On the Automatic Generation of Efficient Parallel Iterative Sorting Algorithms

Gopinath Chennupati
BDS Group
CSIS Department
University of Limerick, Ireland
gopinath.chennupati@ul.ie

R. Muhammad Atif Azad
BDS Group
CSIS Department
University of Limerick, Ireland
atif.azad@ul.ie

Conor Ryan
BDS Group
CSIS Department
University of Limerick, Ireland
conor.ryan@ul.ie

ABSTRACT

Increasing availability of multiple processing elements on the recent desktop and personal computers poses unavoidable challenges in realizing their processing power. The challenges include programming these high processing elements. Parallel programming is an apt solution for such a realization of the computational capacity. However, it has many difficulties in developing the parallel programs.

We present *Multi-core Grammatical Evolution for Parallel Sorting* (MCGE-PS) that automatically produces *native* parallel sorting programs. These programs are of iterative nature that also exploit the processing power of the multi-core processors efficiently. The performance of the resultant programs is measured in terms of the execution time. The results indicate a significant improvement over the state-of-the-art implementations. Finally, we conduct an empirical analysis on computational complexity of the evolving parallel programs. The results are competitive with that of the state-of-the-art evolutionary attempts.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search - Heuristic methods

Keywords

Grammatical Evolution; Multi-cores; Program Synthesis; Performance Optimization; OpenMP; Sorting.

1. INTRODUCTION

With an increase in the number of cores on a single chip, programming them becomes hard for an average programmer. That being the fact, with the *death of scaling*[1], they need to be explicitly programmed in exploiting their true potential. Programming them requires human expertise in producing efficiently executing programs.

[1]http://www.gotw.ca/publications/concurrency-ddj.htm

GECCO '15 Companion, July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739482.2764695

While parallel programming can be challenging for a human programmer considering the associated objectives such as optimising the degree of parallelism and porting code across different parallel architectures, automatically generating parallel programs is even tougher especially when producing a program from scratch. However, given the prevalent parallel hardware, automatic parallel programming can significantly advance the state of the art towards achieving the dream that is automatic programming of computers.

Evolutionary Computation offers an easy solution for this problem, still, generating parallel programs is hard with the existing domain knowledge. Attempts for automatic parallelization were initiated with *Paragen-I* [7, Chapter-5], which mapped the serial programs onto multiprocessors. Then, Ryan and Ivan [8] merged independent tasks of different loops into a single loop.

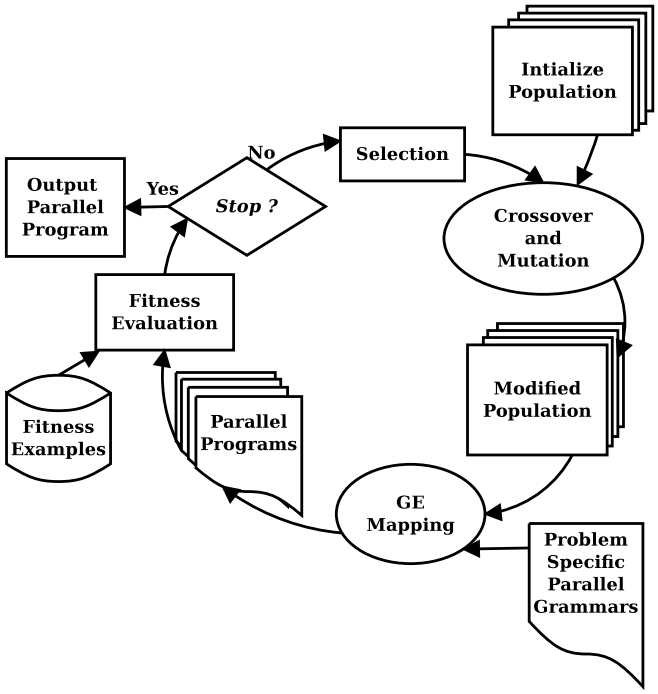

Figure 1: An overview of the Multi-Core Grammatical Evolution for the automatic evolution of parallel programs. The evolutionary cycle of the approach follows that of GE, except for the changes in the BNF grammar and fitness evaluation.

Recently, [1] generated parallel regression programs using Grammatical Evolution (GE) [5] and OpenMP [6], automatically. Then, [3] further scrutinized the efficiency of the evolved regression programs. To that end, this paper presents the evolution of parallel iterative sorting by enhancing the state-of-the-art implementations.

2. PROPOSED APPROACH

We enhance *Multi-Core Grammatical Evolution for Parallel Sorting* (MCGE-PS) [2] that automatically generates efficient parallel iterative sorting programs. The proposed approach follows a similar *single program multiple data* (SPMD) parallelization strategy as that of MCGE-PS. Figure 1 shows an overview for the evolutionary cycle of the proposed approach in generating the optimal parallel sorting programs. However, it differs significantly in the design of the grammar as well as the fitness evaluation. The grammars offer a clear separation between the task and data parallelism, while the fitness evaluation considers the execution time.

