High-Dimensional Adaptive Particle Swarm Optimization on Heterogeneous Systems

High-Dimensional Adaptive Particle Swarm Optimization on Heterogeneous Systems

M P Wachowiak[1]**, B B Sarlo, and A E Lambe Foster**
Department of Computer Science and Mathematics
Nipissing University, North Bay, ON Canada, P1B 8L7

E-mail: markw@nipissingu.ca

Abstract. Much work has recently been reported in parallel GPU-based particle swarm optimization (PSO). Motivated by the encouraging results of these investigations, while also recognizing the limitations of GPU-based methods for big problems using a large amount of data, this paper explores the efficacy of employing other types of parallel hardware for PSO. Most commodity systems feature a variety of architectures whose high-performance capabilities can be exploited. In this paper, high-dimensional problems and those that employ a large amount of external data are explored within the context of heterogeneous systems. Large problems are decomposed into constituent components, and analyses are undertaken of which components would benefit from multi-core or GPU parallelism. The current study therefore provides another demonstration that "supercomputing on a budget" is possible when subtasks of large problems are run on hardware most suited to these tasks. Experimental results show that large speedups can be achieved on high dimensional, data-intensive problems. Cost functions must first be analysed for parallelization opportunities, and assigned hardware based on the particular task.

1. Introduction

The literature on applying graphics processing units (GPUs) to Particle Swarm Optimization (PSO) is increasing. This is not surprising, as PSO is a relatively simple technique that has shown remarkable potential in solving a wide variety of problems in global optimization. PSO is a population-based stochastic global optimization algorithm that can optimize a wide range of cost functions, including large, complex ones [1, 2]. It simulates cooperative behaviour, as opposed to the competitive genetic algorithm and evolutionary strategy paradigms, and has motivated a great deal of research [3].

The inherent parallelism of PSO can be mapped onto high performance computing (HPC) systems, including distributed clusters [4] and, more recently, onto graphics processing units [5, 6]. Many papers have recently appeared that employ GPUs for PSO [7, 8]. Originally designed for 3D graphics rendering, GPUs are also ideal parallel processors for fine-grained, data-parallel, high-throughput and streaming applications, and exceptionally large speedup can be achieved when computations can be streamed onto them [5]. However, many very high-dimensional complex cost functions, such as those used in "big-data" applications, require expensive linear algebra operations, and often require large external data, such as transformation matrices, or large sets of empirical data. Due to the heavy computation required to evaluate these cost functions, the limited amount of memory available on

[1] To whom any correspondence should be addressed.

GPUs, and CPU-to-GPU data transfer bottlenecks, some combination of CPU and GPU processing would offer potential advantages.

Multicore technology for task-parallel medium-granularity tasks where fewer processing elements perform more intense computations also offers efficiency gains [9]. Due to limited memory bandwidth and inefficient memory management schemes, multicore processing at present offers only limited speedups [10]. Consequently, many-core (tens or hundreds of cores), digital signal processors (DSPs), acceleration processing units (APUs), field-programmable gate arrays (FPGAs) and other accelerators will likely lead to great performance gains in the near future [11]. Comparisons of GPU and multicore performance in PSO are also being investigated [6].

GPU and multicore HPC can be used in tandem, wherein each component works on tasks to which it is best suited. These heterogeneous systems are progressively gaining a foothold in many scientific and engineering applications (e.g. [12]). Heterogeneity is achieved by multicore parallelism complemented with accelerators (e.g. GPUs, Cells, DSP chips). This approach is taken in the current paper to increase the efficiency of high-dimensional and difficult global optimization of complex cost functions, wherein parallelization is achieved according to Schnabel's taxonomy [13]. Highly data-parallel but computationally simple tasks were accelerated with GPUs, while more intensive calculations were made more efficient through multicore processing. Assigning tasks to the different types of processors, while intuitive, was verified by profiling analysis. This paper demonstrates that implementing a heterogeneous, problem-specific approach can achieve good performance gains, and address the shortcomings of using strictly multicore (not scalable, not ideal for very fine-grained tasks) or GPU (not ideal for long computations requiring large data).

