Simulation of a Model for Refactoring Approach for Parallelism Using Parallel Computing Tool Box

Shanthi Makka and B.B. Sagar

Abstract Refactoring is the process of retaining the behavior of a program by making changes to the structure of a program. Initially refactoring is used only for sequential programs, but due to highly configurated architectural availability, it also aids parallel programmers in implementing their parallel applications. Refactoring provides many advantages to parallel programmers, in identifying independent modules, in refining process of programs, it also helps in separating concerns between application and system programmers, and it reduces the time for deployment. All mentioned advantages benefit the programmer in writing parallel programs. The approach for refactoring using multi core system is already developed. Hence all these advantages made us to thought of a system to develop refactoring approach for parallelism which uses heterogeneous parallel architectures which uses combination of both Graphic Processing Unit (GPU) and Central Processing Unit (CPU). A Tool in MATLAB, Parallel Computing Toolbox can be used to execute programs on multiple processing elements simultaneously with local workers available in the toolbox, which takes benefit of GPUs. This tool box uses complete processing speed of multi core system to execute applications on local workers without changing the code. Our suggested model can be simulated by using Parallel Computing Toolbox.

Keywords Refactoring · MATLAB · Parallel computing toolbox · GPU · CPU · Heterogeneous parallel architecture

S. Makka (✉) · B.B. Sagar
BITs, Mesra (Noida Campus), Sector-15, Noida, India
e-mail: shanthi_makka@yahoo.com; shanthi.makka@jre.edu.in

B.B. Sagar
e-mail: drbbsagar@gmail.com

S. Makka
JRE Group of Institutes, Plot no-5,6,7,8, Greater Noida 201308, Uttar Pradesh, India

S.C. Satapathy and S. Das (eds.), *Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 2*, Smart Innovation, Systems and Technologies 51, DOI 10.1007/978-3-319-30927-9_8

1 Introduction

According to Moore's law [1], for every two years the number of transistors on Integrated Circuit is getting double, i.e., the speed of processor increases double. For decades, programmers relied on Moore's Law [2] to improve the performance of their applications. With the advent of multi cores, programmers are forced to exploit parallelism if they want to improve the performance of their applications, and if they want to enable new applications and services that were not possible earlier i.e., to have enhanced user experience and better quality of service. Haskell [3], intel's multi core architecture has eight cores by default. In future computer hardware is likely to have even more cores, with many cores. This makes programmers to think parallel, i.e., we should move away from conventional (sequential) programming approach to parallel approach.

There are two different approaches for parallelizing a programs, one is to rewrite it from scratch, it is very expensive and time consuming and second is parallelize a program incrementally, one piece at a time. Each small segment can be seen as a behavior preserving transformation, i.e., a refactoring. Mostly Programmers prefers second approach because it is safer and economical because it provides workable and deployable version of the program. MATLAB is proprietary programming language [4] for implementing mathematical computations and it is also used for development of algorithms, simulation of models, data reduction, testing and evaluation process and it also provides an excellent platform to create an accessible parallel computing environment. A Parallel Computing Toolbox [5], a tool in MATLAB can be used to run applications on a multi core system where the local workers available in the toolbox, can run applications concurrently by taking the benefit of GPUs.

2 Refactoring

The term refactoring was introduced by William Opdyke in his PhD Thesis [6]. Refactoring is process of changing lines of program without changing its originality or in other words changing of internal structure without altering its external behavior to improve understandability and maintainability of code. Refactoring improves [7] its internal structure.

The different activities are to be performed during the refactoring process are:

a. Identify the segments in a program where the refactoring can be applied.
b. Determine what type of refactoring i.e., extract method, renaming etc. Should be applied to the identified segments.
c. Make sure that the applied refactoring preserves application behavior.
d. Apply the refactoring.
e. Assess the effect of refactoring on quality characteristics of the software.
f. Maintain the consistency between the refactored program and other software artifacts.

