Linear Genetic Programming GPGPU on the Xbox 360

Garnett Wilson and Wolfgang Banzhaf
Memorial University of Newfoundland,
St. John's, NL, Canada

Why do GPGPU?

- GPUs exceed Moore's Law, which states that general computing power doubles every 18-24 months.
- In contrast, graphics hardware doubles in speed every 6 months, whereas Intel PC CPUs do not meet expectations of Moore's Law.*

* according to survey of nVidia and ATI graphics cards compared to Intel CPUs from 2002 to late 2005, and separate survey up to 2006 based on nVidia GPUs.

- Today's high-end GPUs also exceed the floating point performance of the host CPU.

Why do GPGPU on Consoles?

- Current generation video game consoles have considerable GPU and CPU power, which can be harnessed for research.
- At launch, they are basically graphics supercomputers with cutting edge hardware.
- E.g. Xbox 360, launched on Nov. 22, 2005, was the first PC (or console) to feature:
  - CPU multi-processing (CMP) with more than 2 cores (using 3 cores)
  - GPU-unified shader architecture (no distinct vertex and pixel shader engines).

A Few Firsts

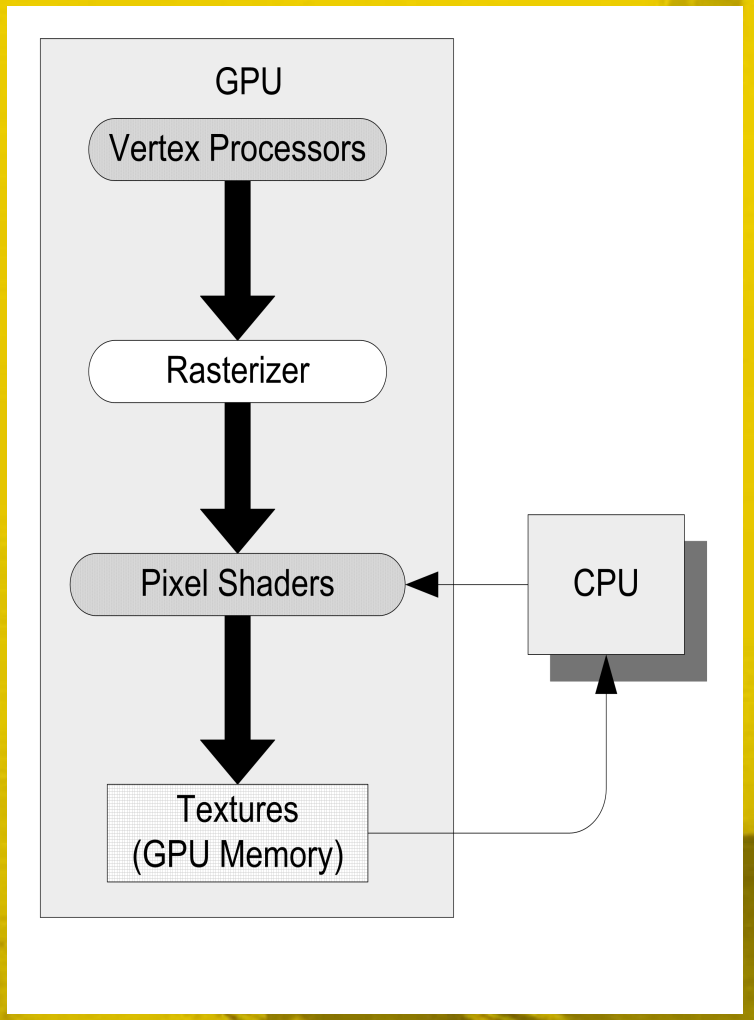
- First implementation of a research-based GP system on a commercial video game platform.
- First time linear GP (LGP) has been implemented in a GPGPU application.
- First instance of XBox 360 being used for any GPGPU purpose.

Xbox 360: Under the Hood

- Custom built IBM PowerPC-based CPU with three 3.2GHz core processors sharing a 1Mb L2 cache.
- CPU core also has an associated complement of SIMD vector processing units.
- CPU cache, cores, and vector units are customized for graphics-intensive computation, and the GPU is able to read directly from the CPU L2 cache.
- Xbox GPU by ATI houses 48 parallel shaders with unified architecture and 10 MB of embedded DRAM (EDRAM).
- 512 MB of DRAM in the system.