2. Adaptive Particle Swarm

The guiding principle of PSO is updating the *D*-dimensional positions of *N* independent "particles" that explore the *D*-D space in search of an optimum solution. These updates are based upon the progress of the search. Each particle's velocity update at time $t + 1$ is updated as:

$$\underbrace{\mathbf{v}_i(t+1)}_{\text{Updated velocity for particle } i} = \underbrace{\omega(t)\mathbf{v}_i(t)}_{\text{Inertia, affected by previous velocity}} + \underbrace{\overbrace{C_1\varphi_1}^{\text{Local acceleration factor} \times \varphi_1 \sim U(0,1)} \left(\mathbf{p}_i^{\text{best}}(t) - \mathbf{x}_i(t)\right)}_{\text{Effect of particle } i\text{'s best position found (local effect)}} + \underbrace{\overbrace{C_2\varphi_2}^{\text{Global acceleration factor} \times \varphi_2 \sim U(0,1)} \left(\mathbf{g}^{\text{best}}(t) - \mathbf{x}_i(t)\right)}_{\text{Effect of the globally best particle with the best function evaluation}} \quad (1)$$

Stochasticity, provided by random factors φ_1, $\varphi_2 \sim U(0, 1)$, maintains diversity in the population. After the new velocity is computed, the position of the *i*-th particle is updated as:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \quad (2)$$

Many variations of particle swarm have been proposed since its inception. Such variations include hybridization with genetic and evolutionary operators (e.g. [14]), subpopulations [15; 16], hybridization with other optimization methods [17], and adapting various components of the PSO algorithm itself (e.g. [18]). A particularly promising approach is to adjust the inertial factor $\omega_t = \omega(t)$ in Eq. 1 (see [19] for a discussion of these methods and references). In the current paper, a specific adaptive PSO (henceforth, APSO) is considered, in which the inertial factor is related to particle clustering [20]. Specifically, in each iteration *t*, the inertial factor is defined by:

$$\omega_t(\phi) = \frac{1}{1+1.5e^{-2.6f}},\ \phi = \frac{d_g - d_{\min}}{d_{\max} - d_{\min}},\ \text{and}\ d_i = \frac{1}{N-1}\sum_{j=1,\, j\neq i}^{N}\sqrt{\sum_{k=1}^{D}\left(x_i^k - x_j^k\right)^2}. \quad (3)$$

Here, the d_i represent the mean distance of particle $\mathbf{x}_i$ to every other particle (d_g for the globally best particle), requiring a distance matrix to be computed. Also, $d_{\min} = \min(d_i)$, $d_{\max} = \max(d_i)$, $i = 1, \ldots, N$, and ϕ is an evolutionary factor calculated to determine the state of the search based on particle clustering. In this formulation, $\phi \in [0, 1]$ so that $\omega(\phi) \in [0.4, 0.9)$. APSO further uses ϕ in a fuzzy membership function to determine the current state of the search, with small values ($\phi < 0.3$) generally indicating a convergent state of the swarm, followed by exploitation ($0.2 < \phi < 0.6$), exploration with more global searching ($0.4 < \phi < 0.8$), and large values ($\phi > 0.7$) indicating that the swarm needs to escape from local optima. The acceleration coefficients C_1 (local) and C_2 (global) are also modified on this basis [20]. In the convergence state, "elitist learning", where random dimensions of the globally best particle are perturbed, is also employed to discourage premature convergence. APSO has shown promise in both unimodal and multimodal problems, with adaptations based on particle positions and current search state, rather than on the basis of time (iteration) alone.

3. Methods

In his seminal paper, Schnabel [13] lists three potential levels of parallelism in global optimization; parallelization of: (1) the cost function; (2) any linear algebra operations, and; (3) the optimization algorithm itself. Schnabel focused primarily on multiple-instruction-multiple-data (MIMD) parallelism, indicating that single-instruction-multiple-data (SIMD) architectures are insufficiently general due to conditionals, branching, etc. in the cost function. Although SIMD systems have improved markedly since 1995, GPU-exclusive methods may not be conducive to evaluating many expensive cost functions. Consequently, in this paper, cost function evaluation is primarily assigned to CPUs, with SIMD (e.g. GPUs) used for linear algebra and high-throughput operations [13]. All multicore parallelism was achieved with OpenMP.

