Parallel Genetic Algorithm on the CUDA Architecture

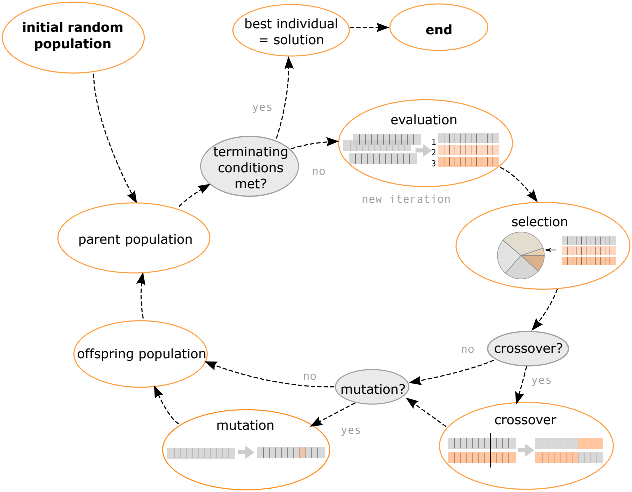
Petr Pospíchal, Jiří Jaroš and Josef Schwarz
{ipospichal,jarosjir,schwarz}@fit.vutbr.cz

Brno University of Technology
Faculty of Information Technology, Department of Computer Systems

Overview

1. Motivation
   - Genetic algorithms
   - Problem
   - GPU vs. CPU
2. Proposed solution
   - GPGPU possibilities and disadvantages
   - CUDA Hardware model
   - Mapping CUDA hardware model to software model
3. Results
   - Speedup
   - Quality
   - Limitations
4. Conclusion

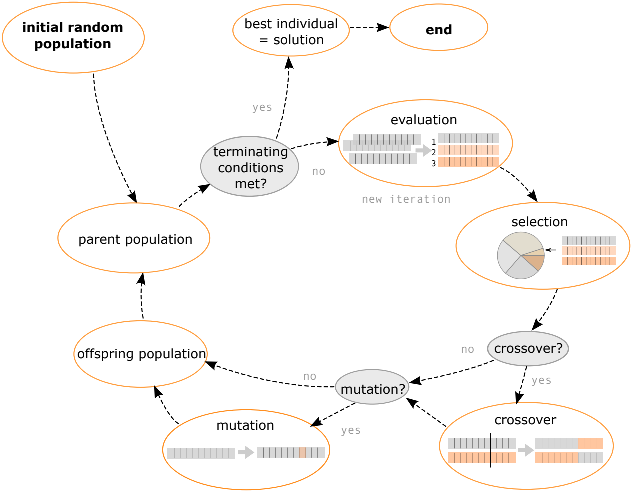
Genetic algorithms 1/2



- stochastic optimization technique
- employs population of candidate solutions
- black-box ⇒ minimal problem knowledge
- robust, wide area of applications
- **inheritely parallel, but slow**

⇒ This work is focused on accleration

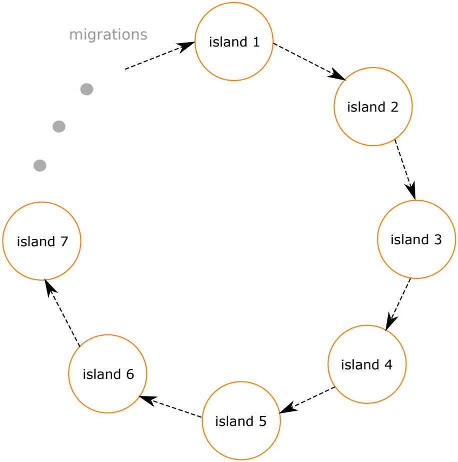
Genetic algorithms 1/2



- stochastic optimization technique
- employs population of candidate solutions
- black-box ⇒ minimal problem knowledge
- robust, wide area of applications
- **inheritely parallel, but slow**

⇒ **This work is focused on accleration**

Genetic algorithms 2/2: Island Model with migrations



- several independent populations
- better convergence tovards different suboptima
- new operator: migration
- migration occasionally transfers good genetic material between islands

Motivation

Problem

Genetic algorithms are effective in solving many practical problems **BUT their exection usually take a long time**.