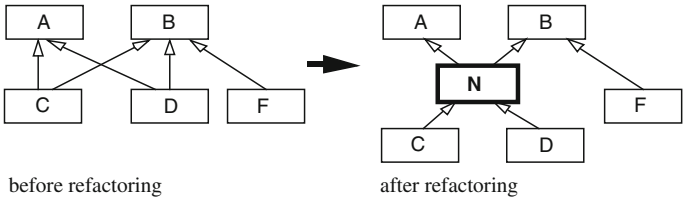

Fig. 1 Example of a refactoring. before refactoring. after refactoring

A refactoring approach for parallelism is already developed for multi core system. Then we thought of to extend our vision to develop a new refactoring approach for parallelism using heterogeneous parallel architectures. Key issues include dealing with advanced *heterogeneous* parallel architectures, involving combinations of GPUs and CPUs; providing good hygienic abstractions that cleanly separate components written in a variety of programming languages; identifying new high level *patterns* of parallelism; developing new rule based mechanisms for rewriting (refactoring) source-level programs based on those patterns etc.

Consider a below example for refactoring "the class A is derived from C and D and class B is derived from C, D, and F. That means the class A can use features of C and D and class B can use features of C, D, and F. Instead of making copy of features of C and D twice once for A and second time for B, We can make new class N which has feature of C and D and that can be used when class A and B requires those features" (Fig. 1).

The refactoring tools [8, 9] or automatic refactoring compilers can improve programmer productivity, performance of applications, and program portability and currently the toolset supports various refactoring techniques for following:

a. To increase throughput, a sequential program can be divided into threads,
b. To make programs thread safe, and
c. To improve scalability of applications.

3 Parallel Computing Toolbox

Parallel Computing Toolbox [5] can solve computationally intensive as well as and data intensive problems on multi core processor environment, GPUs, and computer clusters. This toolbox enables us to parallelize applications without Compute Unified Device Architecture (CUDA) and MPI programming environment. Simulink is a simulator which simulates suggested model which runs multiple parts of the model parallel.

Key Features

a. Parallel Computing Toolbox provides GPU Array, which is associated with certain functions through which we can perform computations on CUDA.

b. Use of multi core processors on a system through workers that run locally.
c. Interactive and batch execution of parallel applications.
d. Data parallel applications implementation can also be done.
e. You can increase the execution speed of applications by dividing application into independent tasks and executing multiple tasks concurrently.
f. Parallel for loops (parfor) are used for run task parallel applications on multi core or processor system.

4 Refactoring for Parallelism Using GPUs and CPU

(See Fig. 2).

Algorithm

1. Program Dependence Graph can be used to represent an application.
2. Independent modules can be identified using refactoring technique.
3. Those modules can be directed (using appropriate mapping technique) to heterogeneous pool during running time of applications.