3.1. Cost functions

Two cost functions, one 200D composite and one realistic (but simplified) problem from geophysics, were optimized with heterogeneous APSO. Ground truth values for these functions were known.

3.1.1. Composition function (200D). Composition functions, weighted sums of transformed search spaces for standard test functions (e.g. Rastrigin, Weierstrass, etc. [20]) were proposed to test PSO in highly irregular, nonconvex space. These functions are characterized by multiple optima, and are extremely difficult to optimize, except in very low dimensions ($D < 10$). The general definition for these functions, $f_{comp}(\cdot)$, is given as [21]:

$$f_{comp}(\mathbf{x}) = f_{bias} + \sum_{m=1}^{M} w_m \left[f'_m\left(\mathbf{T}_m \left(\frac{\mathbf{x} - \mathbf{o}_m}{\lambda_m} \right) \right) \right], \text{ where } w_m = \exp\left(- \frac{\sum_{k=1}^{D} \left(x_d - o_{m,d} \right)^2}{2D\sigma_m^2} \right) . \quad (4)$$

Here, M denotes the number of individual functions in the composition. Each weight is computed for each individual function and for each particle on the basis of a standard deviation σ_m for each of the M functions, and $f'_m(\cdot)$, $m = 1, \ldots, M$, is $f_m(\cdot)$ scaled by the $\max[f_m(\cdot)]$ within the search space and by a constant, $\mathbf{o}_m$ is a vector of offsets of length D, $\mathbf{T}_m$ is a dense $D \times D$ orthogonal matrix, and λ_m is a scaling constant for each individual function. In this paper, $D = 200$, and $M = 5$. The individual cost functions are: {Rastrigin, Weierstrass, Ackley, Griewank, Weierstrass} (see [19, 21] for definitions). Because of the dense linear transformation in 200D, the search space is very complex, and therefore the goal is to determine a near-optimal solution, or at least a very good local optimum.

3.1.2. Geostatic correction (20D). A synthetic 20D geophysics optimization problem was implemented. In this problem, known as geostatic correction (GSC), subsurface images of geologic data are constructed from seismic reflection surveys. Because of noise or differences in surface materials, the images may become distorted, which may be corrected by shifts ("static corrections"). This problem is very difficult due to its high degree of nonlinearity and the presence of many local optima [22]. Genetic, evolutionary, and other approaches have been previously employed [22; 23]. The situation may be formulated mathematically as (see [22] for a more thorough description):

$$\begin{aligned} A_1 &= B_{r1,c1} - x_{c1} \\ A_2 &= B_{r2,c2} - x_{c2} \\ p_{r1,r2,c1,c2} &= \begin{cases} \left(A_1 + O_{r1,c1}\right) - \left(A_2 - O_{r2,c2}\right), & A_1 \le A_2 \\ \left(A_2 + O_{r2,c2}\right) - \left(A_1 - O_{r1,c1}\right), & \text{otherwise} \end{cases} \\ f(\mathbf{x}) &= -\sum_{c1=0}^{D-2} \sum_{c2=c1}^{D-1} \sum_{r1=0}^{Z-1} \sum_{r2=0}^{Z-1} p_{r1,r2,c1,c2} \end{aligned} \tag{5}$$

Here, a matrix of base values, $\mathbf{B}_{55\times20}$ consists of 20 plates (the number of dimensions) at 55 levels deep, and an offset matrix, $\mathbf{O}_{55\times20}$.

3.2. Parallelization of APSO

The entire APSO system was analysed to determine the most computationally intensive components, and the potential parallelism of each component was assessed. A profiling tool, *AMD CodeAnalyst*, collects samples uniformly as the program executes [24]. From the profiling results of the 200D composition with N = 8192, the most time-intensive components were the distance matrix (about 57%), matrix multiplication (about 16%), and the Weierstrass evaluation (about 9%), followed by the particle and weight update sections (each less than 1%). In the 20D geostatic correction problem, the largest component was the cost function itself (about 99%), followed by the distance matrix (about 1%). It was therefore decided to investigate parallelization of: (1) particle updates (Eqs. 1-2); (2) weight updates (Eq. 4); (3) the Weierstrass component to the composition cost functions (Eq. 4); (4) matrix multiplication (Eq. 4), (5) distance matrix calculation (Eq. 3), and (6) geostatic correction cost function (Eq. 5). These components are now described using Schnabel's taxonomy.

