Future Generation Computer Systems 38 (2014) 75–91

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Architecture-based design and optimization of genetic algorithms on multi- and many-core systems

Long Zheng[a,b], Yanchao Lu[a], Minyi Guo[a,*], Song Guo[b], Cheng-Zhong Xu[c]

[a] University of Shanghai Jiao Tong University, Shanghai, 200240, China
[b] University of Aizu, Aizu-Wakamatsu, 965-8580, Japan
[c] Wayne State University, Detroit, MI 48202, United States

HIGHLIGHTS

- Evaluate different PGA schemes and propose the best one for each architecture.
- General optimization approaches and rules are analyzed and proposed.
- Practical comparisons of GA performance on multi-core and many-core are discussed.
- Besides execution speed, solution quality is also concentrated on and analyzed.
- All work is based on features of architectures, instead of specific GA problems.

ARTICLE INFO

Article history:
Received 27 April 2012
Received in revised form
5 April 2013
Accepted 14 September 2013
Available online 10 October 2013

Keywords:
Genetic algorithm
Multi-core
GPU
Accuracy
Architecture
Speedup

ABSTRACT

A Genetic Algorithm (GA) is a heuristic to find exact or approximate solutions to optimization and search problems within an acceptable time. We discuss GAs from an architectural perspective, offering a general analysis of performance of GAs on multi-core CPUs and on many-core GPUs. Based on the widely used Parallel GA (PGA) schemes, we propose the best one for each architecture. More specifically, the Asynchronous Island scheme, Island/Master–Slave Hierarchy PGA and Island/Cellular Hierarchy PGA are the best for multi-core, multi-socket multi-core and many-core architectures, respectively. Optimization approaches and rules based on a deep understanding of multi- and many-core architectures are also analyzed and proposed. Finally, the comparison of GA performance on multi-core and many-core architectures are discussed. Three real GA problems are used as benchmarks to evaluate our analysis and findings.

There are three extra contributions compared to previous work. Firstly, our findings based on deeply analyzing architectures can be applied to all GA problems, even for other parallel computing, not for a particular GA problem. Secondly, the performance of GAs in our work not only concerns execution speed, also the solution quality has not been considered seriously enough. Thirdly, we propose the theoretical performance and optimization models of PGA on multi-core and many-core architectures, finding a more practical result of the performance comparison of the GA on these architectures, so that the speedup presented in this work is more reasonable and is a better guide to practical decisions.

1. Introduction

Nowadays, multi-core processors and many-core GPUs have entered the mainstream of microprocessor development. The multi-core and many-core architecture both successfully make use of Thread Level Parallelism (TLP) to improve the performance rather than just Instruction Level Parallelism (ILP).

When microprocessors use ILP to improve the performance, all parallel mechanisms are hidden by compilers and the architecture of microprocessors. However with TLP, parallel mechanisms cannot be hidden. Users have to know about the architecture of multi-core and many-core systems, and implement their parallel programs explicitly. For example, users need to write multi-threaded code to improve the parallelism of their programs. For GPUs, users even have to be familiar with the details of the architecture, because they must assign threads to different SMs, use hundreds of cores efficiently, and consider the choice of different types of memory. With multi-core systems, several existing or new programming models and environments can help users. For example, Pthread, OpenMP, Cilk [1], and even MapReduce [2] can be considered tools to help users implement programs on multi-core

* Corresponding author.
E-mail addresses: d8112104@u-aizu.ac.jp (L. Zheng), chzblych@sjtu.edu.cn (Y. Lu), guo-my@cs.sjtu.edu.cn (M. Guo), sguo@u-aizu.ac.jp (S. Guo), czxu@wayne.edu (C.-Z. Xu).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.09.029

systems. GPU-based systems are newer than multi-core systems, but work has been done in both industry and academia. Users can use CUDA, OpenCL to interact with hundreds of cores. Recently, a framework called MARS [3] is proposed, so that MapReduce can also help to harness the power of many-core GPUs. These systems do not try to hide the parallelism; they expose it to users, which requires users to know about architectural details and parallel mechanisms; otherwise, the systems cannot perform efficiently.

A Genetic Algorithm (GA) is a heuristic to find exact or approximate solutions to optimization and search problems within an acceptable time; the technique is widely used in business, engineering and science [4–7]. Actually, many real world engineering problems that solved by GA require quite a long computation time. Furthermore, GA is integrated in many other applications, and stands as a key component (such as scheduling) which has a significant influence on the overall performance. These facts are the motivation that multi- and many-core architectures are adopted, focused and researched [8–11].

