Parallelism vs. Speculation: Exploiting Speculative Genetic Algorithm on GPU

Yanchao Lu
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
chzblych@sjtu.edu.cn

Long Zheng
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
longzheng@sjtu.edu.cn

Li Li
School of Software, Shanghai Jiao Tong University, Shanghai, China
lilijp@cs.sjtu.edu.cn

Minyi Guo
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
guo-my@cs.sjtu.edu.cn

ABSTRACT

Graphics Processing Unit (GPU) shows stunning computing power for scientific applications in the past few years, which attracts attention from both industry and academics. The huge number of cores means high parallelism and also powerful computation capacity. Many previous studies have taken advantage of GPU's computing power for accelerating scientific applications. The common theme of those research studies is to exploit the performance improvement provided by massive parallelism on GPU. Despite that there have been fruitful research work for speeding up scientific applications, little attention has been paid to the redundant computation resources on GPU. Recently, the number of cores integrated in a single GPU chip increases rapidly. For example, the newest NVIDIA GTX 980 device has up to 2048 CUDA cores. Some scientific applications, such as Genetic Algorithm (GA), may have an alternative way to further improve their performance. In this paper, based on the biological fundamentals of GA, we propose a speculative approach to use the redundant computation resources (i.e., cores) to improve the performance of parallel genetic algorithm (PGA) applications on GPU. Comparing to the traditional parallelism scheme, our theoretical analysis shows that the speculative approach should improve the performance of GA applications intuitively. We experimentally compare our design with the traditional parallelism scheme on GPU using three Nonlinear Programming problems (NLP). Experimental results demonstrate the effectiveness of our speculative approach in both execution time and solution accuracy of GA applications on GPU.

PMAM '15, February 7-8, 2015, San Francisco Bay Area, USA

http://dx.doi.org/10.1145/2712386.2712398.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming

General Terms

Algorithms, Design, Performance

Keywords

GPGPU, Speculative Execution, Genetic Algorithm, Performance Evaluation

1. INTRODUCTION

Nowadays compared to the traditional multi-core processors, GPUs offer dramatic computation power because of their highly massive parallel architecture. The number of cores integrated into a GPU is the key factor that affects the performance of GPUs. Recently, the number of cores integrated in a single GPU chip increases rapidly. For example, the newest NVIDIA GTX 980 device has up to 2048 CUDA cores. Besides the number of cores, the GPU architecture evolves quickly. GPU manufacturers are trying to hide more and more hardware specifications so that eventually programmers can write their GPU codes more easily.

In the predictable future, more cores will be integrated in a single GPU chip. More cores mean a single GPU device can support higher parallelism. However, the GPU hardware now is a little ahead the computation needs, that is, the GPU hardware may offer the redundant computation resource for some specific applications. The latest CUDA Platform that simplifies the multiple GPUs management increases the redundancy of computation resources provided by GPU devices. The newest GPU architecture allows multiple kernels running on the GPU simultaneously, that is, a single GPU device allows different applications to share the GPU computation resources. It can be considered as one of solutions to use of the redundant computation resources. Therefore, how to use of so many cores of GPUs efficiently is very critical for GPU computing instead of the skills of fine optimization based on the GPU architecture. We find

Table 1: The results of a GA application during 10 executions

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Optimal Result	7052.47	7054.28	7051.95	7051.92	7054.81	7051.94	7053.22	7052.79	7053.95	7052.17
Execution Time	7858.97	10933.39	14267.79	8943.80	14337.15	11986.93	9177.28	13527.16	12718.70	12301.07

that some scientific applications, such as Genetic Algorithm (GA), may have an alternative way to further improve their performance.