Possible solutions

- Hardware accelerators (i.e. FPGA) – difficult
- Multicore parallelization – small speedup
- Grid computing – expensive
- simple, cheap, available for everyone?

$\Rightarrow$ **graphic cards - ideal?**

Motivation

Problem

Genetic algorithms are effective in solving many practical problems **BUT their exection usually take a long time**.

Possible solutions

- Hardware accelerators (i.e. FPGA) – difficult
- Multicore parallelization – small speedup
- Grid computing – expensive
- simple, cheap, available for everyone?

$\Rightarrow$ **graphic cards - ideal?**

Motivation

Problem

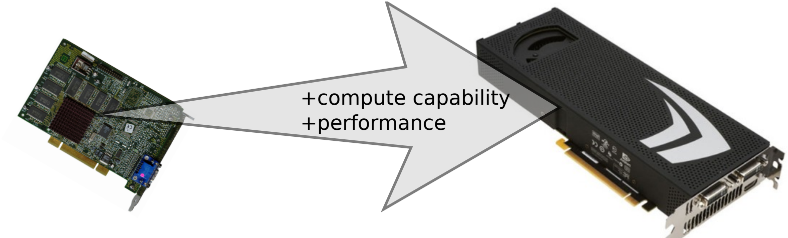
Genetic algorithms are effective in solving many practical problems **BUT their exection usually take a long time**.

Possible solutions

- Hardware accelerators (i.e. FPGA) – difficult
- Multicore parallelization – small speedup
- Grid computing – expensive
- simple, cheap, available for everyone?

$\Rightarrow$ **graphic cards - ideal?**

GPU compute capability history

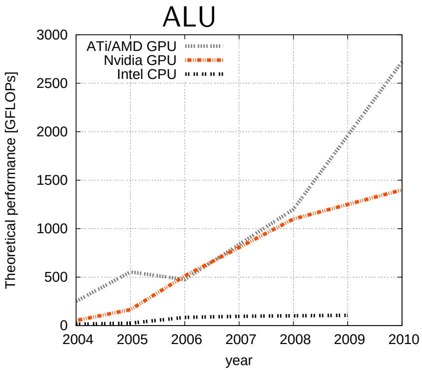


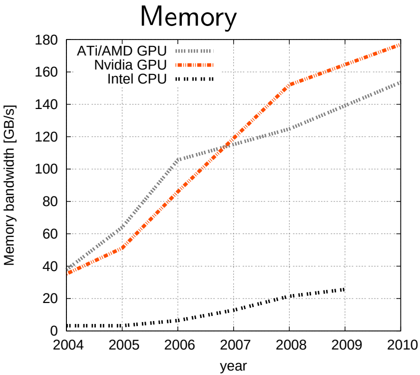
1999 Voodoo 3 — GeForce GTX 295 **2009**

	pixel+vertex units	Cg			**unified shader units**	CUDA	OpenCL	DirecX 11 compute	**C++**
year	**2001**	**2002**	**2003**	**2004**	**2006**	**2007**	**2008**	**2009**	**2010**
shader version	1.0	1.3	2.0	3.0	**4.0**	4.1		5.0	
max. executed instructions	<10		~100	~10000	**no limit**				

⇒ **compute capability is sufficient** (there are some minor limitations)

Graphics Processing Units (GPUs) vs. CPUs





⇒ **raw performance is huge** (memory can be limiting)

Why (not) GPUs

advantages

- huge raw FP power GTX 280 - 240 cores = 1TFLOP
- good power/price and power/Watt ratio
- hardware thread scheduler (little overhead during switching)
- fast on-chip memory (user managed L1 cache)
- external adapter, scalability (multi-GPU solution)

disadvantages

- SIMD hardware – bad branching
- limited data types, double is slow
- low performance per thread, requires massively-parallel tasks
- PCI-Express bus bottleneck