GPGPU Summary

- GPGPU applications tend to use pixel shaders (rather than vertex shaders):
  - typically more pixel shaders
  - pixel shader output fed directly to memory
- In terms of traditional data structures and execution:
  - GPU textures are analogous to arrays.
  - the shader program is like a Kernel program.
  - rendering effectively executes the program.
  - CPU runs the main program, and sends data in texture form to the GPU when parallel processing is required.
  - GPU renders to a texture in its memory (rather than to the screen).
  - the output texture data is consumed by the main (CPU-side) program.



The XNA Framework

- In 2006, Microsoft launched XNA's Not Acronymed (recursive acronym "XNA") Game Studio Express 1.0
- Integrated with C# in Visual Studio variants.
- Game Studio 2.0 and 3 CTP have now been released.
- XNA allowed, for the first time, access to the GPU on a video game console.

Tools for Homebrew Development on the Xbox 360

- The following are required for GPGPU on the Xbox 360:
  - C# Studio Express (Game Studio Express 1.0 and Refresh) or Visual Studio 2005 product (Game Studio 2.0)
  - XNA Game Studio (XNA Framework)
  - nVidia's FX Composer (not absolutely required)
  - Xbox 360 with hard drive and XNA Game Launcher installed.
  - Membership in Creator's Club and internet access to Xbox Live.
  - Windows PC with XP SP2 or Vista variant installed.
  - To maximize texture representations, a graphics card capable of supporting at least Pixel Shader v. 3.0.
  - LAN connection between PC and Xbox 360.

Design Considerations for Xbox 360

- Microsoft is currently the only console vendor allowing access to GPUs.
- Accelerator is not compatible with the XNA framework, so shaders are implemented in HLSL.
- XNA programs run by repeatedly updating the Update and Draw methods (like a video game).
- XNA's "content pipeline" does not permit dynamic loading or switching of shader programs to the GPU (so treating shader programs as individuals to be subject to operators is not possible).
- Hard drive I/O was not possible as of XNA 1.0 Refresh, so data must be output to screen. Means of input for the Xbox 360 include controller and USB keyboard.

Design Considerations (continued…)

- With XNA, GPU cannot implement scatter, thus:
  - Results must be rendered to a texture on an internal target buffer (rather than the screen).
  - Content is read back to the calling program from the internal target.
  - Array data stored on textures must be referenced using texture coordinates with an appropriate mapping.
- Xbox 360 GPU and Pixel Shader 3.0 have additional specifications (available by querying Xbox 360 with XNA GraphicsDevice class):
  - Shader program can consist of 2048 instructions.
  - Flow control of depth 4 (maximum of 4 instructions can be called within one another).
  - Supports 16 simultaneous textures.
  - Maximum texture height and width of 8192.

GP Individual Representation (Textures)

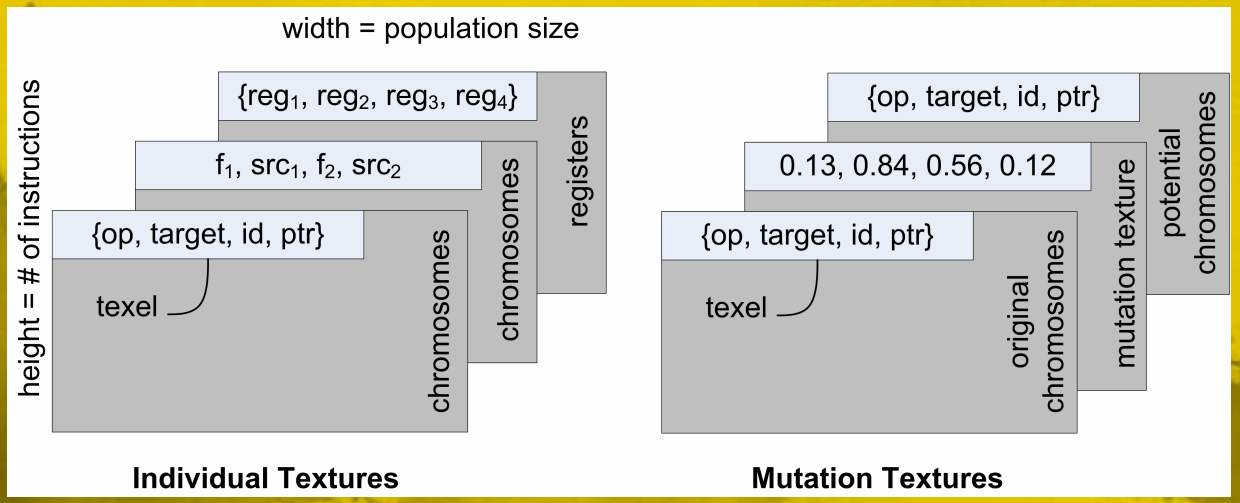
- Eight chromosomes in an instruction, each set of 4 placed on different texture.
- First Texture holds {*op*, *target*, *id*, *ptr*}.
- Second Texture holds {f_1, src_1, f_2, src_2}.
- Each instruction perform an operation on the contents of two sources (fitness case or register content), placing result in target:

target = src1 op src2

  - op = [0, 3] corresponding to ADD, SUB, MUL, or DIV
  - src_1 and src_2 can specify either fitness cases or registers, and thus take values in [0, MAX(*classification features or regression inputs*, *registers*)]
  - *id*, *id* = [0, *population size*] labels the individual
  - *ptr*, *ptr* = [0, *instructions*] serves as a pointer to the current instruction
  - Boolean flags f_1, f_2, indicate whether to load from fitness cases or registers for src_1 and src_2, respectively.

GP Individual Textures

- XNA *HalfVector4* surface format was used, each chromosome (channel) was a 16 bit float.
- The two textures represent a whole population, with each individual being a column of texels, and each texel in the column being an instruction.
- Width of the textures (in texels) is the number of individuals .
- Height is the number of instructions in an individual.
- Current state of an individual's four registers (following an instruction) are kept in a **third** texture's texels (at the same coordinates) as 4 floats.



GPU-side Shader Program: Mutation

- For every channel (all 4) of each pixel of the instructions (2 textures)
  - a "mask" texture, with channels containing values [0.0 … 1.0], is applied.
  - If the mutation threshold is > mask texture amount for a particular channel
    - an appropriate replacement value for the channel is given by randomly generated replacement textures (2 textures corresponding to instruction textures)

GPU-side Shader Program: Fitness

- This was a long shader program, which evaluated each instruction in an individual (of length 16 instructions) .
- Experiments showed that fitness evaluation, at least in the form used in these experiments, was best left to CPU-side processing.
- Further fitness shader optimization may improve GPU-side speeds.
- There are considerations for running the fitness shader on the XBox 360 vs. nVidia GeForce 8800:
  - XBox microcode compiler issues with loops inside other loops relying on instructions of the outer loop.
  - Prevents looping over instructions within loop over fitness cases, for instance.

CPU-side GP Program

```
GPGame {
        GPGame()  //constructor
                provide seedings for each trial
        Initialize()
                prompt for user input using on-screen keyboard
                declare and populate HalfVector4[] data arrays for all textures
Update(GameTime)
                check for exit key pressed on control pad
                parse user keyboard input until completed
Draw(GameTime)  // evaluates fitness case over population
          // each pass evaluates an instruction over all individuals
          for passes in fitnessEffect
                        run Fitness.fx  HLSL program (see above)
                        resolve render target to texture, get array data from texture
          // do for each fitness case
          adjust all individual's fitnesses; fitCase++
          if at the end of a generation
                fitness-proportionate generational selection
                run Mutate.fx HLSL program (on two texture sets)
          if at the end of a trial
                        trial++;  round = 0;
                        add best fitness to growing List for output
        if all trials are not yet done
                display fitness, timer, and population texture output
}
```