A Genetic Algorithm (GA) is a search heuristic that inspired by natural evolutionary biology, such as inheritance, mutation, selection and crossover [6]. The GA can be effectively used to find approximate solutions for optimization and search problems in an acceptable mount of time. Thus, it is successfully used in business, engineering and science fields [2, 9, 7, 11]. However, as GA applications need huge numbers of individuals that composes a population to search probable solutions with enough generations of evolution, GA applications cost lots of computation capacity. The solution accuracy and execution time strongly depends on the development of computing parallelism. With the emergence of GPU devices, the GA researchers focus on the new massive parallel architecture immediately. Many GA applications are transplanted from clusters to GPUs and get tens or hundreds of speedup.

The previous research mainly concentrates on use of the massive parallelism of GPU devices with traditional parallel genetic algorithm approach [8, 12, 1, 5]. However, the redundancy of computation resources provided by GPU devices is not considered seriously enough. In this paper, we begin with a normal fact that we find in our experiences of implementing GA applications on GPU. Inspired by the fact, we propose a new speculative GA approach on GPU to use of the redundant computation resources of GPU devices more effectively for GA applications. Comparing to the traditional parallelism scheme, our theoretical analysis shows that the speculative approach should improve the performance of GA applications intuitively. We take three classic engineering problems solved by GA applications as our case studies to evaluate the effectiveness of our speculation approach. Experimental results show that the proposed speculative GA approach can use GPU computation resources better than the traditional parallelism approach. Our speculative approach is superior to the traditional parallelism scheme in both execution time and solution accuracy of GA applications on GPU.

Our work offers an alternative approach to use GPU's huge computation resources–speculation rather than parallelism. This approach is not limited only in field of GA applications. The speculation approaches should be effective for the algorithms based on searching with random candidates, e.g., evolution algorithm, neural network and machine learning algorithms. We exploit a new perspective to use GPU's powerful computation capacity, and further get performance improvement by using GPU devices.

The remainder of this paper is structured as follows. Section 2 is the motivation of our work, which presents a fact of GA applications that we find in our experiences of implementing GA applications. We describe our speculative GA approach on GPU and make a theoretical analysis in Section 3. Experimental results are presented in Section 4. Section 5 summaries our findings and points our future work.

2. MOTIVATIONS

In nature, the lifetime of each individual is a procedure in which it compete with others and fits the environment. Only the strongest ones can survive from the tough environment. The survivor individuals mate randomly and produce the next generation. During reproducing the next generation, crossover always occurs, while mutation happens rarely, which makes individuals of the next generation stronger for the tough environment.

Genetic Algorithms are heuristic search algorithms that mimic natural species selection and evolution described above. The problem that a GA application tends to solve is the tough environment. Each individual in the population of a GA application is a candidate solution of the problem.

A generation of a GA is generated by the following steps: **fitness computation**, **selection**, **crossover** and **mutation**. The fitness computation is the competition of individuals, and can tell which individual is good for the problem; the selection choose good individuals to survive and eliminates bad ones; the crossover mates two individuals to produce the next generation individuals; and the mutation occurs after crossover, so that the next generation can be more diverse. With enough generations, GAs can evolve an individual that is the optimal solution to the problem. Since GAs are so similar to the biological species evolution, many theories of GAs are motivated and explained by biological theory.

During our experiences of implementing GA applications on GPU devices, we find a fact that when we run the GA application to find suitable results of an engineering problem with several times, one can hardly get the same result even with the same configuration. Table 1 shows the solution accuracy and execution time of an example GA application that solves an tested engineering problem. In this problem, a smaller result means a higher solution accuracy. We run this GA application for 10 times with a maximum of 50000 generations, and show the best result as well as the execution time that the GA application taken to reach the best result.

From Table 1, we can easily find that the results of the GA application are unstable. For example, the best results of the 3rd and 4th run are almost the same, while the time they consumed to reach the best results are quite different. Moreover, although the time of the 3rd and 5th run are almost the same, the 3rd run gets the highest accuracy, rather than the 5th run gets the lowest accuracy.