3.2.1. Parallelization of the cost function. The composition function is parallelized through the matrix multiplication (see below), and, because of the number of conditionals required, is not a candidate for further parallelization. However, the Weierstrass component of these compositions requires a moderate amount of computation, and was therefore parallelized. The geostatic correction problem is a straightforward four-level nested loop, and was therefore analyzed for parallelization.

3.2.2. Parallelization of linear algebra operations. Each evaluation of the composition function requires search space rotation (the dense $\mathbf{T}_m$ matrix in Eq. 4). The NVIDIA CUDA framework (used in this paper) supplies a linear algebra toolkit, cuBLAS, implementing BLAS routines optimized for the GPU. As a SIMD problem, matrix multiplication is an ideal task for the GPU.

3.2.3. Parallelization of the algorithm (distance matrix, weights updates, and particle updates). Calculating the d_i in Eq. 3 requires a distance matrix (DM). In [20], a small number of particles (N = 20) was used, and, consequently, the calculation was not considered to be intensive. However, forming a DM has $O(N^2)$ complexity, with $(N^2 - N)/2$ distances to be calculated, and can be intensive for large N. The DM can be parallelized via multicore by indexing the upper-triangular matrix. A

GPU-based method [25] was also implemented. Timing results for parallel methods were compared to the two-level nested loop method that directly computes the upper-triangular DM efficiently on one core, but, because of the load imbalance, cannot be easily parallelized. Weight and particle updates are straightforward loop parallelizations.

3.3. CUDA implementation

GPU kernels (routines that run on the GPU) were developed in the NVIDIA CUDA (Compute Unified Device Architecture) framework for GPU programming. CUDA also simplifies the transfer of information between RAM and the GPU-specific RAM. Parallelization is based on a number assigned to each thread. OpenCL, a specification from the Khronos Group, is an alternative framework that supports a wide variety of devices, including GPUs, multicore, and FPGAs, and has also been used for PSO parallelization by other researchers (e.g. [6]).

3.4. Experiments

Experiments were performed with populations of sizes N = 1024, 2048, 4096, and 8192 particles. For the component trials described below, dimensions of D = 16, 128, and 1024 were analysed. APSO trials were run on a computing cluster (SHARCNET) with characteristics of a commodity PC, as only one server unit was employed. Each unit contains two quad-core CPUs (Intel E5430) running at 3.0 GHz with 8 GB RAM, connected to 2 GPU servers, each connected to one NVIDIA Tesla S1070 graphics card with 4 GB GPU memory.

3.4.1. Speedup of APSO components. Speedup results for each individual parallelized APSO component described in Sections 3.2.1-3.2.3 were obtained as the mean of 100 tests. On the GPU, the DM was computed with an efficient algorithm [25] and reduced on the CPU, as the average distances (Eq. 3) were much more efficiently computed this way as opposed to GPU reduction.

3.4.2. Speedup of APSO experiments. Speedup results for full APSO trials for the two cost functions were obtained for the following configurations: (1) multicore, with no GPU acceleration; (2) composition function, multicore parallelization (including DM), cuBLAS acceleration for the search space transformation; (3) multicore parallelization with GPU acceleration of the DM computation.

3.4.3. Timing results of APSO experiments. Computation times for 500-iteration trials were obtained for: (1) no parallelization (1 core for all operations), and; (2) eight cores, GPU-accelerated DM calculation, and GPU matrix multiplication (composition function).

4. Results

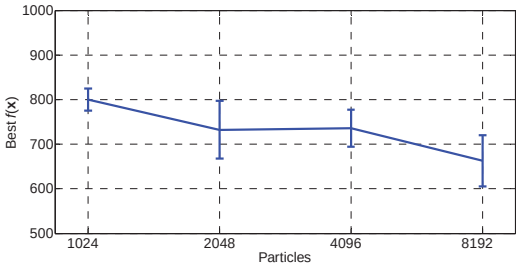
Both cost functions were successfully optimized by APSO (figure 1). Because of the complexity of search space of the 200D composition function, a larger population size results in a better global solution. The 20D GSC problem was successfully optimized with only 1024 particles, but, with more realistic scenarios, the more complex cost function would benefit from larger N.

