60 fold (64 bits)

# Randomised Samples

- 20-mux 2048 of 1 million ( $2 \cdot 10^6$ )
- 37-mux 8192 of 137 billion ( $6 \cdot 10^9$ )
- Same tests for everyone in 4-tournament
- New tests for new gen and each tournament
- (Stats test not needed)

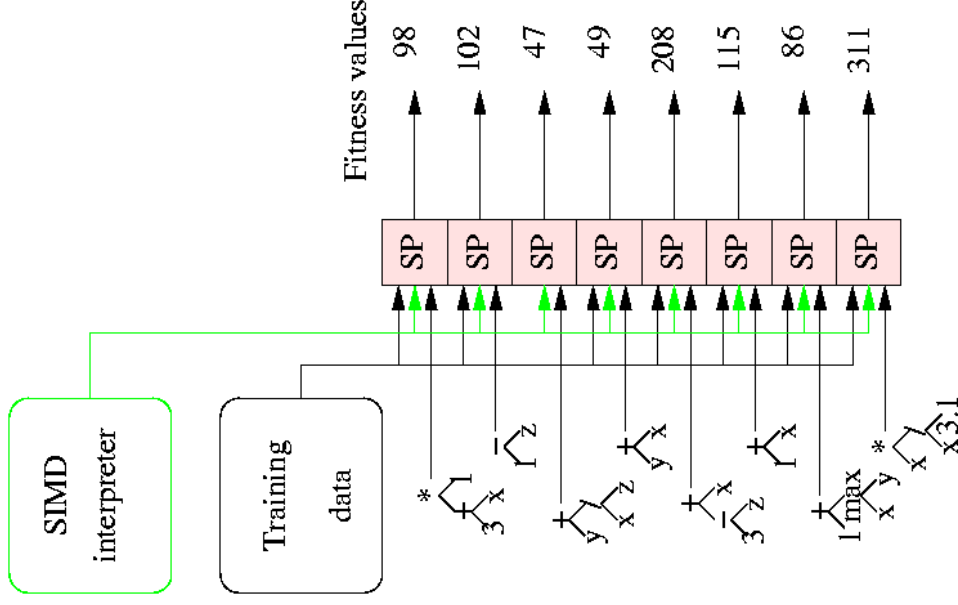
# Single Instruction Multiple Data

- GPU designed for graphics
- Same operation done on many objects
  - Eg appearance of many triangles, different shapes, orientations, distances, surfaces
  - One program, many data → Simple (fast) parallel data streams
- How to run many programs on SIMD computer?



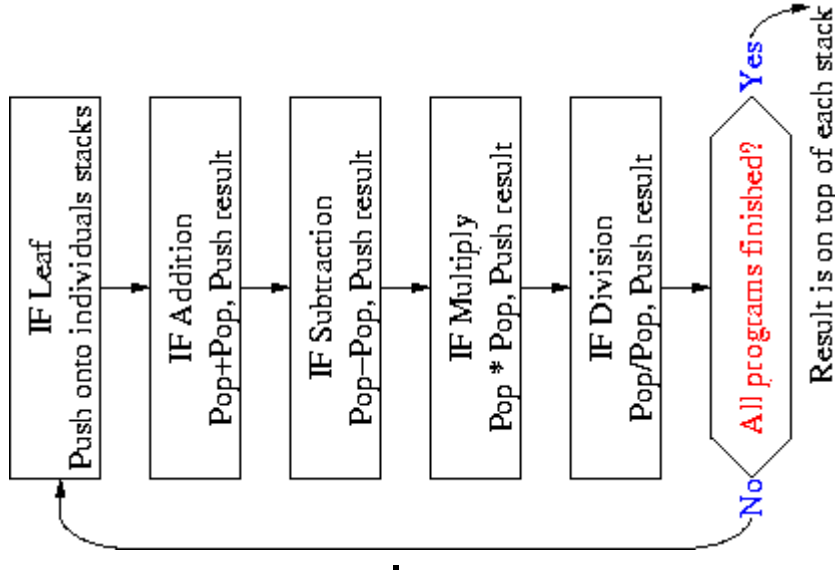
# Interpreting many programs simultaneously

- Can compile gp for GPU on host PC. Then run one program on multiple data (training cases).
- Avoid compilation by interpreting tree
- Run single SIMD interpreter on GPU on many trees.