$\Rightarrow$ **Challenge**

Why (not) GPUs

advantages

- huge raw FP power GTX 280 - 240 cores = 1TFLOP
- good power/price and power/Watt ratio
- hardware thread scheduler (little overhead during switching)
- fast on-chip memory (user managed L1 cache)
- external adapter, scalability (multi-GPU solution)

disadvantages

- SIMD hardware – bad branching
- limited data types, double is slow
- low performance per thread, requires massively-parallel tasks
- PCI-Express bus bottleneck

⇒ Challenge

Why (not) GPUs

advantages

- huge raw FP power GTX 280 - 240 cores = 1TFLOP
- good power/price and power/Watt ratio
- hardware thread scheduler (little overhead during switching)
- fast on-chip memory (user managed L1 cache)
- external adapter, scalability (multi-GPU solution)

disadvantages

- SIMD hardware – bad branching
- limited data types, double is slow
- low performance per thread, requires massively-parallel tasks
- PCI-Express bus bottleneck

⇒ **Challenge**

Next coming up...

Compute Unified Device Architecture (CUDA)

- nVidia framework for general purpose computation on GPUs (GPGPU)
- works on GeForce 8 (first unified shader generation) and better under both Windows and Linux
- good control over hardware, allows direct utilization of on-chip shared memory
- consists of hardware and software model
- best GPGPU results so far

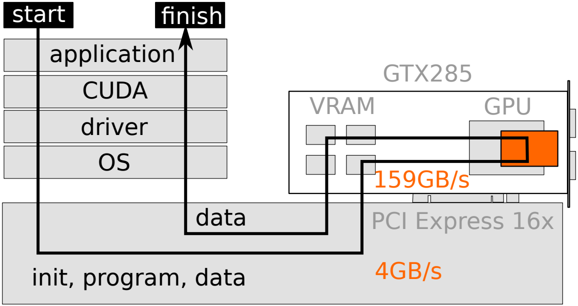
⇒ Selected for our implementation

Compute Unified Device Architecture (CUDA)

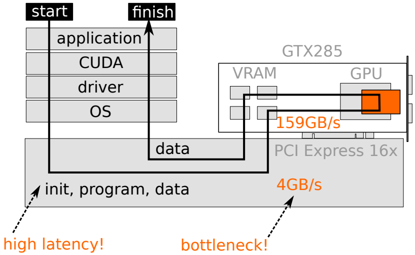
- nVidia framework for general purpose computation on GPUs (GPGPU)
- works on GeForce 8 (first unified shader generation) and better under both Windows and Linux
- good control over hardware, allows direct utilization of on-chip shared memory
- consists of hardware and software model
- best GPGPU results so far

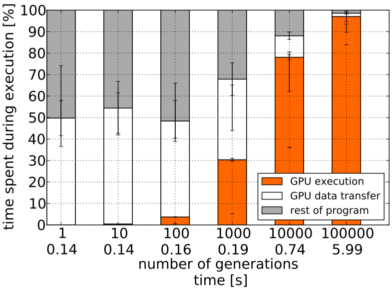
$\Rightarrow$ **Selected for our implementation**

