Fault-tolerance of Distributed Genetic Algorithms on Many-core Processors

Yuji Sato
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo 184-8584 JAPAN
+81-42-387-4533
yuji@k.hosei.ac.jp

Mikiko Sato
Tokyo University of Agriculture and Technology
2-24-16 Naka-cho, Koganei-shi
Tokyo 184-8588 JAPAN
+81-42-388-7139
mikiko@namikilab.tuat.ac.jp

ABSTRACT
This paper reports on fault-tolerant technology for use with high-speed parallel evolutionary computation on many-core processors. In particular, for distributed GA models which communicate between islands, we propose a method where an island's ID number is added to the header of data transferred by this island for use in fault detection, and we evaluate this method using Deceptive functions and Sudoku puzzles. As a result, we show that it is possible to detect single stuck-at faults with practically negligible overheads in applications where the time spent performing genetic operations is large compared with the data transfer speed between islands. We also show that it is still possible to obtain an optimal solution when a single stuck-at fault is assumed to have occurred, and that increasing the number of parallel threads has the effect of making the system less susceptible to faults and more sustainable.

Categories and Subject Descriptors
D.0 [**Computer Applications**]: General

General Terms
Reliability, Performance, Design, Experimentation, Verification.

Keywords
Genetic Algorithms, Fault-tolerance, Many-core Processors

1. INTRODUCTION
With the growing popularity of GPU devices and the use of these devices in fields where high reliability is required, such as scientific computing and data centers, NVIDIA has developed the Tesla series of GPUs with ECC memory (error checking and correction memory) functions.

In this ECC memory, additional information such as a parity code must be added to each data word. This means that the installed memory must be allocated for the storage of parity codes. Although there is no problem allocating the parity codes to part of a high-capacity memory region such as the global memory (GM), this approach is unsuitable for the allocation of smaller memory regions such as registers or shared memory inside a streaming multiprocessor (SM). Next, the ECC memory can correct single-bit errors, and can also detect errors of two or more bits but without the ability to identify the locations of these errors. Therefore, ECC memory is effective for transient faults, but not always sufficient for stuck-at faults (Individual signals are assumed to be stuck at Logical '1' or '0'). Furthermore, modern supercomputers are often configured by connecting multiple GPUs in large-scale networks due to the advantages of such architectures, including their very high cost/performance ratios and low power consumption. Thus, outside of the GMs, it is also essential to have some way of detecting faults in the interconnections between the SMs and GMs, or in the network between the GPUs.

Against this background, we have already showed that when parallel evolutionary computation is performed in many-core processors using a scheme based on independent competition, it seems that benefits such as higher reliability and lower susceptibility to transient errors can be achieved [1]. In this paper, we propose an island-model fault-tolerant technology that takes communication into consideration.

2. AN ISLAND-MODEL FAULT-TOLERANT TECHNIQUE
In a GPU, the transfer of data between SMs is generally specified as being performed in byte units. On the other hand, the data to be exchanged does not normally correspond to an exact number of byte units, so the GPU system pads out the data with random bits to form a whole number of bytes for transfer between the SMs. Our basic idea is to perform error detection by making using these left over bits when data is transferred between SMs.

Figure 1 shows an example of how this idea can be implemented. In an island model, it is usually determined in advance which island will transfer data to which island. Here is an example of an implementation where a single island corresponds to a single SM. In this example, data is transferred between SMs with an added fault detection header that uses the bits that are left over when the data is converted into byte units, and each SM stores a cyclic list showing the predetermined order in which data is sent to it from the other SMs. Faults can then be detected by checking whether or not the header information in the received data matches the information recorded in the cyclic list. When a fault is detected, the genetic operations are continued without accepting the received data.

The target of fault detection is assumed to be single stuck-at faults in the SMs or on the network. When there is a fault in the SM on the transmitting side, errors occur in the transferred header data so that it fails to match the information in the cyclic list registered in the receiving SM. As a result, the fault can be detected and the number of the SM where this fault occurred can be reported. When there is a fault in the network, this can be detected when a fault has occurred at the header insertion location. It is thus possible to detect a single fault at any location on the network by shifting the header insertion position each time data is transferred.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

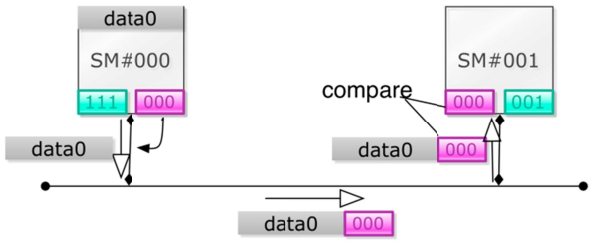

Fig. 1. Conceptual illustration of the fault-tolerant technique for island models.