XNA-based Linear GP GPGPU In Action

GPU REGRESSION PROBLEM Trial 19 Generation: 1 Fitness Case: 37

Operator, Target, Individual, Pointer Texture

Flag 1, Source 1, Flag 2, Source 2 Texture

Registers 1 - 4 Texture

Mutation Operator, Target, Individual, Pointer Texture

Mutation Flag 1, Source 1, Flag 2, Source 2 Texture

Replacement Operator, Target, Individual, Pointer Texture

Replacement Flag 1, Source 1, Flag 2, Source 2 Texture

RESULTS

9 1.92661 8.599059	12 1.842168 6.849171
4 2.522821 7.4575305	8 1.962663 6.38631
7 1.962704 6.927291	7 1.962704 6.538644
7 1.962357 6.302331	8 2.370641 6.940962
8 1.962663 6.2251875	7 2.124706 8.548281
9 1.962368 6.196869	7 1.962539 13.713966
7 1.962704 7.1040375	8 1.962663 11.9133
8 1.962663 6.4107225	9 1.859883 10.5159285
8 1.962856 6.892137	7 1.962368 11.411379
8 1.962663 6.362874	

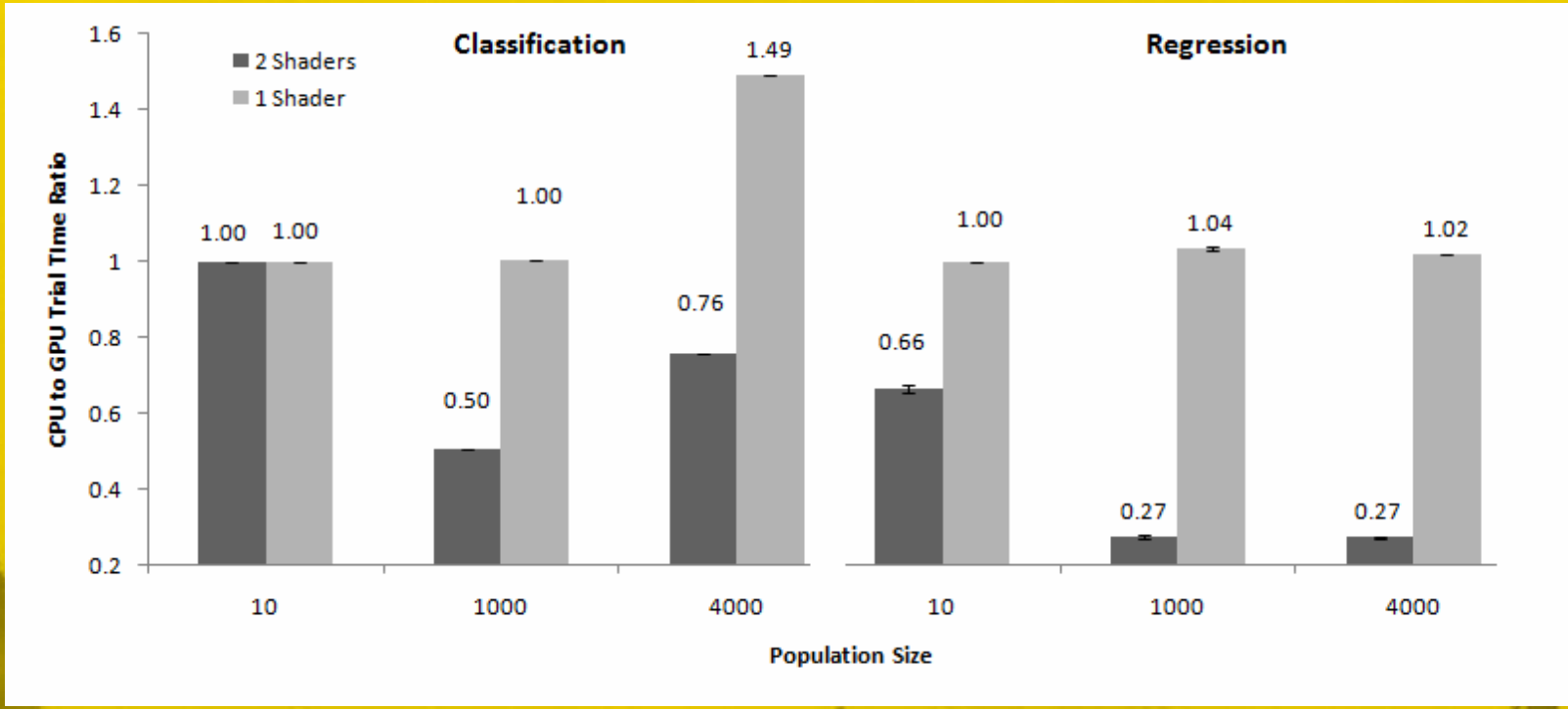
Experimental Set-up

- CPU-only version of the implementation was also created, implementing all shader functionality with appropriate C# code.
- Two benchmark problems:
  - Ecoli problem from the UCI machine learning repository was chosen for classification, using 75% of the training set that retained the class distribution of the entire data set.