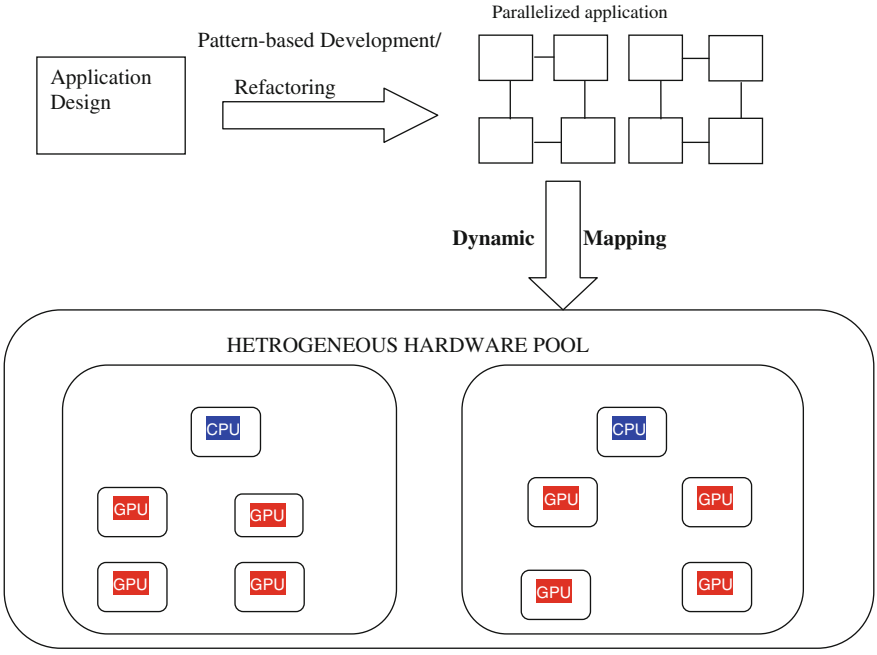

Fig. 2 refactoring approach for parallelism using combination of GPU and CPU

4.1 Why Refactoring

Refactoring [10, 11] is the process of preserving program behavior by changing its structure which makes program more readable, understandable and maintainable. After a refactoring, program should not give any syntactic errors. Refactoring never change the behavior of a program; i.e., if any program called twice before and after refactoring the results must be same for same set of inputs.

While refactoring do not change the external behavior of a program, it can support for designing of software in very effective way and also in evolution process by restructuring a program in the way that allows other modifications to be made more easily and rapidly. Complicated changes to a program can require both refactoring and additions. Refactoring is behavior preserving so that, whenever the preconditions are met it do not break the program.

A refactoring approach has many advantages in parallel programming: it helps the programmer in the process of identification of independent modules, it also guides the programmer in the refining process of a parallel program and it also reduces time for deployment.

Applications of refactoring:

1. Refactoring [12] aims to improve software design.
2. It encourages good program design.
3. It makes software easier to understand.
4. It improves readability of the code.
5. A good program design is not only guarantees rapid rate, but also accurate software development.

4.2 Why the Combination of Both GPUs and CPU

1. GPU are designed for highly parallel architectures [13] which allows large blocks of data to be processed.
2. GPUs does similar computations are being made on data at the same time (rather than in a sequence one after other).
3. The GPU has emerged as a computational accelerator [14] that dramatically reduces the time to discovery in High End Computing (HEC).
4. GPU can easily reduce the execution time of a parallel code.

CPU is required to coordinate all GPUs which are executing segments concurrently or to manage communication overhead between GPUs, and also to execute lighter segments of program. GPUs are very expensive, so it should be used when there essential requirement.

5 Related Work

The refactoring is different for both software development and optimization. The frame work chosen for refactoring is depending on overall program design and knowledge representation. The Object oriented refactoring was first introduced by Opdyke [6] in his Ph.D. thesis. The transformation of program into modules is being described in [15]. There are certain automatic compilers exist which can make transformations automatically by acting either on source level programs or their intermediate language representations or during parsing of program, this approach has been discussed in [16]. Burstall and Darlington [17] mentioned algorithms for transformations of source program for both sequential and parallel machines. In critical situations, non executable modules may transformed into an executable program. Later work is exemplified by the relational approach of Bird and de Moor [18]. A catalogue for refactoring is given by Fowler in [11] an website and this is also kept up to date at www.refactoring.com, which also has links to tools and other resources. The most widely known tool for refactoring is the Refactoring Browser Smalltalk [19].

6 Simulation of Suggested Model Using Parallel Computing Toolbox

The suggested model can be simulated by using parallel computing toolbox as follows: After identification of independent modules which can be executed simultaneously or parallel through refactoring approach can be assigned to local workers, which makes use of processing speed of GPUs. Scheduling of these modules can be done by Job Scheduler (CPU). All these local workers produce solution to their individual modules and finally merging of these solutions gives the solution for original problem (Fig. 3).