GPGPU execution datapath



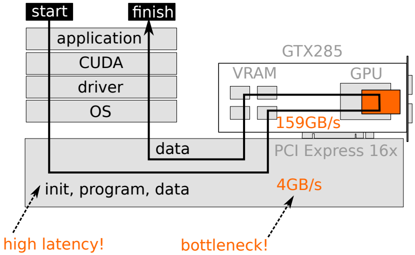
GPGPU execution: drawback

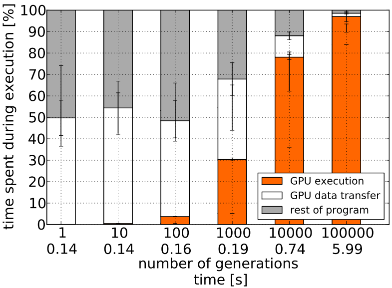




⇒ whole GA should be executed on graphic card

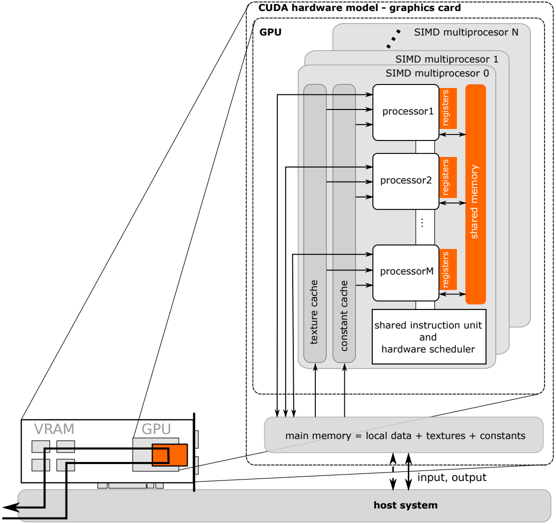
GPGPU execution: drawback





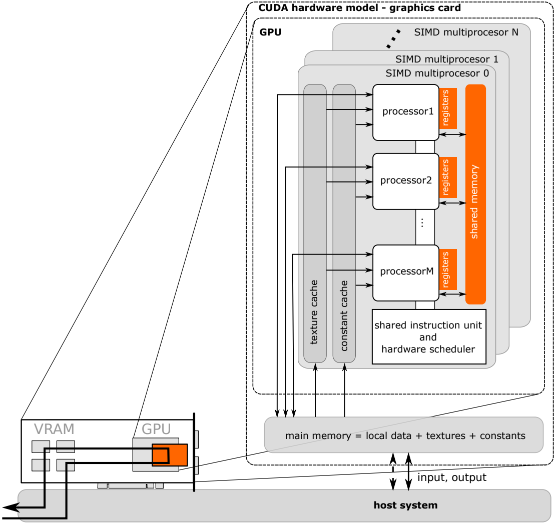
⇒ **whole GA should be executed on graphic card**

CUDA Hardware model



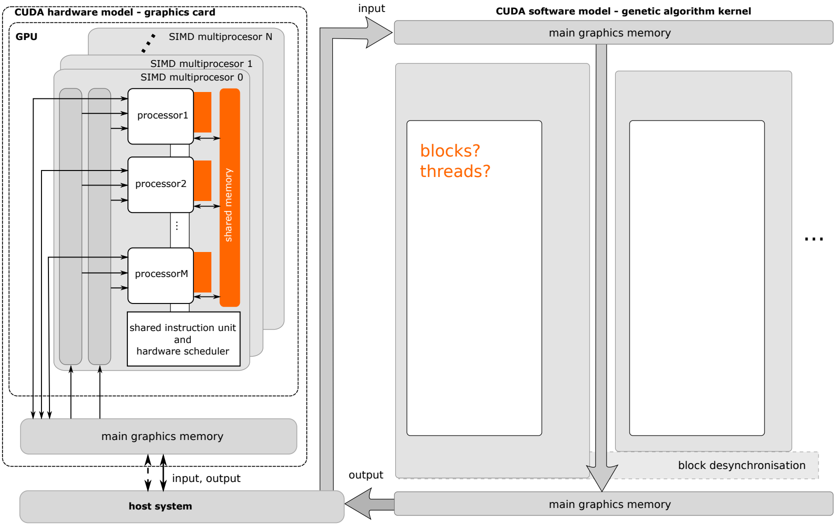
- GPU is divided into SIMD multiprocessors
- multiprocessors contain fast shared memory
- main memory is very slow
- (only) processors within multiprocessors can be synchronized easily
- GTX280 GPU - 30 x 8 processors, 1GB main, 16KB shared memory
- ⇒ effective mapping of the GA is required