  - The sextic polynomial $x^6 - 2x^4 + x^2$ introduced by Koza was implemented for regression, using float inputs in the range [0, 1] for 50 fitness cases.
- Windows PC specifications:
  - OS: Windows Vista Business PC
  - IDE: Visual C# 2005 Express with XNA Game Studio Express 1.0 (Refresh)
  - CPU: AMD Athlon 64 Processor 3500+ (2.21 GHz),
  - Memory: 1023 MB of RAM
  - Graphics Card: ASUS EN8800GTX video card with nVidia GeForce 8800 GTX GPU on board (using128 parallel stream processors with unified shader architecture)

Parameterization

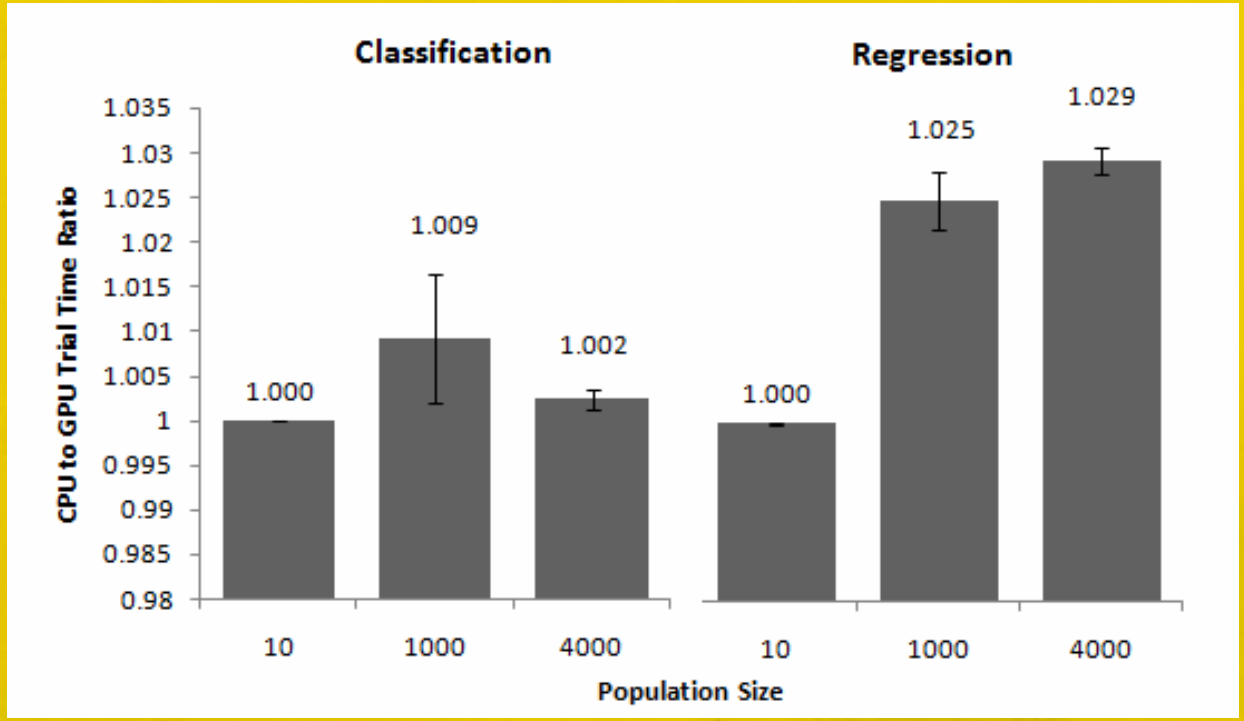
Function Set	ADD, SUB, MUL, DIV (on floats)
Fitness	fitness-proportionate roulette wheel
Population	10, 1000, or 4000 individuals
Mutation	threshold = 0.1
Tournament	generational, 50 rounds
Fitness Cases	Classification: 251 training cases, 7 float features, 8 integer categories Regression: 50 cases, x = [0, 1]
Fitness Metric	Classification: correct classification, based on Reg[0] mapping to category Regression: 50 hits, where a hit is Absolute(Reg[0] – y) <= 0.01

