APPROACHING DEVELOPMENTS ON PARALLEL PROGRAMMING MODELS THROUGH JAVA

By

BALA DHANDAYUTHAPANI VEERASAMY * G.M. NASIRA **

* Research Scholar, Department of Information Technology, Manonmaniam Sundaranar University, Tamilnadu, India.
** Assistant Professor, Department of Computer Science, Chikkanna Govt Arts College, Tamilnadu, India.

ABSTRACT

Multicore platforms allow developers to optimize applications by intelligent partitioning at different workloads on different processor cores. Currently, application programs are optimized to use multiple processor resources, resulting in faster application performance. The authors earlier research work focused on native thread for Java on windows thread, Pthread, and Intel TBB. The authors also developed Native Threads, Native Pthread, Java Native Intel TBB beneath windows 32-bit platform. This article aims to identify the future directions of native thread for Java on windows thread, Pthread, and Intel TBB through JNI beneath windows 64-bit platforms and other platform besides. Furthermore, it articulates additional opening to pursue approaching developments on parallel programming models through Java.

Keywords: CPU, CUDA, GPGPU, GPU, Java, jCuda, jocl, Multicore, OpenCL.

INTRODUCTION

Parallel computers can be approximately categorized according to the level at which the hardware supports parallelism. Through multicore and multiprocessor computers having many processing elements within a particular machine, clusters, Massively Parallel Processing (MPP) and Grids use multiple computers to work on the similar tasks.

Multicore Computing

A multicore processor [1] comprises multiple execution units (cores) on the same chip. Each core in a multicore processor can possibly be superscalar as well as on each cycle, each core can perform multiple instructions from one instruction stream. Simultaneous Multi Threading (SMT) was an initial form of pseudo multicoreism. A processor capable of SMT has only one core, but when that execution unit is idling, it uses that execution unit to process a second thread.

Symmetric Multiprocessing

A SMP [2] is a computer system with multiple identical processors that share memory and associate via a bus. The bus argument prevents bus architectures from scaling, which do not include more than 32 processors.

Distributed Computing

A distributed computer [3] is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are extremely scalable.

Cluster Computing

Clusters [3] are a collection of multiple stand alone computers connected by a network. Though the computers in a cluster do not have to be symmetric, load balancing is more problematic if they are not. The most collective type of cluster is the Beowulf cluster, is a cluster applied on multiple identical commercial standard computers connected with a TCP/IP Ethernet Local Area Network.

Massive Parallel Processing (MPP)

It is a single computer with many networked processors. MPPs [4] have numerous of the same features as clusters, but MPPs have particular interconnect networks. MPPs also tend to be larger than clusters, normally having more than 100 processors.

Grid Computing

Grid computing [3,4] is the most distributed forms of

parallel computing. It makes use of computers connecting over the Internet to work on a given problem. Grid computing applications use middleware software that connects between the operating system and the application to manage network resources and standardize the software interface.

Field Programmable Gate Arrays (FPGA)

Reconfigurable computing is the use of an FPGA [4] as a co-processor to a general-purpose computer. The FPGA is in essence a computer chip that can rewire itself for a given task.

General Purpose Computing on Graphics Processing Units (GPGPU)

GPUs [1,4] are coprocessors have been deeply enhanced for computer graphics processing. Computer graphics processing is a field dominated by data parallel operations mostly on linear algebra matrix operations. Programming languages and platforms have been made to do general purpose computation on GPUs with both NVIDIA and Advanced Micro Devices (AMD) releasing programming environments with CUDA and Stream SDK respectively. The consortium Khronos Group released the OpenCL specification, it is a framework for writing programs that executes across platforms consisting of CPUs and GPUs, AMD, Apple, Intel, NVIDIA.

1. Research Work Developed

The aimed research work were identified and developed for the parallel programming model implementations through Java. The research work provided a prototype model to produce any other language threading features to the Java language. The research works done are summarized as follows.

- Provided methods to set CPU affinity [5] for Java threads in Win32 platform.
- Developed an application with parallel execution, such as parallel one time pad [6] using Java.
- Developed a JNI program to support Win32 platform thread [7] for Java, evaluating Java threads and Native Thread to schedule and execute in the hybrid mode.
- Developed JNI program to support Pthread [8] for Java and also evaluated Java threads and Native Thread to schedule and execute in the hybrid mode.
- Developed JNI program to support Intel TBB [9] for Java.
- Evaluated the performance [10] of Java Native Thread and Native PThread on Win32 platform.
- Developed Native PThread [11] on Android platform using Android NDK.
- Explored the contrast [12] on GPGPU computing through CUDA and OpenCL.