3. EXPERIMENTS

The proposed enhancements are evaluated on four standard benchmark iterative sorting programs (bubble sort, quick sort, odd-even sort, rank sort). The experiments use the default parameter setting of GE. Mostly, literature concentrated mainly on the evolution of sequential sorts [4] of quadratic complexity with the use of *swap* primitive.

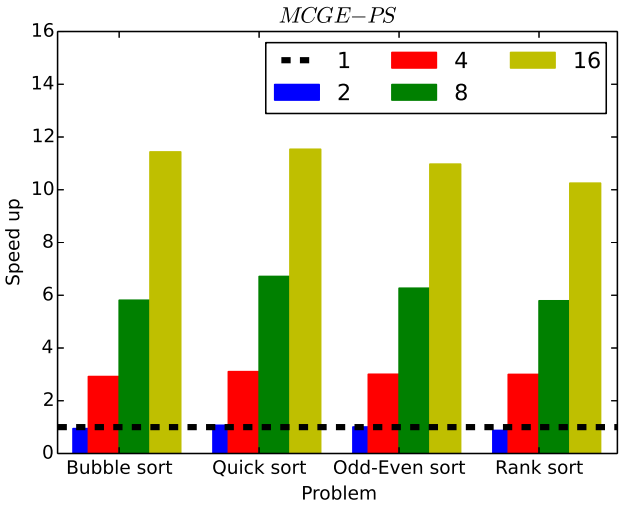

Figure 2: Scale up (in terms of execution time) of MCGE-PS generated parallel sorting programs for all the four experimental problems. The results are for different cores (2, 4, 8, and 16) with reference to uni-core (horizontal dashed line) results.

We measure the performance of the MCGE-PS resulting parallel iterative sorts as speed up in terms of their execution time over the specified number of cores. The programs are executed on 2, 4, 8 and 16 cores of an Intel processor. Figure 2 shows the performance of the four experimental problems on different cores that showed better results.

These programs are also compared with the sequential programs that are generated with GE. The results indicate a significant performance over the sequential programs. However, the results are insignificant for 2 cores due to the thread scheduling and code growth issues discussed in [3]. On an average, the proposed enhancements reports a speed up of 11.56 on 16 cores. Finally, we observe the computational complexity of the best resultant programs. Their complexity results are competitive with that of the state of the art implementations with a reported complexity of $O(n^2)$ for *bubble sort*, $O(nlog(n))$ for *quick sort*.

4. CONCLUSION

We presented an enhanced implementation of MCGE-PS that offers greater flexibility with the changes to the design of the grammars as well as the fitness evaluation. The enhancements resulted in the automatic evolution of efficient parallel iterative sorts with significantly better scale up results over the state-of-the-art implementations.

The improvements in the performance are then proved with the help of an empirical analysis on the computational complexity. However, future work seems promising, expanding its applications into many directions. Foremost of which, we apply our approach to problems such as Lock-less programming, where shared resources are manipulated by multiple threads without the need for locks.

5. REFERENCES

[1] G. Chennupati, R. M. A. Azad, and C. Ryan. Multi-core GE: automatic evolution of CPU based multi-core parallel programs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1041–1044. ACM, 2014.

[2] G. Chennupati, R. M. A. Azad, and C. Ryan. Automatic evolution of parallel sorting programs on multi-cores. In A. M. Mora and G. Squillero, editors, *EvoApplications 2015*, volume 9028 of *LNCS*, pages 706–717. Springer, Heidelberg, 2015.

[3] G. Chennupati, J. Fitzgerald, and C. Ryan. On the efficiency of multi-core grammatical evolution (mcge) evolving multi-core parallel programs. In *Proceedings of the Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 238–243. IEEE, 2014.

[4] K. E. Kinnear Jr. Evolving a sort: Lessons in genetic programming. In *IEEE International Conference on Neural Networks*, pages 881–888. IEEE, 1993.

[5] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[6] OpenMP Architecture Review Board. OpenMP application program interface version 3.0. http://www.openmp.org/mp-documents/spec30.pdf, May 2008.

[7] C. Ryan. *Automatic Re-engineering of Software Using Genetic Programming*, volume 2 of *Genetic Programming*. Springer, 1999.

[8] C. Ryan and L. Ivan. Automatic parallelization of arbitrary programs. In R. Poli et. al., editor, *EuroGP 1999*, volume 1598 of *LNCS*, pages 244–254. Springer, Heidelberg, 1999.