The individual APSO components were timed with 1, 2, 4, and 8 cores, and analysed for population sizes of N = 1024, 2048, 4096, and 8192, and dimensions of D = 16, 128, and 1024. Speedup results are shown in figures 2a-e, optimized cuBLAS matrix multiplication (composition) speedups are shown in figure 2f, and multicore DM calculations in figure 2g.

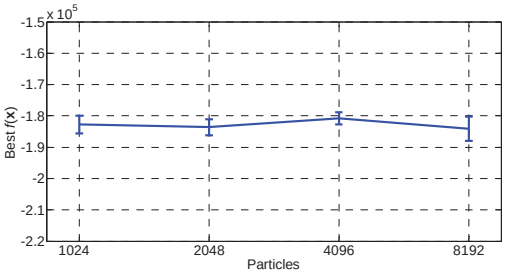
The full APSO algorithm with CPU-based DM computation was run with the 200D composition function with cuBLAS-accelerated rotational matrix multiplication (figure 3a). The 20D geostatic correction function was run with the CPU-based DM technique (figure 3b). Finally, both cost functions were run with the GPU-accelerated DM (and the GPU-accelerated matrix multiplication for the composition function) (figures 3c-d). Timing results for two configurations – one core, no parallelism vs. eight cores with GPU acceleration, are shown in Table 1. For the composition, speedup is shown for (1) 8-core with cuBLAS on the GPU, and (2) 8-core with cuBLAS and DM on the GPU.

For GSC, speedup is shown for (1) 8-core, and (2) 8-core with DM on the GPU. As their speedups and percentage of total computation were both low, weight updates were not parallelized.

From these results, it is seen that the heterogeneous approach achieves better speedup than multicore alone for the composition function. Because of their sizes, and given current limited GPU memory and CPU-GPU bandwidth bottlenecks, these problems are not runnable exclusively on the GPU, and, likely, other accelerators as well. GSC optimized faster with multicore (the problem is too large to fit all on the GPU). Therefore, for large problems, such as those arising in "big data" applications, analysis of the various optimization components, per Schnabel, leads to heterogeneous solutions that achieve acceptable levels of performance and efficiency gains.

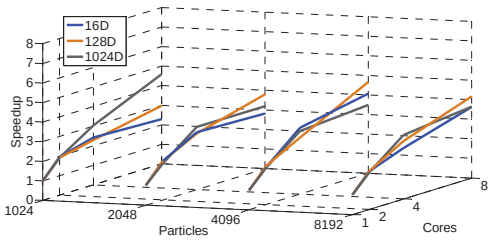


(a) 200D composition function.

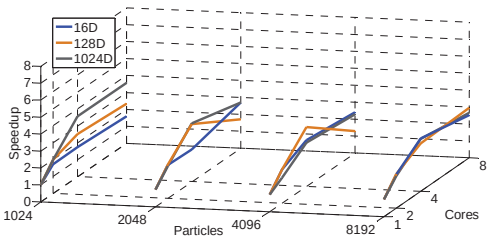


(b) 20D geostatic correction problem.

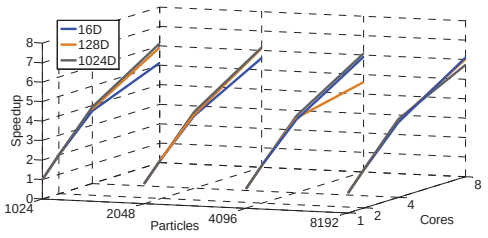
Figure 1. Best $f(\mathbf{x})$ cost function values returned by the globally best particle with varying N.



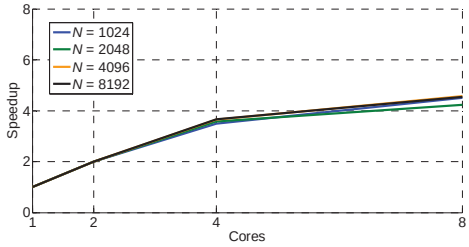
(a) Particle update.