Intra-Platform CPU vs. GPU Performance: PC



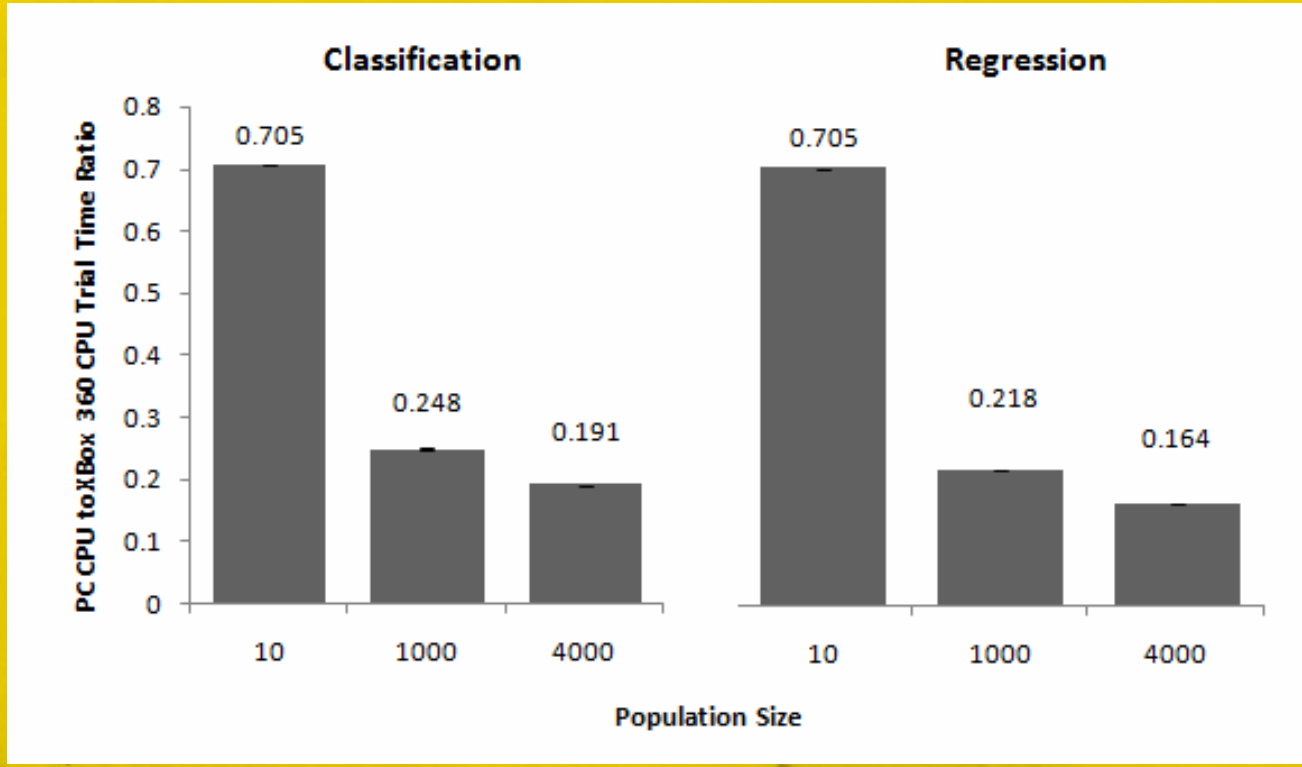
CPU to GPU (both 1 and 2 shaders) mean trial time ratios on PC with standard error, based on 10 trials of 50 generations for classification (left) and regression (right) benchmarks. Ratios of greater than 1 show GPU use is faster, less than 1 that CPU is faster.