There are two reasons for the instability of GA applications. First, the evolution progress in the nature (e.g., mating and mutation) are highly random, which is full of random operations. A little difference in mating or mutation progress will lead to a total different evolution track. Second, the population may evolve into a trap that is hard to jump out, which leads GA applications get the bad results. All above observations are exactly the same as the species evolutions in the nature. There are millions of species because they evolve into different evolution track. Meanwhile, lots of species extinguished because they were trapped and

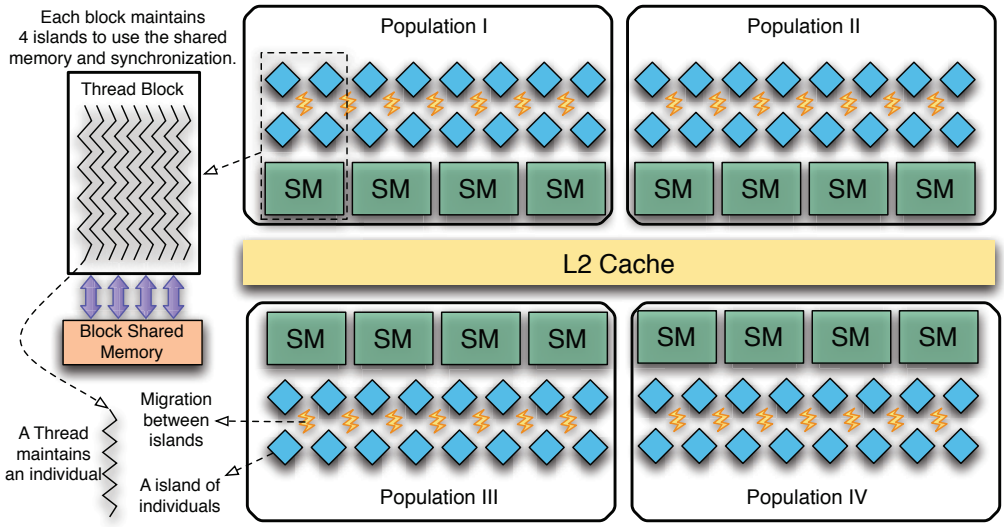


Figure 1: The Speculation Methodology that Implements GAs on a GPU.

evolved into the dead end.

Although we can develop some rules for mating and mutation to improve the performance of GA applications, one can not manipulate the progress of mating and mutation in GA applications. This progress highly depends on randomness, so that what we can do is to accept the instability of GA applications. Actually, this is also the biological fundamental of GA. Lots of biologists, philosophers and even religionists have been discussing whether Darwin's theory is right, one of which is "if the all species on earth evolve again, our world may be totally different.". Whether this statement is right or not, it should be right in fields of GA applications.

3. A SPECULATIVE GA SCHEME ON GPU

The traditional GA applications on GPU devices usually use island-based parallel genetic algorithm (PGA) models to make usage of GPU cores. In a typical PGA model, the population of a GA application is divided into islands, which can be considered as sub-populations. The individuals within a island evolve separately. Every a pre-defined number of generations, individuals in different islands will exchange, which is called **migration**. The island mechanism is designed to map GA applications to parallel computing devices easily and reduce the communication overhead as much as possible. Compared to the original GA schemes, PGA schemes reduce the execution time so that lots of problems can be solved within an acceptable time, at the cost of decreasing the solution accuracy with a same generations.

The island-based PGA model is perfectly fit for the GPU architecture, which has been shown the most effective approach on GPU [13]. Although a GPU device has hundreds of cores, they are organized into Streaming Multiprocessors (SMs). The threads running on cores in a SM can share the fast on-chip shared memory, while the communication between SMs are quite expensive. The classic implementation of island-based PGA models on GPU is that each block of threads maintains an island, and each block is further assigned to a particular SM. Island-based PGA models try to reduce the communication overhead between islands caused by **migrations**. Thus, the communication overhead between SMs is not much.