# GPU Genetic Programming Interpreter

- Programs wait for the interpreter to offer an instruction they need evaluating.
- For example an addition.
  - When the interpreter wants to do an addition, everyone in the whole population who is waiting for addition is evaluated.
  - The operation is ignored by everyone else.
  - They then individually wait for their next instruction.
- The interpreter moves on to its next operation.
- The interpreter runs round its loop until the whole population has been interpreted. (Or a timeout?)



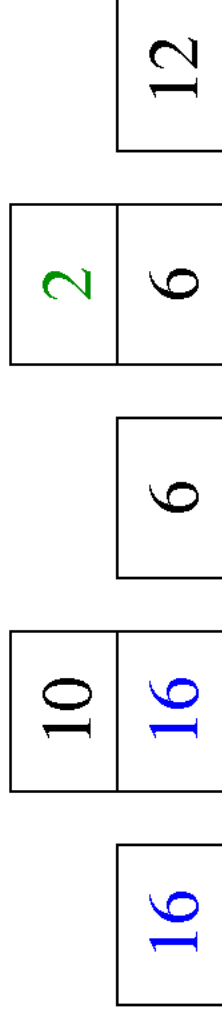
# Representing the Population

- Data is pushed onto stack before operations pop them (i.e. reverse polish.  $x+y \rightarrow \boxed{x} \boxed{y} \boxed{+}$ )
- The tree is stored as linear expression in reverse polish.
- Same structure on host as GPU.
  - Avoid explicit format conversion when population is loaded onto GPU.
- Genetic operations act on reverse polish:
  - random tree generation (eg ramped-half-and-half)
  - subtree crossover
  - 2 types of mutation
- Requires only one byte per leaf or function.
  - So large populations (millions of individuals) are possible.