Intra-Platform CPU vs. GPU Performance: Xbox 360



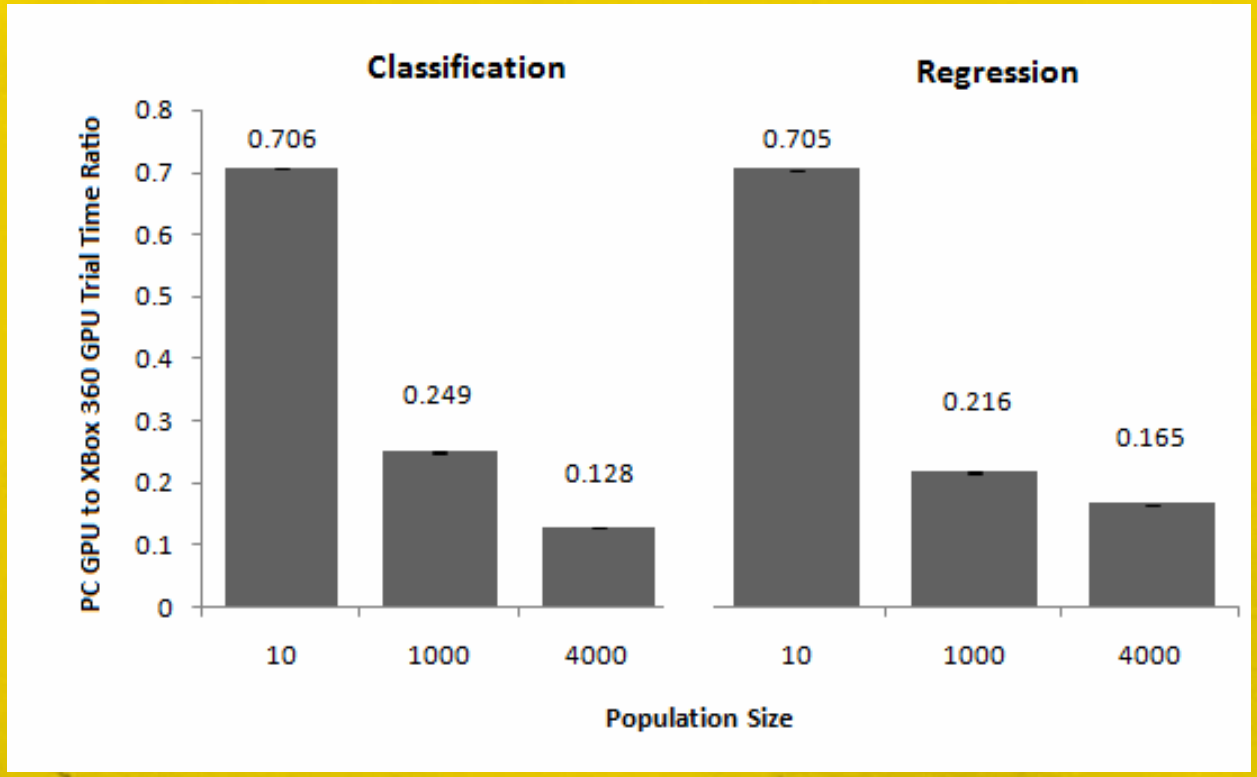
CPU to GPU mean trial time ratios on Xbox 360 with standard error, based on 10 trials of 50 generations. Ratios of greater than 1 show GPU use is faster, less than 1 that CPU is faster.