CUDA Hardware model

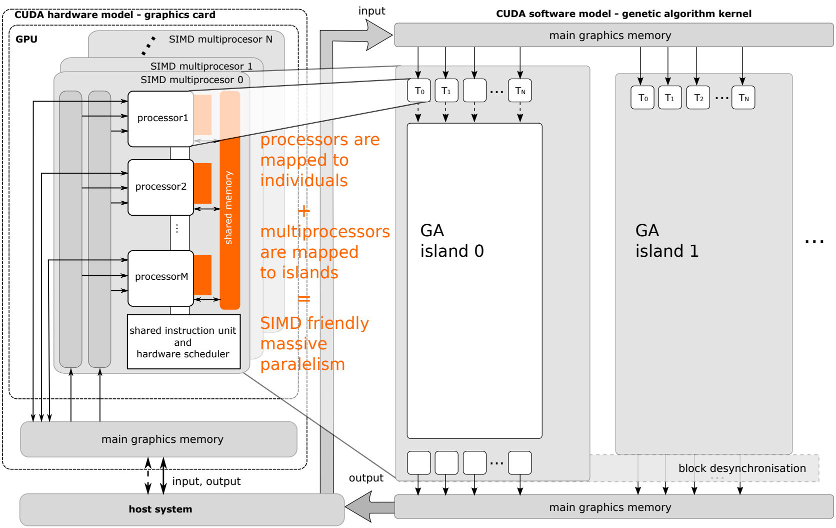


- GPU is divided into SIMD multiprocessors
- multiprocessors contain fast shared memory
- main memory is very slow
- (only) processors within multiprocessors can be synchronized easily
- GTX280 GPU - 30 x 8 processors, 1GB main, 16KB shared memory
- $\Rightarrow$ **effective mapping of the GA is required**

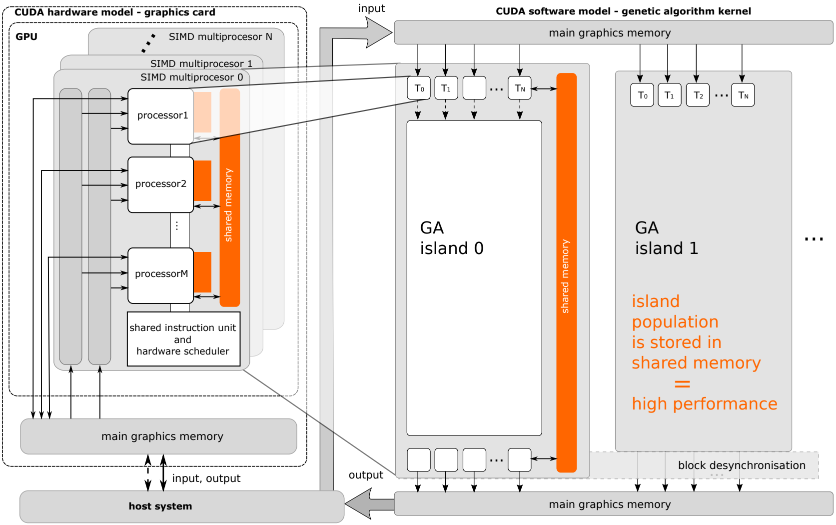
Mapping CUDA hardware model to software model: step 1



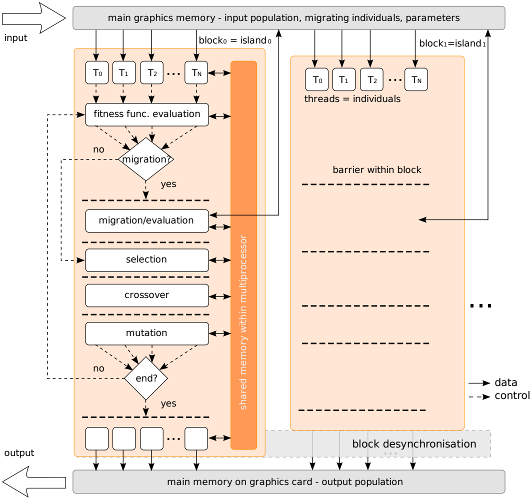
Mapping CUDA hardware model to software model: step 2



Mapping CUDA hardware model to software model: step 3



Complete GPU GA kernel



- island GA with asynchronous unidirectinal ring migration
- whole GA is executed on the GPU in parallel
- fast shared memory is used to maintain population
- SIMD-friendly execution
- usage of slow main memory is minimized
- compiler constants/macro parameters

Next coming up...

