OpenCL implementation of PSO: a comparison between multi-core CPU and GPU performances

Stefano Cagnoni¹, Alessandro Bacchini¹,², Luca Mussi¹
¹Dept. of Information Engineering, University of Parma, Italy
²Henesis srl, Parma, Italy
Overview

• Motivation

• PSO parallelization

• GPU / Multi-core CPU implementation

• Experimental results

• Conclusions
Motivation

• GPUs
  • massively parallel execution of tasks on hundreds of cores

• Multi-core CPUs
  • coarser grain
  • fewer, more powerful and complex cores
Motivation

• GPU-based code is overwhelmingly faster than single-threaded sequential code

• Most papers describing GPU-based parallel algorithms report only this comparison; the power of multi-core CPUs is underexploited

• What about the performance of multi-core CPU implementations?
Goal

• Comparing performances of GPU-based and multi-core CPU-based parallelization of a bio-inspired metaheuristic

• OpenCL chosen as development environment, since it can produce code for both GPUs and multi-core CPUs

• Based on our previous implementations, we chose PSO parallelization as a test-bed
Why is PSO so attractive?

Not the best metaheuristic at all...

However...

• Easy to implement
• Fast-converging
• Effective for many practical problems

and (last but not least)

• Very well parallelizable
Why is PSO so attractive?

Parallelization opportunities offered by many fitness functions

• Functions based on cumulative sums of independent computations
• Functions implying operations on large matrices,
• etc...
Previous GPU-PSO implementations

- **Three-kernel synchronous** *(Information Sciences, 2011)*
  - Any topology allowed
  - Any problem size
  - Large overhead (three memory swaps)

- **Single-kernel asynchronous** *(GECCO 2011)*
  - Ring topology, radius = 1
  - Limited number of particles
  - Fastest possible (no swaps)
Previous work on GPU-PSO
Single-kernel vs. Multi-kernel

Synchronous multi-kernel PSO
Previous work on GPU-PSO
Single-kernel vs. Multi-kernel

Asynchronous single-kernel PSO
(ring topology, radius=1)
Previous work on GPU-PSO

Single-kernel vs. Multi-kernel

• Single-kernel (all computations in local memory)
  + No (limited) need for synchronization
    No data exchange between GPU and CPU
  − Limited local resources
    Small maximum number of particles in a swarm

• Multi-kernel (need for 3 data swaps)
  + Virtually no resource-related limitation
    Any swarm size possible (up to several hundreds)
  − Large memory overhead due to the need for synchronization after each kernel is run
New implementation

- Single kernel
- Synchronization at the end of each cycle
  - One can schedule as many threads as necessary
- Suitable for both CPUs & GPUs
- Virtually no limits to the number of particles
- Smaller memory overhead wrt the multi-kernel version
GPU

- Massively parallel architecture
  - Hundreds or thousands of simple cores
- Simple instruction set
  - Synchronization primitives
- Deep memory hierarchy
  - Private, local, global, constant memory
- Each one has a different role
Multi-core CPU

- Parallel architecture
  - 2 to 12 cores
- Complex instruction set
  - Vectorized instructions (SSE, AVX)
- Shallow memory hierarchy
  - Global and local memory share the same chips
Vectorization instructions

• A single instruction operates on multiple data

• OpenCL natively supports vector data types
  • The OpenCL compiler has auto-vectorization capabilities, but manually optimized vectorization still offers better results

• GPU/CPU comparison:
  • Intel i7, with 8 cores and AVX SIMD instructions, can process 64 floats in parallel
  • Nvidia GeForce GTX560 Ti can process 384 floats in parallel
    • 6 times as many as the CPU
Vectorization

• Non-vectorized
  One thread per dimension
  128 particles on a 128-D problem = 16384 threads

• Better for GPUs

• Vectorized
  • 8 dimensions per thread
  • 128 particles on a 128-D problem = 2048 threads

• Better for CPUs
Tests

• A set of 5 commonly (ab)used functions was used as benchmark:
  • Sphere \([-100, +100]^N\]
  • Elliptic \([-100, +100]^N\]
  • Rastrigin \([-5.12, +5.12]^N\]
  • Rosenbrock \([-30, +30]^N\]
  • Griewank \([-600, +600]^N\]

• Our goal was to compare execution speed
• Algorithm equivalence was also checked
Tests

- 2 multi-core CPUs:
  - Intel i7 2630M (high-end laptops)
  - Intel i7 2600K (medium/high-end desktops)

were compared to 3 GPUs:

- nVidia GT540M (medium/high-end laptops)
- nVidia GT560Ti (medium/high-end desktops)
- ATI Radeon HD6950 (medium-end laptops)
Tests

• We tested the scaling properties of our GPU-based and CPU-based implementations
  • With respect to problem size
    • 32, 64, 128 dimensions
  • With respect to swarm size:
    • 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 particles
• Other PSO parameters
  • $C_1 = C_2 = 1.19315$
  • $\omega = 0.72134$
Results: 64D Griewank

![Graph showing the relationship between swarm size and time for different hardware configurations. The x-axis represents swarm size, ranging from 32 to 8192, and the y-axis represents time in milliseconds, ranging from 100 to 10000. Different hardware configurations are represented by various markers and line styles.](image-url)
Results: 32D, 128D Griewank

Time (ms)

Swarm size
Results: general remarks

- Scaling properties are not surprising:
  - Initial ‘flat’ segment, followed by linear increase after maximum degree of parallelism is reached

- Peculiarities:
  - nVidia GT540M is sometimes the fastest for small sizes and problem dimensions, for its slightly higher clock frequency
  - The gap between i7 and i7M narrows as problem complexity and swarm size increase: no explanation related to code or processor; possibly caused by other hardware components.
Results: GPU/CPU comparison

- GPUs are generally faster than multi-core CPUs, however:
  - Not necessarily for small swarm sizes (32-64 particles are enough for most real-world problems)
  - PSO is highly parallelizable, as are highly parallelizable the fitness functions we have used in our tests
  - Tests were generated up to huge swarm sizes, much larger than usually necessary in typical real-world applications
Results: GPU/CPU comparison

• The spread is larger for high-dimensional problems

• For larger dimensions even a cheap GPU as the GT540M has similar performances as a high-end Intel i7 processor

• In any case GPUs were never more than 6 times faster than CPUs
Results: GPU/CPU comparison

- Taking development costs into consideration:
  - Writing parallel code is more expensive, and may take more time than it saves
  - If the cost of parallelization is acceptable AND the algorithm is intrinsically parallel, then GPUs are preferable
  - Results obtained by multi-core CPUs can be close to GPUs’ when GPUs cannot be used (e.g., if the graphics card must also do its traditional job...)
Some publicly-available GPU code developed at the IBIS Lab

  - Three-kernel implementation and some benchmark functions

- **libCUDAOptimize** (http://sourceforge.net/projects/libcudaoptimize/)
  - PSO, DE, Scatter Search plus benchmark functions and utilities (not yet online but coming soon)

- **libCUDANN** (http://sourceforge.net/projects/libcudann/)
  - Multi-layer perceptron training (BP algorithm)

- OpenCL PSO probably also available soon.
Thank you