Because of GPU's powerful computation capacity, GPU offers two or even three orders of magnitude speedups compared to the multi-core processors. Currently, the implementations of GA applications on GPU devices mostly follow island-based PGA models [8, 12, 1]. However, GPU has too powerful computation capacity. In island-based PGA models on GPU, each thread represents a individual. The newest NVIDIA GTX 980 GPU can support over 20,000 threads, which means the size of a GA application's population on a single GPU device can exceed 20,000 individuals. The essential number of individuals is related to the number of variables of the problem that a GA application tries to solve. In general, hundreds or thousands of individuals are enough to get a good result in reasonable generations [3]. The computation capacity of GPU devices for GA applications is obviously redundant now.

Therefore, how to make usage of the redundant computation resources of GPU devices for GA applications effectively now is very critical. As we analyze in Section 2, the results of GA applications are not stable. When running the GA application on GPU, we wish our GA application can get benefits rather than suffer from the instability.

We propose a new GA approach on GPU that is based on the speculation thinking, so that the GA applications on GPU can gain the benefits from the instability, which leads to improve both the solution accuracy and execution time. In short, we split GPU SMs into groups. Each group maintains a separate population of the GA application, which is independent with each other. The islands of each population still depends on the number of SMs in a group. The more groups of SMs are partitioned, the more opportunities that the GA application can try to get a better result.

Figure 1 illustrates the basic concept of our speculative GA approach on GPU. In Figure 1, the GPU device has 16 SMs which is partitioned into four groups. Therefore, four separate populations can evolve simultaneously, which indicates that the GA application can get four speculative results during each running. After the four populations

Table 2: Speculative Genetic Algorithm Configurations With Different Values of CP

CP	1	2	4	8	16
Individuals per Island	64	64	64	64	64
Islands per Population	64	32	16	8	4
Individuals per Population	4096	2048	1024	512	256
Populations	1	2	4	8	16
Total Individuals	4096	4096	4096	4096	4096

evolve to a pre-defined generations, we choose the best result among the results of four populations provided. Actually, we also can split the 16 SMs into different number of groups. We introduce the Configure Parameter (CP) to represent the number of groups in the GPU device. For example, when the value of CP is 4, it implies that the GA application can get 4 speculative results during each running. When the value of CP is 1, it is the traditional island-based PGA scheme on GPU. The candidate CP can be set to 2^n, which means the value of CP can be 1, 2, 4, till the maximum number of blocks that a GPU can support. As the value of CP increases, we can get more speculative results, while the size of each population decreases.

The size of population is very important for GA applications. If the population size is too small, the optimization space of a GA application is too small so that it evolves very slowly. Thus, individuals in the population can easily evolve to a bad result. On the contrary if the population size is too large, it will not offer the corresponding performance improvement, which is a waste of computation resources on GPU. Besides, with the speculation methodology, the number of speculative results is another factor that affects the performance of GA applications. The more speculative results means that the GA application has high probability to obtain a better result.

Therefore, if the value of CP is too large, we can get enough speculative results to get the benefits from the instability of GA executions, but the size of each population may be too small which leads to the individuals in the population traps into bad results. Oppositely, if the value of CP is too small, the size of population can be guaranteed, but the effect of speculation is weak. Additionally, the size of each population perhaps is too big that the precious computing resources are wasted. With the analysis above, a suitable value of CP is critical to the performance of GA applications on GPU. In Section 4, we demonstrate that our speculative GA approach is superior to the traditional parallelism scheme (i.e., the value of CP equals 1) in practice. Our future work will focus on the relationship between the performance and the value of CP.

4. EVALUATION

In this section, we present the quantitative evaluation of our proposed speculative approach for solving GA applications on GPU.

4.1 Methodology

In order to evaluate our speculative GA approach, we choose three Nonlinear Programming problems (NLP) as our benchmark, which are widely used to evaluate the performance of different GA schemes or other optimization algorithms [10, 4, 13]. The detailed descriptions of these test problems can be found in Appendix A.