Testing enviroment

CPU	hardware	Core i7 920
	software	single threaded Galib, no elitism, custom Tournament
GPU	hardware	88000 GTX (128 cores)
		GTX 260 (216 cores)
		GTX 285 (240 cores)
	software	presented custom GA

GA parameter	**value**
number of generations	1000
individuals per island	varying from 2 to 256
number of islands	varying from 1 to 1024
selection	Tournament (N=2)
mutation	Gauss
fitness	Rosenbrock, Michalewicz and Griewank
genome length	varying from 2 to 10

$\Rightarrow$ **speedup and quality**

Testing enviroment

CPU	hardware	Core i7 920
	software	single threaded Galib, no elitism, custom Tournament
GPU	hardware	88000 GTX (128 cores)
		GTX 260 (216 cores)
		GTX 285 (240 cores)
	software	presented custom GA

GA parameter	**value**
number of generations	1000
individuals per island	varying from 2 to 256
number of islands	varying from 1 to 1024
selection	Tournament (N=2)
mutation	Gauss
fitness	Rosenbrock, Michalewicz and Griewank
genome length	varying from 2 to 10

⇒ **speedup and quality**

Testing enviroment

CPU	hardware	Core i7 920
	software	single threaded Galib, no elitism, custom Tournament
GPU	hardware	88000 GTX (128 cores)
		GTX 260 (216 cores)
		GTX 285 (240 cores)
	software	presented custom GA

GA parameter	**value**
number of generations	1000
individuals per island	varying from 2 to 256
number of islands	varying from 1 to 1024
selection	Tournament (N=2)
mutation	Gauss
fitness	Rosenbrock, Michalewicz and Griewank
genome length	varying from 2 to 10

⇒ **speedup and quality**

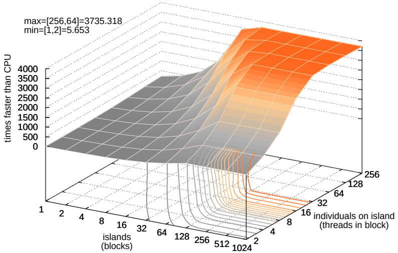
Speedup

- **Performance of the GPU highly depends on population size** – from 2 to 256 individuals per island ⇒ the performance unit is population-size independent, IIGG=$\prod$(Island population size, number of Islands, Genotype length, number of Generations) per second
- Migration cost for 100% of individuals every generation about 30%
- Mean value from 5 runs (max. about 5% difference), chromosome length = 2

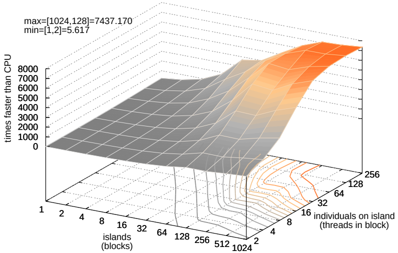
arch.	**fitness function**	(min – max) **IIGG**$\cdot 10^6$ **per second**		
	Rosenbrock		2.6 – 2.8	
CPU	Michalewicz		1.8 – 2.5	
	Griewank		2.5 – 2.8	
		8800GTX	**GTX260**	**GTX285**
GPU	Rosenbrock	14.2 – 8877	12.0 – 13094	14.3 – 18669
	Rosenbrock-FastMath	18.5 – 11914	15.5 – 17318	18.5 – 24288
	Michalewicz	6.9 – 5893	5.8 – 8850	7.0 – 12937
	Michalewicz-FastMath	11.7 – 9894	9.8 – 13692	11.6 – 19400
	Griewank	9.6 – 7108	8.0 – 10515	9.9 – 14496
	Griewank-FastMath	15.9 – 10507	13.3 – 15360	15.8 – 20920

Speedup – cont.