2. Summary of the Research Work

The parallel programming model developed and examined with the framework model and its organization is shown in Figure 1. The encircled dotted portion of the framework shows the overview of the carried out research work. Parallel programs can be utilized on CPU, GPU, Distributed, Cluster, Grid, MPP and FPGA. The framework which the authors implemented is suitable to the multicore CPU platform for both task parallelism and data parallelism.

Multicore platforms allow developers to enhanced applications by intelligent partitioning of the dissimilar workloads on dissimilar processor cores. Application code can be improved to use multiple processor resources resulting in faster application performance. Successive multi-threaded applications on multicore platforms have dissimilar design thoughts than performing successive multi-threaded applications on single-core platforms.

The authors provided task parallelism on a desktop or laptop computer through Just Peculiar Algorithm, Java Native Access, Win32 thread and Pthread. Just Peculiar Algorithm has been used to set affinity for thread in multicore processor's environment.

Java Native Access mainly provided all the operating system related functionalities in which, they only use to create and set the affinity for thread in multicore processors. The task parallelism [7,8,13] on Win32 thread, POSIX thread were developed using JNI, which enabled to the usage of extra features like setting affinity for thread

to schedule the task parallelism on different multicore processors.

The data parallelism on native Intel TBB threads [9] is developed using JNI, which enabled to support data parallelism on different multicore processors. In native TBB, the authors have used parallel_for template, which allowed to perform addition, subtraction, multiplication and division operations. The native Intel TBB can also be extended in the future to use another template available in Intel TBB such as, parallel_scan, parallel_do, parallel_for_each, parallel_pipeline, parallel_sort and parallel_invoke.

The authors provided task parallelism on Android platform [11], which are developed using Android NDK. Android NDK provides a platform for specific structures and trusts on JNI expertise to stick the native code to the Android applications. They also deliberated, how android applications can facilitate in setting affinity using Pthread through Android NDK that can make Pthread to execute task parallelism in hybrid mode with Java threads.

Programming on GPU [12] is supported through SIMD architectures. General Purpose GPU (GPGPU) computing has allowed the GPU to arise as successful co-processors that can be employed to improve the presentation of