Although some transplanting of GAs from CPUs to GPUs has been done, the lack of understanding of detailed multi-core and many-core architectures leads to the following shortcomings in the previous work:

1. Many GA which work on multi-core and many-core systems are done on a case by case basis, describing how to use multi-core and many-core systems to accelerate the specific GA problems. Some work even misunderstands the fundamentals of implementation on the new architecture. The different GA problems have more commonalities than differences. Features of multi- and many-core architectures actually determine which Parallel GA (PGA) is the best for both execution speed and solution quality, rather than features of different GA problems.
2. Most work focuses exclusively on the relationship between the speedup on the multi-core and many-core architecture, pursuing a fast execution time, but ignoring the solution quality. Since GAs mostly find only approximate solutions, more attention should be paid to solution quality. The relationship between speedup and architecture should be discussed along with the solution quality.
3. Most previous work demonstrates the GPU performance by comparing the execution speed on GPUs to a serial implementation on CPUs. Although this metric is reasonable, the serial implementation can only use a single CPU core. Now multi-core CPU is the mainstream. The lack of expressing the speedup of GPUs compared to multi-core CPUs makes the speedup in previous work less practical to GA users.

These problems motivate us to discuss GAs from an architecture perspective, to offer a general analysis of GAs on multi-core and many-core architectures, considering the quality of solutions. We first introduce the fundamentals of GAs followed by several popular PGA schemes; then we make an architecture-based analysis of the different PGA models on multi-core and many-core architectures. The best PGA schemes for multi-core, multi-socket multi-core and many-core architectures are proposed as well. Also, we propose some approaches and rules to optimize the performance of GAs on multi- and many-core architectures. Finally, three real and widely used GA problems are used as benchmarks, so that our findings can be validated.

Especially, our work concentrates on features of architectures which are considered as the key points to improve GA performance. Hence our work aims at all kinds of GAs not a specific GA problem. We consider the GAs from the abstract level of a parallel model; performance improvements in GA fundamentals – for example, migration topology, mutation strategy or selection method – will not affect our findings. These fundamental GA improvements can get more performance benefit from the architecture with our findings. Existing multi- and many-core acceleration work for GAs mostly aims to reduce the execution time, ignoring the solution quality. We take the solution quality into account, so that we try to make GAs get the best solutions in the shortest time, since a solution is what the engineers and researchers who use GAs actually need. As we propose the best approaches to make GA use multi-core and many-core architectures, we can make a fairer comparison of GA performance between multi-core and many-core. Besides the speedup of execution time, a speedup with consideration of solution quality is also proposed. The speedup based on our experimental results is more reasonable and is a better guide to practical decisions when engineers and researchers use GAs to solve their problems.

We highlight our contributions compared to previous work as follows.

1. Analyze the performance of generic GAs on both multi-core and many-core architectures in depth, and proposing the best PGA scheme and implementation on each architecture.
2. Consider a more reasonable performance metric that combines the execution speed and the solution quality to compare the performance of different PGA schemes on different architectures.
3. Propose the theoretical performance and optimization models of PGA on multi-core and many-core architectures and evaluate them with three real world engineering problems, finding a more practical and bias result of the performance comparison of the GA on these architectures.

The remainder of this paper is structured as follows. Section 2 gives an essential overview of GA. Sections 3 and 4 offer an architecture-based theoretical analysis of GAs and propose the best approaches and optimization rules for multi-core and many-core systems. We evaluate our analysis and findings in Section 5. Related work is discussed in Section 6. Section 7 summarizes our findings.

2. Background of GA

Before analyzing GA on multi-core and many-core architectures, we give a quick overview of GAs. In this section, we begin with a review of species selection and evolution in nature, which is a good way to understand the fundamentals of GAs. Based on this, we present several GA schemes for parallel and distributed computing environments.

2.1. Fundamental of GA

In nature, individuals compete with each other and adapt to the environment. Only the strongest ones can survive in a tough environment. The survivors mate more-or-less randomly and produce the next generation. During reproduction, mutation always occurs, which makes some individuals of the next generation better fitted for the environment.

GAs are heuristic search algorithms that mimic natural species selection and evolution as described above. The problem that a GA intends to solve is the tough environment. Each individual in the population of a GA is a candidate solution for 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* chooses 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. This is the classic serial GA scheme [12]. Since GAs are so similar to the