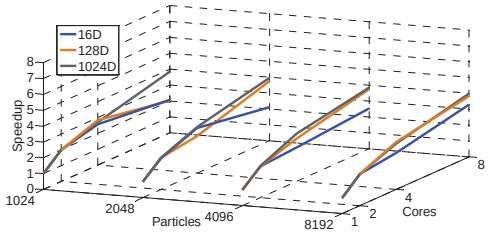
(b) Weight calculations (composition function).



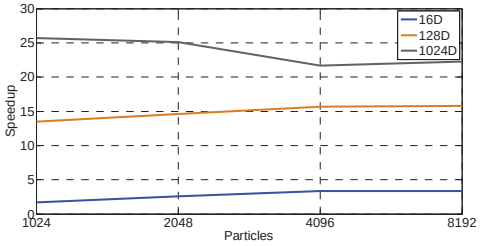
(c) Weierstrass function.



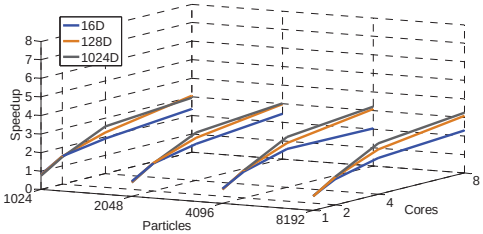
(d) Geostatic correction cost function.



(e) Multicore matrix multiplication.

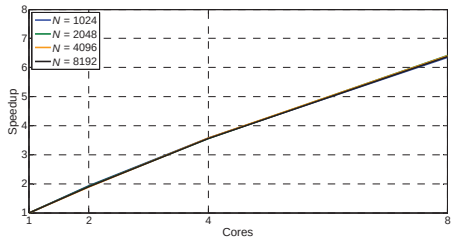


(f) cuBLAS matrix multiplication.

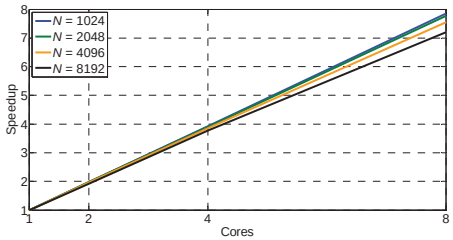


(g) Multicore distance matrix calculation.

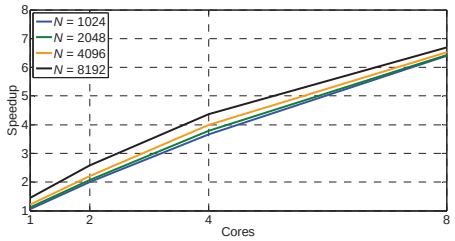
Figure 2. Speedups for individual APSO components with varying *N* and *D*.



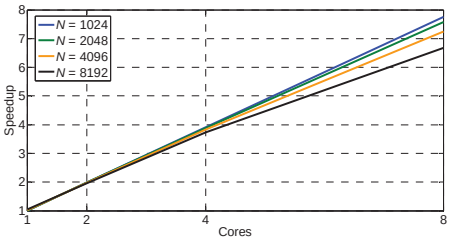
(a) Multicore, cuBLAS, 200D composition.



(b) Multicore, 20D GSC.



(c) Multicore, cuBLAS GPU DM, composition.



(d) Multicore, GPU DM, GSC.

Figure 3. APSO speedups.

5. Discussion

Although the speedups are consistent between trials using the GPU-accelerated DM and those using multicore parallelization (figures 3a, c), the timings for the GPU-accelerated method are much better than the multicore DM method (Table 1) for the high-dimensional composition, but are slightly worse for the GSC. From the profiling results, it is seen that computing $f(\mathbf{x})$ for GSC is the most intensive operation, and overwhelms the expensive DM calculation. Because GSC is 20D, the overhead incurred by data transfer slightly outweighs efficiency gains. For the 200D composition, the greater amount of work results in better speedup. In any case, parallelization in one form or another is required for DM. Additionally, although DM is a large part of APSO, there are other expensive components as well (e.g. complex cost function evaluation). Computation times for APSO were reduced from the single-core, no GPU configurations to the 8-core parallelized, with GPU-based DM calculation configurations by rates of 6.72 – 7.75 for GSC, and 6.49 – 6.85 for the composition function with cuBLAS matrix multiplication (see Table 1). Even better speedups are obtained when comparing the fully parallelized composition (with cuBLAS acceleration and GPU-based DM calculation) to the non-parallelized (single-core, no GPU) configuration, where speedups of 8.45 – 8.67 are achieved.