3. EXPERIMENT

3.1 Evaluation using Sudoku puzzles

We investigated the overhead for fault detection using an AMD Opteron Quad Core 2380 multi-core processor. Experiments were performed in a system configured with 8 core processors by connecting two Opteron 2380 processors together. Figures 2 shows the relationship between the number of generations between data transfers, the number of generations needed to reach an optimal solution, and the processing performance per generation for solving Sudoku puzzles [2]. The bar graph shows the number of generations until convergence, and the values are shown on the left side. The line graph shows the processing performance, with the number of operations per second shown on the right side. This Figure shows the results obtained with 4 threads. The two bar graphs for each transfer interval represent ordinary migration from left to right, and migration where fault detection headers are inserted. Both sets of results were obtained in experiments with the same initial population. As this Figure shows, the overheads incurred by implementing the proposed idea are negligibly small, regardless of the data transfer interval.

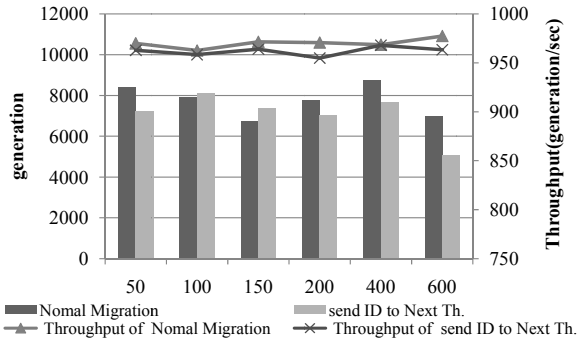

Fig. 2. Data transfer interval (number of generations) vs. number of generations needed to reach an optimal solution and processing performance per generation (number of threads: 4)

Figures 3 shows the relationship between the number of threads and the number of generations needed to obtain an optimal solution when assuming a single stuck-at fault. The three bar graphs for each thread correspond to (from left to right) ordinary migration, migration with fault detection headers inserted, and migration with fault detection headers and a single stuck-at fault. Fig. 3 shows the results of a method where sequential data is transferred to a neighboring thread ("Next" method). In this method, the results obtained with a data transfer interval of 200 generations are shown. In this Figure, when there are two threads and we assume a fault in one thread, then the number of generations needed for convergence increases but it is still possible to obtain an optimal solution. With four or eight threads, the assumption of a fault in one thread has hardly any effect on the ability to find a solution.

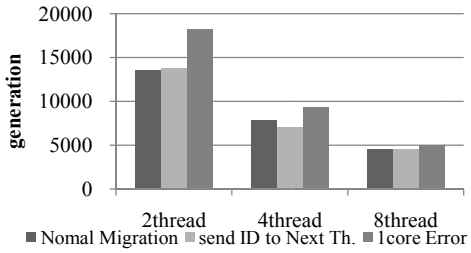

Fig. 3. Relationship between number of threads and number of generations needed to find a solution when assuming a single stuck-at fault ("Next" method, transfer interval: 200 generations).

3.2 Evaluation using Deceptive functions

We investigated the overhead for fault detection using a Nvidia's Geforce GTX 680 GPU. Experiments were performed in a system configured with 8 SMs. In this experiment, the original data length is 64-bits (8-bytes) and there is no space for header insertion, an extra byte has to be added. Table I shows the overheads obtained when an extra bits has to be added for header insertion. Whole sets of results were obtained in experiments with the same initial population. As this Table shows, the overheads incurred by implementing the proposed idea are about 13%, for any bit length of fault detection headers, but it is still possible to obtain an optimal solution.

TABLE I. HEADER LENGTH AND ECESUTION TIME

Header length	0-bits	3-bits	8-bits
Execution time [sec]	0.46	0.52	0.53

4. CONCLUSIONS

In this paper, for an island model where communication is performed between islands, we proposed a method where the ID number of an island is added to the header of data transferred by the island for use in fault detection, and we have shown that the processing time of the proposed idea is practically negligible. We have also shown that an optimal solution can be obtained even with a single stuck-at fault, and that increasing the number of parallel threads makes the system less susceptible to faults.

5. REFERENCES

[1] Sato, Y.: Parallelization of Genetic Algorithms and Sustainability on Many-core Processors. In Advances in Intelligent Systems and Computing, 202, pp. 175-187, Springer (2012).

[2] Super difficult Sudoku-1. Available via WWW: http://lipas.uwasa.fi/~timan/sudoku/EA_ht_2008.pdf#search='CT20A6300%20AIterative%20Project20work%202008' (cited 10. 2. 2013).