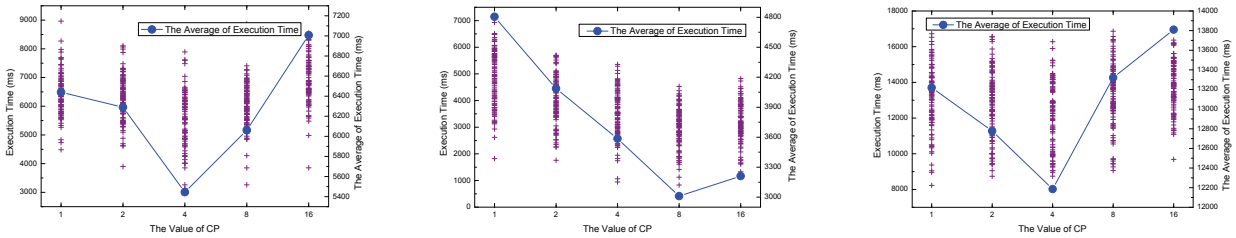
We use a NVIDIA GTX 580 GPU device that is the Fermi architecture to evaluate our speculative GA approach. The GTX 580 GPU device consists of 512 CUDA cores that are organized into 16 SMs. Each SM has 32 CUDA cores. The Fermi architecture allows programmers to set the configuration of L1 Cache and Shared Memory in a SM. In our evaluation, we set the configuration to 48KB/16KB, which means the size of L1 Cache is 48KB while the size of Shared Memory is 16KB. Compared to the configuration that is 16KB/48KB, we find that a larger L1 Cache can provide a better performance of GA applications on GPU.

When implementing those three Nonlinear Programming problems on GPu, each island consists of 64 individuals, four islands are organized into a block, and we initialize 16 blocks in total. Hence, the value of CP can be 1, 2, 4, 8 and 16. With different values of CP, we have different number of populations. However, we keep the number of individuals in all populations the same (i.e., 4096). The detailed configurations of GA applications with different values of CP are shown in Table 2.

4.2 Experimental Results

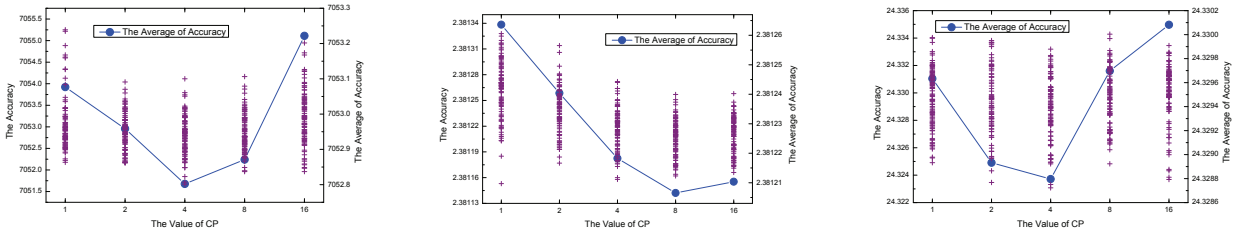
In this section, we evaluate the performance of our speculative GA approach, and compare it with the traditional parallelism one. 100 runs are performed for each value of CP in order to assure to get the stable results. The performance of different GA schemes are measured by execution time and result accuracy. When the value of CP is 1 (i.e., only one population exists on GPU), our speculative approach becomes the traditional parallelism scheme, which is similar to the Hierarchy (Async) PGA approach developed in [13].