Cache size also has an effect on performance. Some speedup drops are seen in individual APSO components for moderate and high dimensions (see figures 2a-c), likely because the small cache on the

CPU cannot keep larger problems in cache, resulting in accesses to slower RAM. It is also seen that, in some cases, smaller N show a greater speedup than for larger populations (figures 2a-b). A plausible explanation is that as the size of the problem increases, less cache memory can be used, forcing access to higher, slower levels of cache, or even into RAM. However, the lower gains made by using multicore are justified because if the GPU is exclusively utilized, the CPU would be mostly idle. Furthermore, although the GPU has many cores, they are inherently weak and memory-limited vis-à-vis the CPU, which limits them to certain types of parallel problems. As access to RAM is very slow, required data may be preloaded, but only a small amount can fit into GPU memory.

Table 1. Mean (± std. dev.) optimization time (seconds) for cost functions by number of particles.

	Particles			
Cost Function / Configuration	1024	2048	4096	8192
GSC (1 CPU)	1050.37 ± 2.75	2125.61 ± 2.81	4329.69 ± 9.11	9004.92 ± 14.52
GSC (8 CPUs)	133.66 ± 0.28	273.73 ± 0.28	574.48 ± 0.81	1249.26 ± 1.17
GSC (8 CPUs-GPU DM)	135.57 ± 0.44	280.58 ± 0.57	597.20 ± 0.99	1340.92 ± 2.10
Speedup (8 CPUs/8 CPUs-GPU)	**7.86 / 7.75**	**7.76 / 7.58**	**7.54 / 7.25**	**7.21 / 6.72**
Comp. cuBLAS (1 CPU)	1149.52 ± 10.70	2454.29 ± 24.88	5483.02 ± 11.53	13460.55 ± 43.68
Comp. cuBLAS (8 CPUs)	178.87 ± 2.22	377.90 ± 3.80	844.23 ± 2.24	2067.58 ± 9.60
Comp cuBLAS (8 CPUs-GPU DM)	177.11 ± 0.50	374.80 ± 1.74	831.13 ± 5.97	1964.22 ± 8.75
Speedup (8 CPUs/8 CPUs-GPU)	**6.42 / 6.49**	**6.49 / 6.55**	**6.49 / 6.60**	**6.51 / 6.85**
Composition (1 CPU)	1535.99 ± 16.40	3206.76 ± 5.55	7060.97 ± 85.63	16588.32 ± 140.65
Composition (8 CPUs)	224.88 ± 1.02	471.10 ± 3.53	1028.84 ± 4.67	2434.49 ± 5.80
Composition (8 CPUs-GPU DM)	177.11 ± 0.50	374.80 ± 1.74	831.13 ± 5.97	1964.22 ± 8.75
Speedup (8 CPUs/8 CPUs-GPU)	**6.83 / 8.67**	**6.81 / 8.56**	**6.86 / 8.50**	**6.81 / 8.45**

6. Conclusion

The results presented in this paper are encouraging for applying heterogeneous computing to difficult global optimization problems using PSO. However, especially as evidenced in the GSC results, one cannot rely on GPU acceleration alone, and the key to increased multicore improvement is tighter memory-CPU integration [10]. Increasing CPU-GPU integration may reduce or eliminate the data transfer bottleneck. It is expected that in the near future, many-core chips and hybrid CPU/GPU processors, as well as greater exploitation of DSP and FPGA capabilities, will lead to further efficiency improvements in computationally-intensive applications, such as high-dimensional global optimization. These heterogeneous frameworks will also be instrumental in new "big-data" applications, so that "big-data supercomputing on a budget" will be more attainable.