# Reverse Polish Interpreter

$(A-10) * B$     $A=16$     $B=2$

$A \ 10 \ - \ B \ *$



$(A-10) \times B \equiv A \ 10 \ - \ B \ \times$

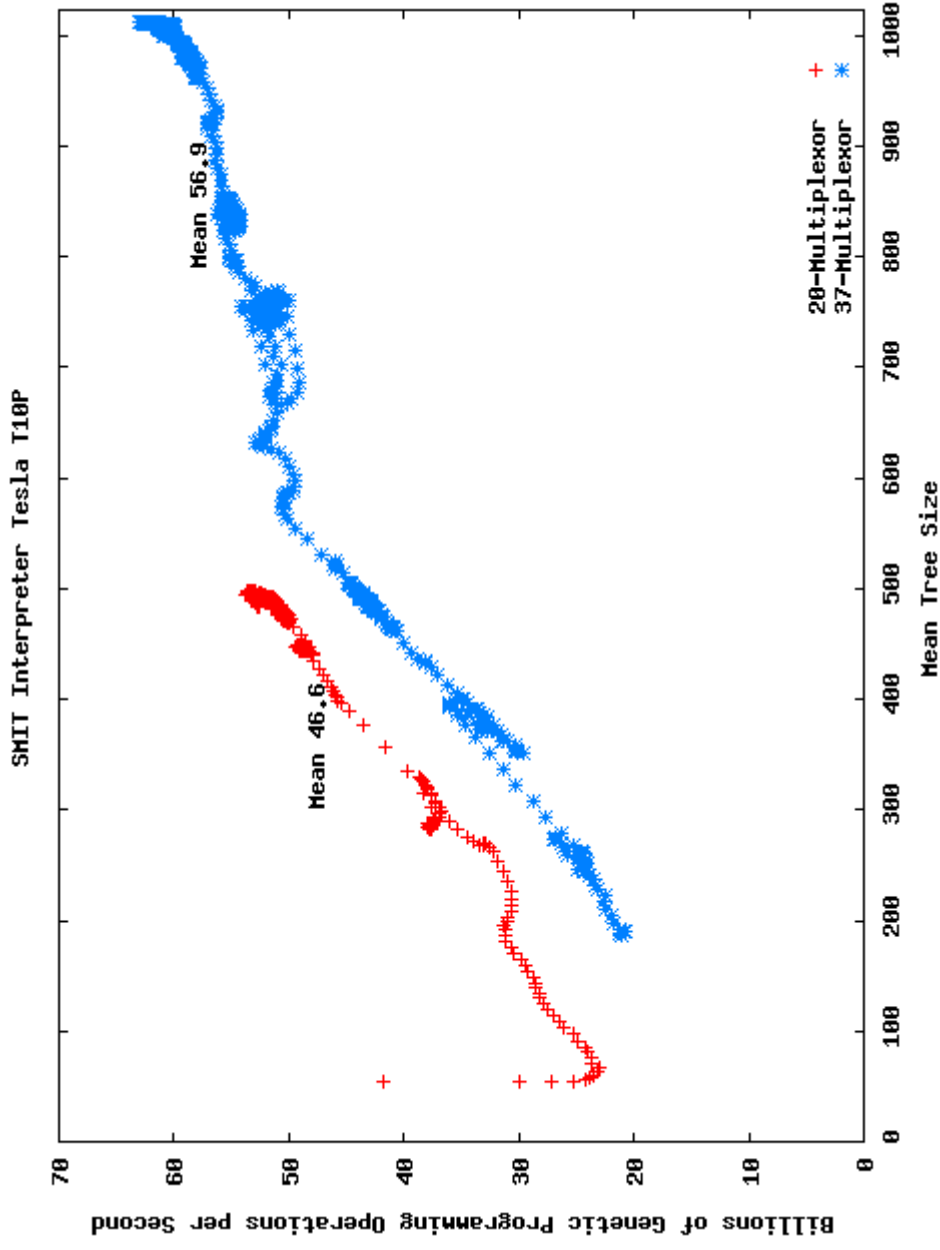
Variable: push onto stack

Function pop arguments, do operation, push result  
 1 stack per program. All stacks in shared memory.

# Validation

- Solutions run on all test cases in GPU.
- Evolved 20-mux and 37-mux expressions converted to C code, compiled and run against all tests

# Performance v RPN size



# Performance

- nVidia early engineering sample (192 SP)
- 57  $10^9$  GP operations/second (peak 212)
- Branchless array based  $\approx$  if based
  - Small improvement 192  $\rightarrow$  240 thread
  - Big (88  $10^9 \rightarrow$  212  $10^9$  GPOps)
- 70 (260) times [CIGPU 2008].
- hardware similar nVidia GeForce 8800 GTX (128 SP)