We select a pre-defined acceptable result that is 0.1% close to the optimal result for each test problem, and evaluate the execution time that the GA application can reach the pre-defined accuracy. Figure 2 shows the execution time of 100 runs, which is a combination of scatter and line chart. Each + of the scatter chart represents the execution time of each run, and the solid circle symbol on lines shows the average execution time of 100 runs when the value of CP varies from 1 to 16. For Test Problem 1 and 3, when the value of CP increases from 1 to 4, the average execution time decreases significantly. And for Test Problem 2, the average execution time decreases as the value of CP varies from 1 to 8. When the value of CP increases, more populations can evolve simultaneously. Thus, the GA applications have more opportunities to reach the pre-defined accuracy as soon as possible. This also demonstrates that our speculative GA approach outperforms the traditional parallelism scheme (i.e., the value of CP equals 1) in practice. However, when the value of CP is greater than 4 (8 for Test Problem 3), the average execution time becomes longer as the value of CP increases. Although we can get more speculative re-



(a) Test Problem 1 (b) Test Problem 2 (c) Test Problem 3

Figure 2: The execution time of reaching a pre-defined accuracy.



(a) Test Problem 1 (b) Test Problem 2 (c) Test Problem 3

Figure 3: The solution accuracy with a fixed number of generations (50,000 Generations).

sults when the value of CP is more larger, the size of each population is too small that the speculation effect cannot compete the negative effect of small population sizes.

Most GA applications are set a fixed generation for evolution to get a suitable result. Therefore, we also conduct experimental studies in which we set 50,000 generations to solve each test problem. Figure 3 shows the solution accuracy of each test problem with different values of CP after a 50,000-generation evolution. Similar to Figure 2, scatter and line charts are used to represent the solution accuracy of each run and the average solution accuracy, respectively. Thanks to the speculative execution, we observe the best solution accuracy are obtained when the value of CP is 4 for Test Problem 1 & 3, and 8 for Test Problem 3, respectively. When the value of CP becomes too larger, the solution accuracy drops significantly. This observation is consistent with our findings in Figure 2. In a word, a suitable value of CP is sensitive to the problem's characteristics. Our future work will focus on the relationship between the performance and the value of CP.

Finally, we show a detailed comparison between our speculative approach and the traditional parallelism scheme using Test Problem 2. All data in Tables 3 and 4 are the average value of 100 runs.

From Table 3, we observe that compared to the traditional parallelism scheme, our speculative approach can save up to 10,274 generations that is 1,795 ms to reach the pre-defined accuracy. Overall, our speculative approach outperforms the traditional parallelism scheme with the maximum speedup of 1.6.

Table 4 indicates that the solution accuracy also improves when the speculation methodology is used, which is only $+4.7 \times 10^{-5}$ away from the optimum solution. However, the result of the traditional parallelism scheme is $+10.4 \times 10^{-5}$ away from the optimum solution. We also observe that, except for when the value of CP is 16, the total execution times of 50,000 generations of speculation and parallelism approaches are similar, which means our speculation methodology does not introduce any overhead. When the value of CP is 16, all islands of a population are within a single block. Thus, all data exchanges during migrations are in the shared memory of the GPU. Nevertheless, when the value of CP is not 16, islands of a population are in two blocks at least, which means migration operations need to access the global memory. The global memory access is 100 times slower than the shared memory one. However, the number of threads on GPU is big enough to hide most the global memory accesses, so there is only one millisecond difference, which we can omit reasonably.

5. CONCLUSION

Currently, existed GA applications on GPU mostly exploit the massive parallelism provided by GPU devices to improve their performance. However, GPU can offer more and more computation capacity with its fast development in the future. In this paper, we start with the biological fundamentals of GA, and show that the results of GA applications are unstable across each execution. Different from the traditional parallelism methodology, we propose a speculative approach to get benefits from the instability of GA applications. With our theoretical analysis, the speculative approach can make usage of the redundant computation resources of GPU devices more efficiently. Thus, the performance of GA applications can be further improved on GPU intuitively. Our experimental results show that the speculative approach outperforms the traditional parallelism scheme both in execution time and the solution accuracy. Our future work will focus on the relationship between GA applications' performance and the value of CP, so that we can help researchers and engineers to use our speculation methodology to achieve a better performance in practice.