Speedup of kernel execution against CPU (Griewank):



(a) 8800 GTX

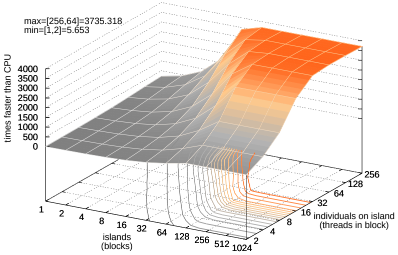


(b) GTX 285

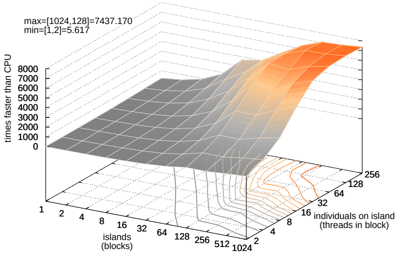
⇒ **99% GPU utilization for peak performance, up to 8000 times faster execution compared to single threaded Galib**

Speedup – cont.

Speedup of kernel execution against CPU (Griewank):



(c) 8800 GTX



(d) GTX 285

⇒ **99% GPU utilization for peak performance, up to 8000 times faster execution compared to single threaded Galib**

Quality comparsion

- same parameters and 32 individuals per island
- CPU and GPU1 is single island
- GPU2: fully utilized GPU, 1024-islands, 10% individuals are migrated every 10 generations
- Quality comparsion, mean value from 100 runs

genes	mean best fitness from whole population								
	Rosenbrock			Michalewicz			Griewank		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
2	0.086	3.468	$7.57{\cdot}10^{-7}$	-1.022	-1.768	-1.801	0.0005	0.0020	$3.99{\cdot}10^{-12}$
3	1.897	4.996	0.447	-1.220	-2.336	-2.760	0.0051	0.0048	$1.06{\cdot}10^{-8}$
4	8.900	4.997	0.494	-1.459	-2.748	-3.696	0.0156	0.0188	$1.22{\cdot}10^{-7}$
5	22.112	17.332	2.042	-1.684	-3.184	-4.628	0.0246	0.0414	0.0001
6	48.450	56.045	4.313	-1.817	-3.654	-5.440	0.0408	0.0570	0.0005
7	83.455	42.509	6.903	-2.035	-3.646	-6.163	0.0479	0.0620	0.0012
8	128.710	155.233	9.257	-2.120	-3.805	-6.659	0.0650	0.1360	0.0027
9	167.329	131.737	12.045	-2.176	-4.830	-7.136	0.0749	0.1444	0.0042
10	233.364	184.370	15.379	-2.391	-5.009	-7.649	0.0805	0.1758	0.0058

GPU1 is 20% better $\Rightarrow$ GPU is able to optimise simple numerical functions and do it better in the same time

Quality comparsion

- same parameters and 32 individuals per island
- CPU and GPU1 is single island
- GPU2: fully utilized GPU, 1024-islands, 10% individuals are migrated every 10 generations
- Quality comparsion, mean value from 100 runs

genes	mean best fitness from whole population								
	Rosenbrock			Michalewicz			Griewank		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
2	0.086	3.468	$7.57 \cdot 10^{-7}$	-1.022	-1.768	-1.801	0.0005	0.0020	$3.99 \cdot 10^{-12}$
3	1.897	4.996	0.447	-1.220	-2.336	-2.760	0.0051	0.0048	$1.06 \cdot 10^{-8}$
4	8.900	4.997	0.494	-1.459	-2.748	-3.696	0.0156	0.0188	$1.22 \cdot 10^{-7}$
5	22.112	17.332	2.042	-1.684	-3.184	-4.628	0.0246	0.0414	0.0001
6	48.450	56.045	4.313	-1.817	-3.654	-5.440	0.0408	0.0570	0.0005
7	83.455	42.509	6.903	-2.035	-3.646	-6.163	0.0479	0.0620	0.0012
8	128.710	155.233	9.257	-2.120	-3.805	-6.659	0.0650	0.1360	0.0027
9	167.329	131.737	12.045	-2.176	-4.830	-7.136	0.0749	0.1444	0.0042
10	233.364	184.370	15.379	-2.391	-5.009	-7.649	0.0805	0.1758	0.0058