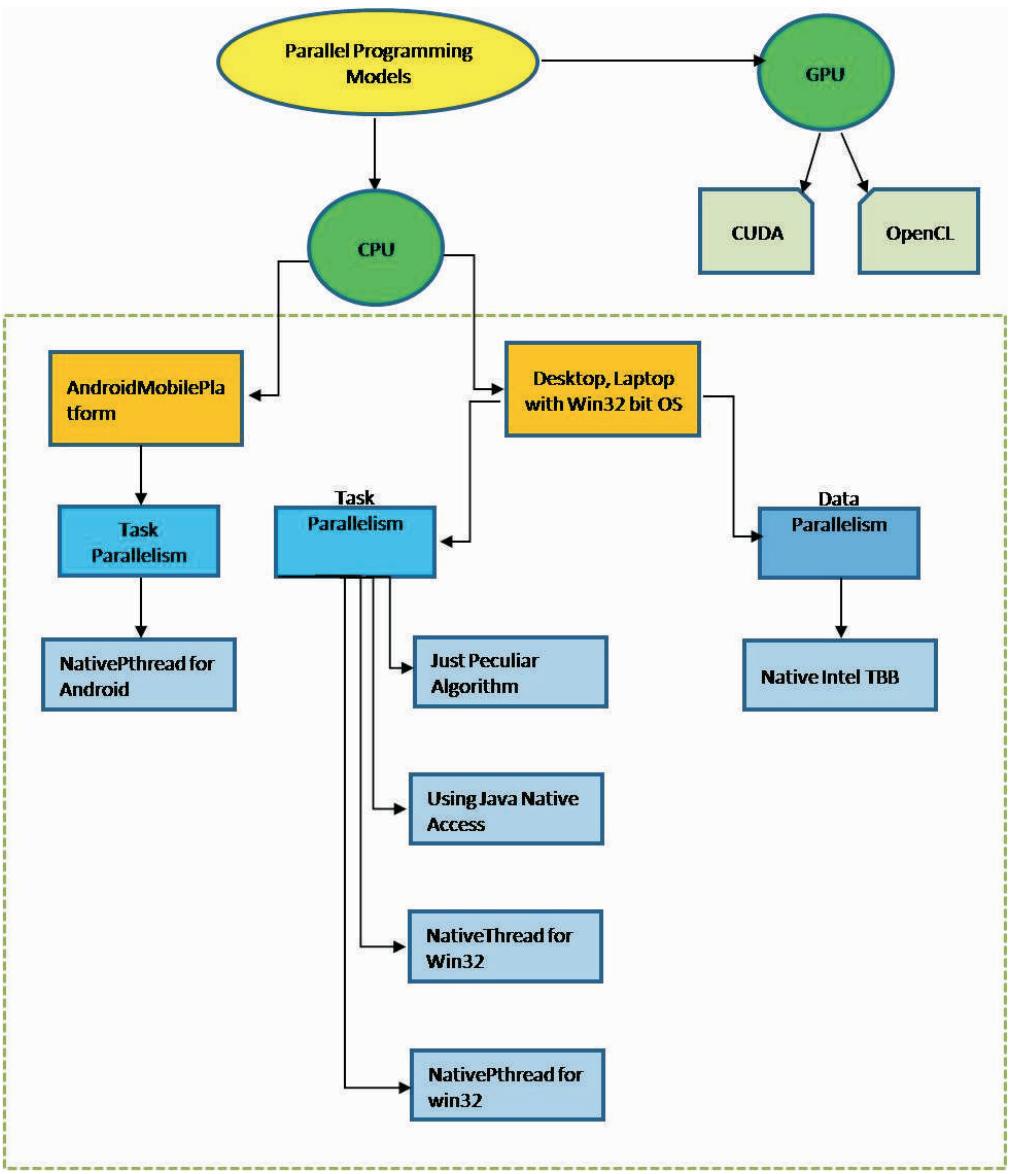

Figure 1. Framework for Parallel Programming Model

many dissimilar non-graphical applications. CUDA and OpenCL offer two different interfaces for programming GPUs. Java can facilitate for General Purpose GPU Computing. The authors identified the contrast between CUDA and OpenCL to help the HPC programmers to familiarize with GPGPU. The jcuda and jocl are the native libraries available for Java to perform data parallelism on GPU. However the jcuda or jocl libraries are not up to the present version available for CUDA or OpenCL, since there is a large opening on improving the data parallel library on GPU for Java language.

3. Promoting the Research

There are determined tasks, that promotes the research. They are listed below.

- The advantage of fixing CPU affinity is optimizing cache performances.
- Thread migration on multicore processor are able to migrate thread execution from the core processor to another.
- Java threads and NativeThread can be executed in the hybrid mode.
- Performance improvements are done in parallel programming models using Java Native Interfaces.
- Both task parallelism and data parallelism can be implemented to take advantage of the multicore processing environment.
- Supports more than one native thread methods in each program in different devices like Desktop, Laptop and Mobile Devices.

The research work has been identified and developed a parallel programming model through the Java Native Interface. It provided a prototype model to incorporate the features of another language to the Java language.

4. Challenges in Parallel Programming Implementation

Java Native Interface (JNI) is a robust aspect of the Java. JNI designate a method for controlling the program written in the Java programming language to jointly make effort with the native program that is written in C/C++. JNI permits accurate methods of Java classes to be applied natively and are still be entitled and used as normal Java methods. The Native Thread is scheduled by the operating system that is hosted in the virtual machine. There were challenges in the model implementations, which are listed below.

- Java does not support pointer concept, but C/C++ does. If any flaw occurs technically in pointer conversion, it may lead to memory leakage which are the major challenges of this research work to write native code to support affinity thread for Java.
- In this research, the authors have implemented both win32 thread and Pthread with limited functionalities and have not focused on all the other features of win32 thread and Pthread such as, thread priorities, thread synchronization, and access to some resource, which locks the resource that many threads may need to access. That can be accessed by only one thread at a time.
- In the research, the authors have implemented Intel TBB with parallel_for template to do data parallelism and they have not used all the other features of Intel TBB such as parallel_reduce, parallel_scan, parallel_do, parallel_for_each, parallel_sort, locks and atomic operations, a task scheduler and a scalable memory allocator.
- The authors are unable to implement all functionalities of Pthread on Android platform, since Android NDK provides a limited version of Pthread library.
- Except JPA model, all other parallel programming models that were developed are platform dependent.
- Assigning and retrieving the names of any thread is also a big challenge.

5. Future Directions

- The future direction will be focused on the windows thread, Pthread, and Intel TBB through JNI for windows 64bit platforms and other platforms.
- Boost threads, C^{++} 11 threads, and Intel Cilk Plus thread features have been also implemented on Win32 bit and Win64 bit platforms for Java.

- Improving the available native Intel TBB from Intel TBB, other than parallel_for loop.
- Improving the available Native Thread for win32 and NativePThread for win32 to the next level.
- GPU Computing for Java can be provided through the JNI. Through JNI, OpenCL and CUDA programming can be availed for Java. However, this is identified that jocl and jcuda are the Java binding for GPGPU computing, though this binding are not latest to the current maintained library on OpenCL or CUDA. Hence the authors can also bring a full-fledged version of OpenCL and CUDA programming for Java.
- The performance can be compared between Intel TBB on CPU data parallelization with jcuda and jocl on GPU data parallelization.