6. ACKNOWLEDGMENTS

Table 3: The comparison of execution time our speculative approach and the traditional parallelism scheme

Methodology	Parallelism	Speculation			
CP	1	2	4	8	16
Generation	27324	23230	20302	17050	18179
Time (ms)	4805	4086	3585	3010	3210
Speedup	1	1.17	1.34	1.60	1.50

Table 4: The comparison of solution accuracy between our speculative approach and the traditional parallelism scheme

	Parallelism	Speculation			
CP	1	2	4	8	16
Result	2.38126	2.38124	2.38122	2.38121	2.38121
Accuracy	+10.4	+8.0	+5.8	+4.7	+5.0
Time (ms)	8845	8845	8845	8845	8844

The authors would like to thank anonymous reviewers for their insightful comments. This work is supported by the National Basic Research Program of China (973 Project Grant No. 2015CB352400). This work is also partly supported by Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT) China, NSFC (Grant No. 61272099) and Scientific Innovation Act of STCSM (No. 13511504200).

7. REFERENCES

[1] R. Arora, R. Tulshyan, and K. Deb. Parallelization of binary and real-coded genetic algorithms on gpu using cuda. In *CEC '10*, pages 1–8. IEEE, 2010.

[2] A. Beham, S. Winkler, S. Wagner, and M. Affenzeller. A genetic programming approach to solve scheduling problems with parallel simulation. In *IPDPS '08*, pages 1–5. IEEE, 2008.

[3] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[4] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

[5] B. Dorronsoro and P. Bouvry. Cellular genetic algorithms without additional parameters. *The Journal of Supercomputing*, 63(3):816–835, 2013.

[6] Z. Konfrt. Parallel genetic algorithms: Advances, computing trends, applications and perspectives. In *IPDPS '04*, page 162b. IEEE, 2004.

[7] M. Lahiri and M. Cebrian. The genetic algorithm as a general diffusion model for social networks. In *AAAI '10*, pages 494–499, 2010.

[8] T. V. Luong, N. Melab, and E.-G. Talbi. Gpu-based island model for evolutionary algorithms. In *GECCO '10*, pages 1089–1096. ACM, 2010.

[9] A. Markham and N. Trigoni. Discrete gene regulatory networks dgrns: A novel approach to configuring sensor networks. In *INFOCOM '10*, pages 1–9. IEEE, 2010.

[10] Z. Michalewicz. Genetic algorithms, numerical optimization, and constraints. In *ICGA '95*, pages 151–158. Morgan Kaufmann, 1995.

[11] G. Renner and A. Ekart. Genetic algorithms in computer aided design. *Computer-Aided Design*, 35(8):709–726, 2003.

[12] P. Vidal and E. Alba. A multi-gpu implementation of a cellular genetic algorithm. In *CEC '10*, pages 1–7. IEEE, 2010.

[13] L. Zheng, Y. Lu, M. Guo, S. Guo, and C.-Z. Xu. Architecture-based design and optimization of genetic algorithms on multi- and many-core systems. *Future Generation Computer Systems*, 38(0):75–91, 2014.

APPENDIX

A. DESCRIPTIONS OF TEST PROBLEMS

In the following, we provide detailed descriptions of the benchmarks used in our evaluation. Those three Nonlinear Programming problems (NLP) are widely used to evaluate the performance of different GA schemes or other optimization algorithms [10, 4, 13]. Specifically, we show the mathematical definition, the optimal solution and the best solution obtained by GA-based methods in the literature for each test problem.

A.1 Test Problem 1

This problem has eight variables and six inequality constraints.