# LESSONS

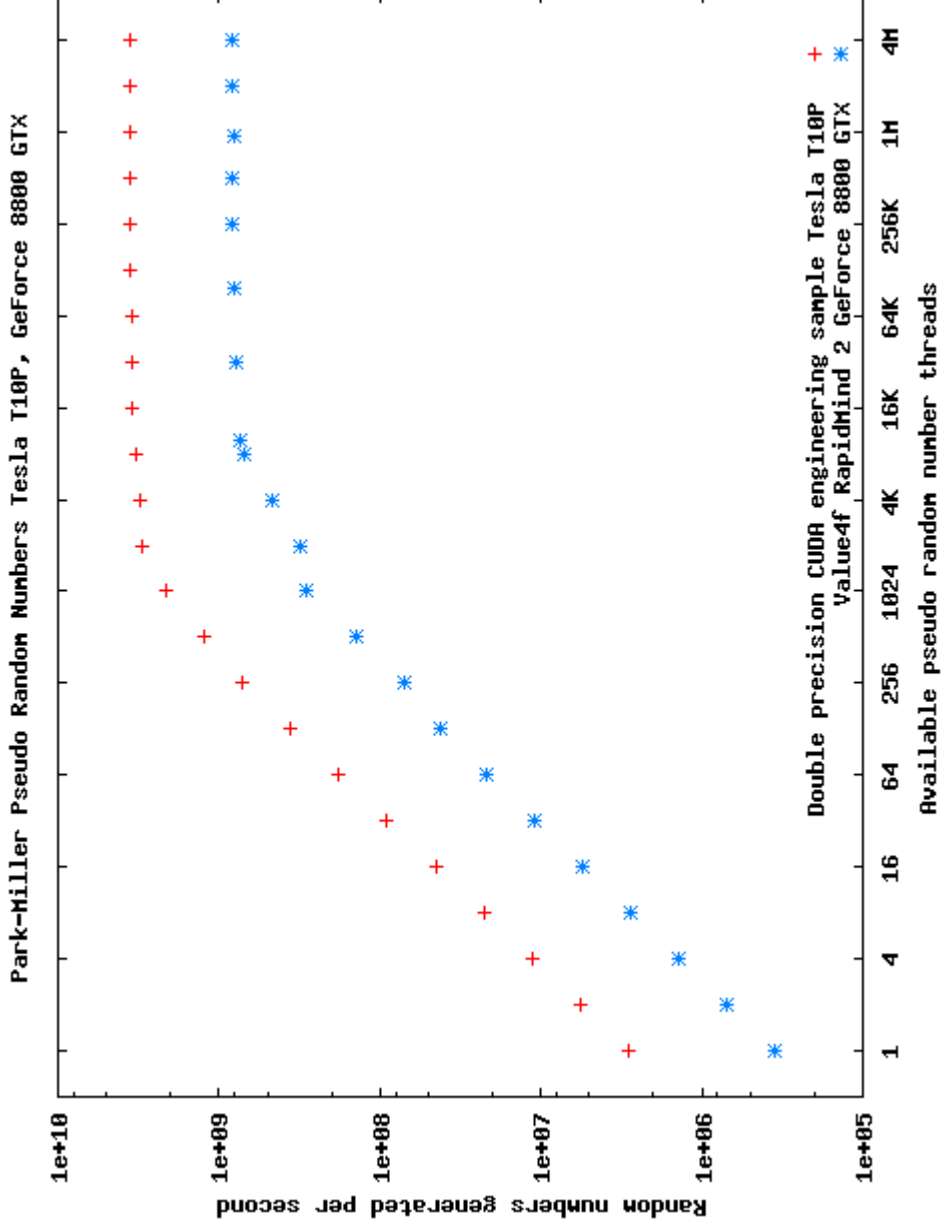
- Computation is cheap. Data is expensive.
- Suggest interpreting GP trees on the GPU is dominated by leafs:
  - since there are lots of them and typically they require data transfers across the GPU.
  - adding more functions will slow interpreter less than might have been expected.
- To get the best of the GPU it needs to be given large chunks of work to do:
  - Aim for at least one second
  - GeForce: more than 10 seconds and Linux dies
    - Solved by not using GPU as main video interface??
  - Less than 1millisec Linux task switching dominates
- Poor debug, performance tools

# Discussion

- Interpreter competitive with compiled GP
  - However using modest number of test cases (8192)
- 32/64-bit suitable for Boolean problems. Also used in regression problems (8 bit resolution), graphics OCR.
- Speed up due to 32bits and CUDA
- Main bottle neck is access to GPU's main memory. But GP pop allows many threads.
- No on-chip local arrays; stack in shared memory
  - Limits number of threads 192 or 240.



# Performance v threads



Threads (log scale)

# Performance v threads 2

- Graph emphasises the importance of using many threads (minimum 4000).
- When a thread stalls because of waiting for off chip memory another thread is automatically scheduled if ready. Thus access to GPU's main memory bottle neck can be overcome.
- At least 20 threads per stream processor
  - 57  $10^9$  GP op/sec with 12 threads per SP
  - 88  $10^9$  GP op/sec with 15 threads per SP

# Conclusions

- GPU offers huge power on your desk
- Interpreted GP can effectively use GPU
- 57 (best 212) billion GP operations per second (0.8 at CIGPU-2008)
- Submachine code GP good on GPU
- Multiplexor solved by sample of test cases
- Tesla first to solve 2 GP benchmarks
  - 20 mux solved (<1 hour v. >4 years)
  - 37 mux solved. 137 billion test cases. <1day