Conclusion

The parallel programming model has been developed for both task and data parallelism on Desktops, Laptops and Android mobile devices. The task parallelism are carried out using Just Peculiar Algorithm (JPA), Java Native Access (JNA), Native Thread for win32, Native PThread for win32 and Native PThread for Android, whereas the parallel programming model developed for the data parallelism on multicore processor uses Native Intel TBB. The data parallelism model of GPU is GPGPU computing. Most of the research works focus on native thread for Java on windows 32-bit platform. The listed future directions enable the researchers to enhance the developments of parallel programming models for Java on windows 64-bit. The researchers can attempt to implement boost threads, C^{++} 11 threads, and Intel Cilk Plus thread features for Java on windows 32 bit as well as on windows 64 bit platforms. It is not limited; the researchers have got opening to enhance other features.

References

[1]. Darryl Gove, (2011). *Multicore Application Programming For Windows, Linux, and Oracle® Solaris*, Pearson Education, ISBN:10: 0-321-71137-8.

[2]. Thomas Rauber, and Gudula Runger, (2010). *Parallel Programming for Multicore and Cluster Systems*. Springer-Verlag Berlin Heidelberg, ISBN 978-3-642-04817-3, DOI 10.1007/978-3-642-04818-0.

[3]. Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill, (2004). *Pattern Language for Parallel Programming*. Addison-Wesley, ISBN:10: 0321228111.

[4]. David B. Kirk and Wen-mei W. Hwu, (2010). "*Programming Massively Parallel Processors - A Hands-on Approach*", Elsevier.

[5]. Bala Dhandayuthapani Veerasamy, and G.M. Nasira, (2012). "Setting CPU Affinity in Windows-based SMP Systems using Java". *International Journal of Scientific & Engineering Research*, Vol.3, No.4, pp.893-900, ISSN: 2229-5518.

[6]. Bala Dhandayuthapani Veerasamy, and G.M. Nasira, (2012). "Parallel: One Time Pad using Java". *International Journal of Scientific & Engineering Research*, Vol.3, No.11, pp.1109-1117, ISSN 2229-5518.

[7]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2013). "JNT - Java Native Thread for Win32 Platform". *International Journal of Computer Applications*, Vol.70, No.24, pp.1-9, Foundation of Computer Science, New York, USA, ISSN: 0975 - 8887.

[8]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2014). "Java Native PThreads for Win32 Platform". *World Congress on Computing and Communication Technologies (WCCCT'14)*, pp.195-199, Tiruchirappalli.

[9]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, "Java Native Intel Thread Building Blocks for Win32 Platform". *Asian Journal of Information Technology*, Vol.13, No.8, pp.431-437, Medwell Publishing, ISSN: 682-3915 (Print), 993-5994 (Online).

[10]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2015). "Performance Analysis of Java NativeThread and Native Pthread on Win32 Platform". *International Journal of Computational Intelligence and Informatics*, Vol.4, No.4, pp.255-263, ISSN: 2349-6363.

[11]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2014). "Native PThread on Android Platform using Android NDK". *Karpagam Journal of Computer Science (KJCS)*, Vol.9, No.1, pp.1-18, ISSN : 0973-2926

[12]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2014). "Exploring the contrast on GPGPU computing through CUDA and OpenCL". *i-manager's Journal on Software Engineering*, Vol.9, No.1, pp.1-8, ISSN Print: 0973-5151, ISSN Online: 2230-7168.

[13]. Bala Dhandayuthapani Veerasamy and G.M. Nasira, (2014). "Overall Aspects of Java Native Thread on Win32 Platform". *Second International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA-2014)*, Vol. 2, pp.667-675, Bangalore, published in Elsevier in India. ISBN: 9789351072621.

ABOUT THE AUTHORS

Bala Dhandayuthapani Veerasamy is currently working as an IT Lecturer at Shinas College of Technology, Oman. He received his B.Sc in Computer Science from Bharathidasan University in 2000. He received his first master Degree (M.S) in Information Technology from Bharathidasan University in 2002 and he received his second master Degree (M.Tech) in Information Technology from Allahabad Agricultural Institute Deemed University in 2005. Presently, he is pursuing his part-time external PhD in the areas of Information Technology from Manonmaniam Sundaranar University. So far, he has presented more than twenty-five peer reviewed research papers in various International Conferences and Journals.

G.M. Nasira is currently working as an Assistant Professor in the Department of Computer Science, at Chikkannna Government Arts College, Tiruppur, India. She got her B.Sc (Computer Science) from Madras University, MCA from Bharathidasan University, B.Ed and M.Phil from Bharathiyar University. She got her Ph.D. in Computer Science from Mother Teresa Women's University, Kodaikanal with the specialisation if Artificial Neural Networks. She has published 12 research papers in referred Journals and presented 50 papers in various Conferences and Seminars. In addition, she has also authored a book titled 'Fundamentals of Middleware Technologies and Web Technologies.