Minimize

$$f(\vec{x}) = x_1 + x_2 + x_3$$

Subject to

$$\begin{aligned}
g_1(\vec{x}) &\equiv 1 - 0.0025(x_4 + x_6) \geq 0, \\
g_2(\vec{x}) &\equiv 1 - 0.0025(x_5 + x_7 - x_4) \geq 0, \\
g_3(\vec{x}) &\equiv 1 - 0.01(x_8 - x_5) \geq 0, \\
g_4(\vec{x}) &\equiv x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0, \\
g_5(\vec{x}) &\equiv x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0, \\
g_6(\vec{x}) &\equiv x_3 x_8 - x_3 x_5 + 2500 x_5 - 1250000 \geq 0, \\
100 &\leq x_1 \leq 10000, \\
1000 &\leq (x_2, x_3) \leq 10000, \\
10 &\leq x_i \leq 1000, i = 4, \cdots, 8.
\end{aligned}$$

The optimum solution is $f^*(\vec{x}) = 7049.330923$.

A.2 Test Problem 2

This problem has four variables and five inequality constraints, and is known as the welded beam design problem (WBD).

Minimize

$$f(\vec{x}) = 1.10471h^2l + 0.04811tb(14.0 + l)$$

Subject to

$$\begin{aligned}
g_1(\vec{x}) &\equiv 13600 - \tau(\vec{x}) \geq 0, \\
g_2(\vec{x}) &\equiv 30000 - \sigma(\vec{x}) \geq 0, \\
g_3(\vec{x}) &\equiv b - h \geq 0, \\
g_4(\vec{x}) &\equiv P_c(\vec{x}) - 6000 \geq 0, \\
g_5(\vec{x}) &\equiv 0.25 - \delta(\vec{x}) \geq 0, \\
0.125 \leq h \leq 10, \\
0.1 \leq l, t, b \leq 10,
\end{aligned}$$

The terms $\tau(\vec{x}), \sigma(\vec{x}), P_c(\vec{x}), \delta(\vec{x})$ are given below.

$$\begin{aligned}
\tau(\vec{x}) &= \Big(((\tau'(\vec{x}))^2 + (\tau''(\vec{x}))^2 \\
&\quad + \frac{l\tau'(\vec{x})\tau''(\vec{x})}{\sqrt{0.25(l^2+(h+t)^2)}}\Big)^{\frac{1}{2}}, \\
\sigma(\vec{x}) &= \frac{504000}{t^2 b}, \\
P_c(\vec{x}) &= 64746.022(1 - 0.0282346t)tb^3, \\
\delta(\vec{x}) &= \frac{2.1952}{t^3 b},
\end{aligned}$$

where

$$\begin{aligned}
\tau'(\vec{x}) &= \frac{6000}{\sqrt{2}hl}, \\
\tau''(\vec{x}) &= \frac{6000(14+0.5l)\sqrt{0.25(l^2+(h+l)^2)}}{2\{0.707hl(l^2/12+0.25(h+t)^2)\}}.
\end{aligned}$$

The optimum solution is $f^*(\vec{x}) = 2.38116$.

A.3 Test Problem 3

This problem has ten variables and eight inequality constraints.

Minimize

$$\begin{aligned}
f(\vec{x}) &= x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 \\
&\quad + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\
&\quad + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 \\
&\quad + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45
\end{aligned}$$

Subject to

$$\begin{aligned}
g_1(\vec{x}) &\equiv 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0, \\
g_2(\vec{x}) &\equiv -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0, \\
g_3(\vec{x}) &\equiv 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0, \\
g_4(\vec{x}) &\equiv -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0, \\
g_5(\vec{x}) &\equiv -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0, \\
g_6(\vec{x}) &\equiv -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0, \\
g_7(\vec{x}) &\equiv -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0, \\
g_8(\vec{x}) &\equiv 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0, \\
-10 \leq x_i \leq 10, i = 1, \cdots, 10.
\end{aligned}$$

The optimum solution is $f^*(\vec{x}) = 24.3062091$.