Future work will include further improving the robustness and efficiency of PSO. For the lower-dimensional GSC problem, the effect of decreasing the population size and increasing the number of iterations to the overall efficiency will be assessed. For all problems, after 500 iterations, the population size can be decreased, or an efficient local method (e.g. Powell's method, multidirectional search) can be used to refine the best optimum found by APSO [26]. In addition, the intensive nested loop GSC cost function will be revisited for possible GPU execution. In future GPUs and other accelerators, it is expected that tighter accelerator/CPU connections will alleviate some of the bandwidth difficulties currently experienced in large problems. Finally, although PSO is robust for

many classes of problems, many advanced techniques, including surrogate response surface methods and those using gradient information, will also be analyzed for efficiency gains using heterogeneous hardware, especially many-core systems with GPU or DSP acceleration.

Acknowledgments
The authors thank Dr. Renata Wachowiak-Smolíková and Devin Rotondo for helpful comments and criticisms. This work was supported by SHARCNET facilities (www.sharcnet.ca). The authors are supported by the Natural Sciences and Engineering Research Council of Canada (MPW, #386586-2011).

References

[1] Blum C and Merkle D 2008 *Swarm Intelligence: Introduction and Applications* (New York: Springer-Verlag)
[2] Engelbrecht A P 2006 *Fundamentals of Computational Swarm Intelligence* (New Jersey: John Wiley & Sons)
[3] Schutte J F and Groenwold A A 2005 *J Global. Optim.* **31** 93–108
[4] Schutte J F, Reinbolt J A, Fregly B J, Haftka R T and George A D 2004 *Int. J. Numer. Meth. Eng.* **61(13)** 2296–315
[5] Hung Y and Wang W 2012 *Optimization Methods and Software* **27** 33-51
[6] Cagnoni S, Bacchini A and Mussi L 2012 *Applications of Evolutionary Computation* **7248** 406-415
[7] Mussi L, Daolio F and Cagnoni S 2010 *Information Sciences* **181** 4642-57
[8] Wachowiak M P and Lambe Foster A E 2012 *J. Phys.: Conf. Ser.* **385**
[9] Slabaugh G, Boyes R and Yang X 2010 *IEEE Signal Processing Magazine* **27-2** 134-8
[10] Moore S K 2008 *IEEE Spectrum* **Nov** 15
[11] Moore S K 2011 *IEEE Spectrum* (New York: Institute of Electrical and Electronics Engineers) **Jan** 40
[12] Kindratenko V V and Trancoso P 2011 *Computing in Science & Engineering* **13** 92-5
[13] Schnabel R 1995 *Parallel Computing* **21** 875-905
[14] Shi X, Liang Y, Lee H, Lu C and Wang L 2005 *Information Processing Letters* **93** 255-61
[15] Lovbjerg M, Rasmussen T K and Krink T 2001 *Proc. 3rd Genetic Evolutionary Computation Conf. (California)* 469-76
[16] Veeramachaneni K, Peram T, Mohan C and Osadiciw L A 2003 *Proc. Genetic and Evolutionary Computation Conf. 2003 (Illinois)* 110-21
[17] Montes De Oca M A, Stutzle T, Birattari M and Dorigo M 2009 *IEEE Transactions on Evolutionary Computation* **13** 1120-32
[18] Banks A, Vincent J and Anyakoha C 2008 *Natural Computing* **7.1** 109–24
[19] Nickabadi A, Ebadzadeh M M and Safabakhsh R 2011 *Applied Soft Computing* **11** 3658-70
[20] Zhan Z-H, Zhang J, Li Y and Chung H S-H 2009 *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **39.6** 1362-81
[21] Liang J, Ponnuthurai J, Suganthan N and Deb K 2005 *Proc. 2005 IEEE In Swarm Intelligence Symposium (California)* 68-75
[22] Mathias K E, Whitley D D, Stork C and Kusuma T 1994 *Proc. of the First IEEE Conf. on IEEE World Congress on Computational Intelligence (Florida)* **1** 356-61
[23] Whitley D, Lunacek M and Sokolov A 2006 *Lecture Notes in Computer Science* **4193** 988-997
[24] Moore R 2011 *IS&T/SPIE Electronic Imaging* (pp. 78720L-78720L)
[25] Chang D, Jones N A, Li D, Ouyang M, Ragade R K 2008 *Proc. of the IASTED Int. Symposium on Computational Biology and Bioinformatics* 278-283.
[26] Wachowiak M P and Peters T M 2006 *IEEE Trans Inform. Tech Bio* **10(2)** 344-53