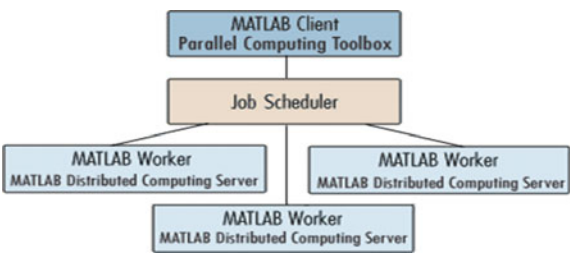

Fig. 3 Parallel computing tool box in MATLAB

7 Conclusion

a. To increase the execution speed of applications, if we use single processor system to run applications at higher clock speeds which consumes lots of power and which also generates large amount of heat. To avoid such circumstances, programmer should think parallel.
b. Refactoring is an approach through which we can identify modules which can be executed simultaneously.
c. To execute applications parallel, we can use multi core system, which has been already developed.
d. Finally we thought of to develop a system which uses refactoring approach to identify modules for simultaneous execution on heterogeneous parallel architectures which uses the combination of both GPUs and CPUs.
e. A parallel computing toolbox can be used to simulate above said Model.

References

1. Moore, G.E.: Readings in computer architecture: Cramming more components onto integrated circuits. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 56–59 (2000)
2. Dig, D.: A refactoring approach to parallelism. Software, IEEE **28**(1), 17–22 (2011)
3. Brown, C., Loidl, H.W., Hammond, K.: Paraforming: forming parallel haskell programs using novel refactoring techniques. Trends in Functional Programming. Springer, Berlin Heidelberg, 82–97 (2012)
4. Kim, H., Mullen, J., Kepner, J.: Introduction to parallel programming and pMatlab v2. 0. Massachusetts Inst Of Tech Lexington Lincoln Lab (2011)
5. http://in.mathworks.com/products/parallel-computing
6. Opdyke, W.F.: Refactoring: a program restructuring aid in designing object-oriented application frameworks. University of Illinois at Urbana Champaign, (1992)
7. Tom, M., Tourwé, T.: A survey of software refactoring. IEEE Trans. on Softw. Eng. 30(2), 126–139 February 2004
8. Liao, SW., Diwan, A., Bosch Jr., R.P., Ghuloum, A., Lam, M.S.: Suif explorer: an interactive and interprocedural parallelizer. In PPoPP'99 7th symposium on Principles and practice of parallel programming ACM SIGPLAN, 37–48 (1999)
9. Dig, D., Marrero, J. Ernst, M.D.: Refactoring sequential Java code for concurrency via concurrent libraries. In 31st International Conference on Software Engineering. IEEE Computer Society, 397–407 (2009)
10. Murphy-Hill, E., Black, A.P.: Breaking the barriers to successful refactoring. 30th International Conference on ACM/IEEE Software Engineering ICSE'08. IEEE (2008)
11. Fowler, M., Refactoring: Improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc (1999)
12. Berthold, H., Pére, J., Mens, T.: A case study for program refactoring. Proc. of the GraBaTS Tool Context (2008)
13. Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., Sarrafzadeh, M.: Energy-aware high performance computing with graphic processing units. In Workshop on Power Aware Computing and System, December 2008

14. Huang, S., Xiao, S., Feng, W.C.: On the energy efficiency of graphics processing units for scientific computing. International Symposium on. Parallel & Distributed Processing IPDPS. IEEE (2009)
15. Partsch, H., Steinbrüggen, R.: Program transformation systems. ACM Computing Surveys, 15 (3). September 1983
16. Jones, S.L.P.: Compiling Haskell by program transformation: a report from the trenches. In European Symposium on Programming (ESOP'96), April 1996
17. Burstall, R.M., Darlington, J.: A transformation system for developing recursive programs. J. ACM **24**(1), 44–67 (1977)
18. Bird, R., De Moor, O.: Algebra of Programming. Prentice-Hall, (1997)
19. Roberts, D., Brant, J., Johnson, R. A.: Refactoring Tool for Smalltalk. Theory and Prac. of Obj. Sys. (TAPOS). Special Issue on Software Reengineering, 3(4),253–263 (1997) see also http://st-www.cs.uiuc.edu/users/brant/Refactory/