Inter-Platform CPU Comparison



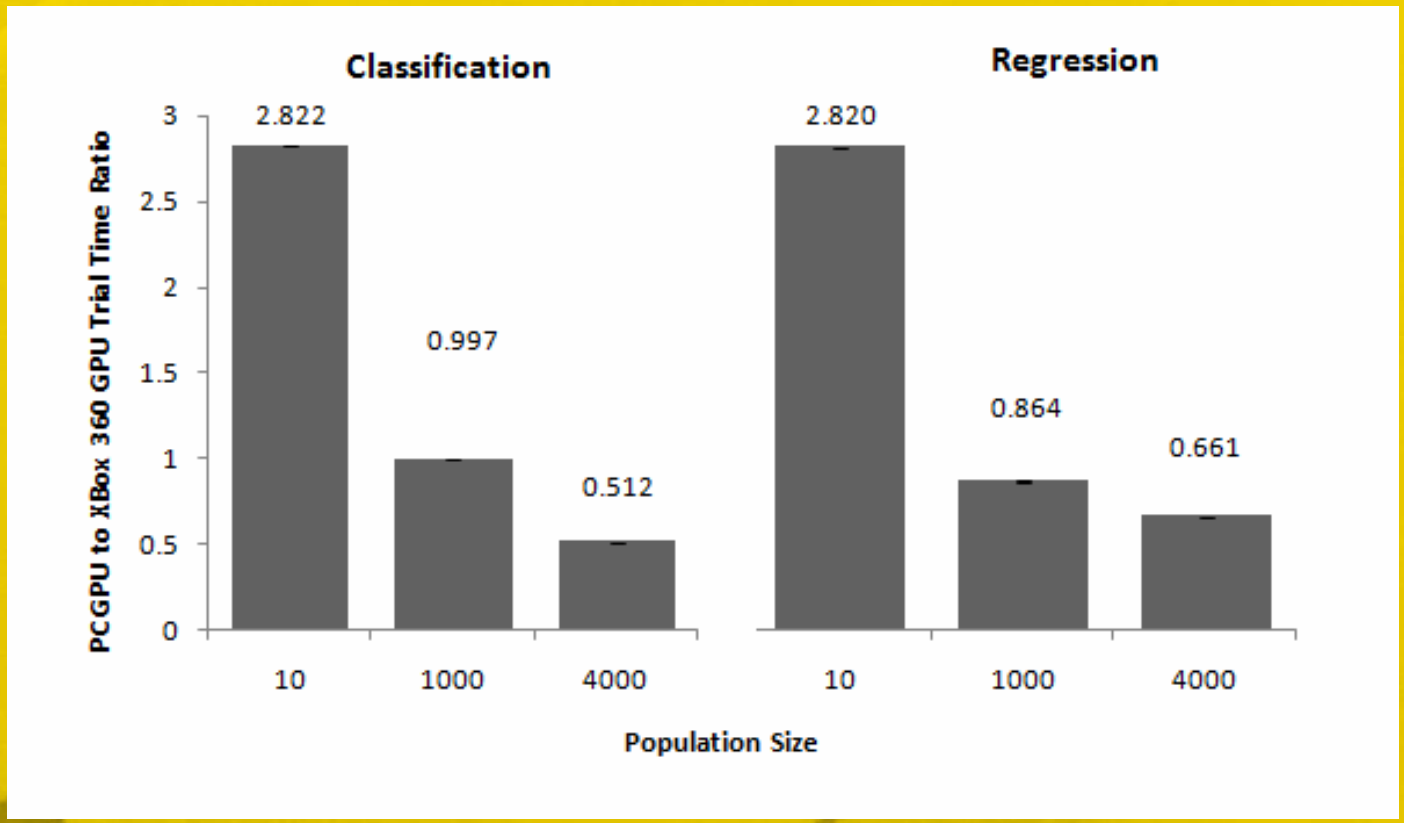
PC CPU to Xbox 360 CPU mean trial time ratios with standard error, based on 10 trials of 50 generations. Ratios of greater than 1 show the Xbox 360 CPU use is faster, less than 1 that the PC CPU is faster.

Inter-Platform GPU Comparison



PC GPU to Xbox 360 GPU mean trial time ratios with standard error, based on 10 trials of 50 generations. Ratios of greater than 1 show the Xbox 360 GPU use is faster, less than 1 that the PC GPU is faster.

Projected Inter-Platform GPU Comparison



PC GPU to Xbox 360 GPU mean trial time ratios with standard error, normalized to current generation of GPUs, based on 10 trials of 50 generations. Ratios of greater than 1 show the Xbox 360 GPU use is faster, less than 1 that the PC GPU is faster.

Conclusions

- The main goal of this work was to show how to implement a GP system on a commercial video game console (Xbox 360) using GPGPU.
- First time (to our knowledge) that GPGPU, or genetic programming, has been implemented on a commercial video game console for research purposes.
- First instance of a Linear GP implementation using GPGPU.
- Xbox 360 to offers tightly coupled CPU and GPU graphics performance.
- Use of XNA ought to allow programmers to take advantage of the next Microsoft console launch, putting research implemented in XNA at the front of the hardware market at that point.

Acknowledgements

We would like to acknowledge the support of:

- PRECARN postdoctoral fellowship
- Memorial University of Newfoundland