GPU1 is 20% better ⇒ GPU is able to optimise simple numerical functions and do it better in the same time

Limitations

- ?? too many simulated islands ⇒ GPU can simulate multiple independent runs at the same time
- fitness function must be executed on the GPU
- limited shared memory per island (currently 16KB)

New generation has bigger shared memory, L2 cache and supports C++ ⇒ future is bright for GA on GPUs

Limitations

- ?? too many simulated islands $\Rightarrow$ GPU can simulate multiple independent runs at the same time
- fitness function must be executed on the GPU
- limited shared memory per island (currently 16KB)

New generation has bigger shared memory, L2 cache and supports C++ $\Rightarrow$ future is bright for GA on GPUs

Next coming up...

Conclusions

- we have used GPU to accelerate genetic algorithms and published results on EvoStar 2010
- **GPU has great potential for acceleration of simple numerical function optimization!**
- Galib is not very well optimized, hand-tuning on CUDA leads to massive speedup
- up to 400 times better power/watt ratio, consumer level hardware

Future work

- Grammatical evolution
- More complex GA

Conclusions

- we have used GPU to accelerate genetic algorithms and published results on EvoStar 2010
- **GPU has great potential for acceleration of simple numerical function optimization!**
- Galib is not very well optimized, hand-tuning on CUDA leads to massive speedup
- up to 400 times better power/watt ratio, consumer level hardware

Future work

- Grammatical evolution
- More complex GA

Thank you

Thank you for your attention

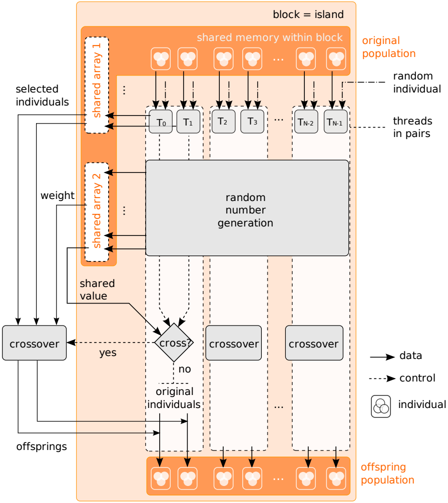
Feel free to ask questions!

Thank you

Thank you for your attention

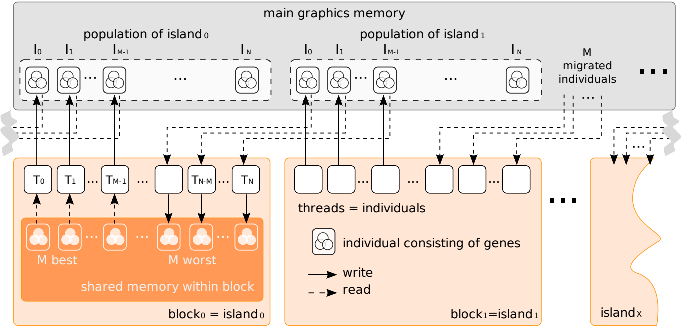
Feel free to ask questions!

Appendix A: Genetic operators: Tournament selection



- uses fast, parallel PRNG HybridTaus from GPU Gems 3 book seris (mutation uses BoxMuller PRNG)
- threads use shared memory to work in pairs

Appendix B: Genetic operators: Migration



- unidirectinal ring
- asynchronous, but works well
- uses main memory for chromosomes and Bitonic-Merge sort algorithm implementation (GPU Gems book)