# FOGA 2011

Submissions by 5 July 2010

W. B. Langdon, King's London



- Code via ftp
  - <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gp32cuda.tar.gz>
  - [gpu\\_gp\\_2.tar.gz](http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gp32cuda.tar.gz) [wbl\\_gecco2004lb\\_protein.tar.gz](http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gp32cuda.tar.gz) [gpu\\_park-miller.tar.gz](http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gp32cuda.tar.gz)
- Movies of evolving populations
  - Evolving  $\Pi$ 
    - [http://www.cs.ucl.ac.uk/staff/W.Langdon/pi\\_movie.gif](http://www.cs.ucl.ac.uk/staff/W.Langdon/pi_movie.gif)
    - [http://www.cs.ucl.ac.uk/staff/W.Langdon/pi2\\_movie.html](http://www.cs.ucl.ac.uk/staff/W.Langdon/pi2_movie.html)
  - Evolving Protein Prediction Pop=Million 1000gens
    - [http://www.cs.mun.ca/~blangdon/gpu\\_gp\\_slides/nuclear.gif](http://www.cs.mun.ca/~blangdon/gpu_gp_slides/nuclear.gif)
- [gp-gpu.org](http://gp-gpu.org) [gp-gpu.com](http://gp-gpu.com) [nvidia.com/cuda](http://nvidia.com/cuda)

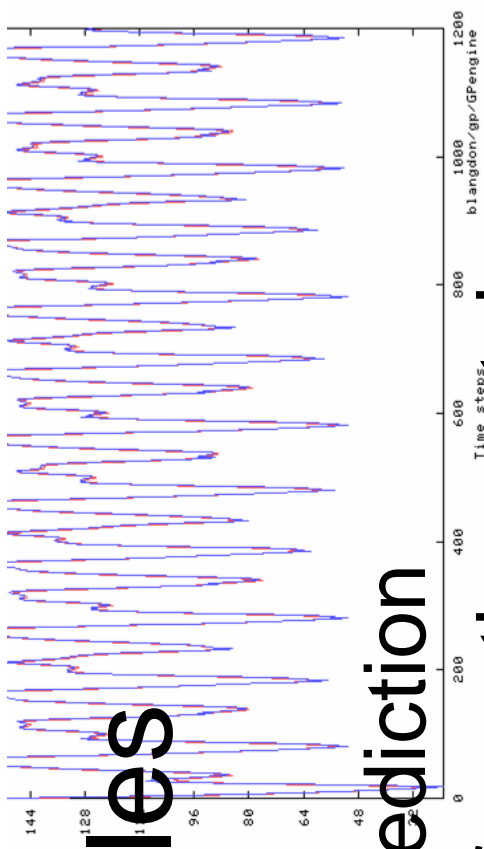


# Speed of GPU interpreter GeForce 8800 GTX.

| Experiment         | Number of Terminals | F | Population | Program size | Stack depth | Test cases       | Speed (million OPs/sec) |
|--------------------|---------------------|---|------------|--------------|-------------|------------------|-------------------------|
| Mackey-Glass       | 8+128               | 4 | 204 800    | 11.0         | 4           | 1200             | 895                     |
| Mackey-Glass       | 8+128               | 4 | 204 800    | 13.0         | 4           | 1200             | 1056                    |
| Protein            | 20+128              | 4 | 1 048 576  | 56.9         | 8           | 200              | 504                     |
| Laser <sub>a</sub> | 3+128               | 4 | 18 225     | 55.4         | 8           | 151 360          | 656                     |
| Laser <sub>b</sub> | 9+128               | 4 | 5 000      | 49.6         | 8           | 376 640          | 190                     |
| Cancer             | 1 013 888+1001      | 4 | 5 242 880  | ≤15.0        | 4           | 128              | 535                     |
| GeneChip           | 47+1001             | 6 | 16 384     | ≤ 63.0       | 8           | 1/3M, sample 200 | 314                     |

CUDA 2.8 billion      Compiled on 16 mac 4.2 billion (100 10<sup>6</sup> data points)  
[2009] 3.8 billion

# Examples



- Approximating  $\Pi$
- Chaotic Time Series Prediction
- Mega population. Bioinformatics protein classification
  - Is protein nuclear based on num of 20 amino acids
- Predicting Breast Cancer fatalities
  - HG-U133A/B probes  $\rightarrow$  10year outcome
- Predicting problems with DNA GeneChips
  - HG-U133A correlation between probes in probesets  $\rightarrow$  MM, A/G ratio and AxC