This work-in-progress paper was presented as part of the main technical program at IEEE ETFA'2011

Development of Embedded Multicore Systems

Célio Estevan Morón
Federal University of São Carlos
Department of Computer Science, Brazil
celio@dc.ufscar.br

Allen D. Malony
University of Oregon
Department of Computer & Information Science, USA
malony@cs.uoregon.edu

Abstract

The concepts involved in the programming process of multicore systems have been quite well known for decades. The problem is to produce it in a form as easy as sequential programming. This new trend will change the way we think about the whole development process. We will show that it is possible to develop a multicore embedded system application using existing tools and the model-driven development process proposed. To do this, two tools will be used: VisualRTXC (available at www.quadrosbrasil.com.br) for generating the multithread communication/synchronization structures and a performance tool called TAU (available at http://www.cs.uoregon.edu/research/tau/home.php) for the tuning of the final implementation.

1. Introduction

It has become evident for some time that it is not economically wise to speed up further the processor clock in order to get an increase in the computer power. The result of this is an exponential increase in the power consumption and power dissipation. This recognition has forced manufacturers to look for alternatives to increase performance. Among the alternatives, the one that is proving to be very successful is to use several cores within a single chip. By using several cores, it is possible to improve the performance while keeping the power consumption unchanged. This trend has reached all areas of computing, from mainstream computation throughout embedded systems. All computers being sold nowadays are multicore and this will change completely the way we see the field of computing.

Be it in the form of specialized high-performance systems, or dedicated embedded systems, multiprocessor machines have been present among us for decades. The concepts involved in the programming process of multicore systems are quite well known in the field of Operating Systems and Parallel Programming. However, so far there seems to be no tool available which could be able to make the programming process of these systems as easy as it is for sequential programming. In order to be able to program multicore systems efficiently, we need knowledge in: (1) Parallel Programming Concepts; Mechanisms of Communication and Synchronization of Processes/Threads; (2) Architectural details of the processor (kind of caches, etc); (3) Tools to evaluate the performance and tuning of the system.

At a certain degree, those requisites are already used in the arena of parallel programming and real-time systems. In the real-time field to fulfill the time constraints, while in parallel programming in order to try balancing the load and avoid bottlenecks.

The question that major manufacturers are trying to answer is the effect of the multicore processor in the software development process. This question is quite complex and we will answer it for a subset of computing; that is, embedded systems.

One typical use of multitasking/multithread is in the development of embedded multicore systems. The term embedded system [2] refers to any computer system built within a device and working as part of it. Most of the embedded systems have real-time features associated to them. Embedded systems using microcontrollers typically rely on a Real-Time Operating System (RTOS) to provide multitasking capabilities. RTOSs improve performance and enable more sophisticated programs on less expensive processors.

The development of a multitask/multithread system is inherently complex, since it involves the need of synchronization among tasks/threads and the analysis of data dependences. Multitask/multithread systems are composed by several processes, called tasks/threads, which depend upon each other to execute in a proper manner. To create these systems, the developer needs to split the application modules into a group of tasks/threads able to run simultaneously. Besides, it is very important to apply efficient methods for communication and synchronization to make sure that the processes interact correctly.

This paper is organized as follows: Section 2 presents the Programming Models for multicore systems. Section 3 presents our Model-Driven

978-1-4577-0018-7/11/$26.00

Development Process. Section 4 is a Case Study that shows how to apply our method. A conclusion is presented in Section 5.

2. Programming Models

Basically, there are two techniques to program multicore systems: multitasking and multithreading. Multitasking is the ability to run multiple processes at the same time. The processes can then be assigned to different cores in order to extract the performance benefits. For embedded applications, multitasking is a key technique that can lead to substantial performance improvements or reductions in cost.

Multithreading is one of the main techniques that allow users to get the performance benefits of general multicore processors. However, multithreading requires that applications are designed in such a way that the work can be completed by independent workers, acting in the same process. The different processes can be placed to run in different cores.

There have been several attempts to offer new parallel programming models, languages, and libraries for multicore systems, through which programmers can provide additional information related to the parallelism in their programs. Among these attempts are Map-Reduce [3], Cilk [4], UPC [5], X10 [6] and STAPL [7].

A considerable amount of research in the area of visual environments have been carried out aimed at the development of software able to reduce the difficulties found in the development of parallel systems. This effort resulted in user-friendly tools such as TEV [8], PVMGraph [9], Millipede [10], TRAPPER [11] and GRADE [12, 13]. Even though there are several differences between these environments, they all focus on making the program development easier, providing graphical representations to map paradigms or libraries designed for the development of parallel systems.

In despite of the benefits provided by the tools described above, it is clear that they were not designed to support the development of embedded multicore systems. These environments were rather designed for multitask systems based on workstations or workstation clusters. In these architectures, the processing nodes are distinct machines, interconnected through a high-speed network and running general-purpose operating systems.

3. Development of Multicore Embedded Systems

We are proposing a development process that can be carried out in a cyclic way generating more refined versions of the application at each iteration. Within each cycle the tasks are categorized into five Disciplines: Modelling, Partial Code Generation, Implementation, Testing/Debugging and Tuning. The software engineer may use any analysis and design method in conjunction with the proposed development process, including object-oriented methods such as UML [14], or even other approaches.

The Development Process will use two existing tools, VisualRTXC [15] and TAU [16] to help in the programming of the system by providing an intuitive user interface and high-level design objects that are tightly coupled to the underlying kernel architecture. This development process allows the developer to rapidly move between design concepts and generated C code. In addition, it provides visual abstraction and design help for each of the typical phases of the development life cycle.

The representation of applications through graphs is another advantage offered by a tool like VisualRTXC [15], since this approach is familiar to most designers and facilitates the use of the services provided by commercial kernels. As a result, the productivity of the development team is highly increased and as a consequence, better results can be obtained in less time at lower costs.

The possibility of quick experimentation makes the proposed development process quite adequate for the creation of prototypes of the multicore embedded application during the initial stages of development. Through the creation of prototypes, problems existing in the design can be detected and fixed early, minimizing the consequences of them.

In addition, the division of the graphical representation into layers allows the application to be structured as a hierarchy of subsystems. This modular approach makes our approach ideal for the development of large multicore embedded applications, where it is impossible to represent the whole system within a single diagram.

The combination between the proposed model-driven development process and the tools VisualRTXC [15] and TAU [16] also facilitates the integration of different tools into a single programming environment, allowing the same graphical notation to be used by the environment tools throughout the different development stages.

3.1. VisualRTXC Implementation Tool

A large number of research in the area of visual environments have been carried out aimed at the development of software able to reduce the difficulties found in the development of parallel systems. Using those tools, it is possible to specify the parallelism of the program at a high level of abstraction and from which the source code can be automatically generated. In the case of embedded systems, tools for their development have been aimed at maximizing the

performance of the hardware rather than to improve the user productivity.

VisualRTXC is a graphical tool designed to help the development, documentation and visualization of embedded applications. It can be thought of as a layer above the services offered by commercial kernels, acting mainly over basic structures used by these kernels such as tasks, semaphores, resources, timers and others.

Differently from the traditional approach [17], where the program implementation is carried out on the source code, VisualRTXC offers a higher abstraction layer, where it is possible to represent graphically [1] most of the embedded application characteristics. As VisualRTXC allows the user to divide the application into several layers, it is possible to run each layer code on a specific core.

3.2. TAU Performance Tool

The TAU (Tuning and Analysis Utilities) parallel performance system is the product of fourteen years of development to create a robust, flexible, portable, and integrated framework and toolset for performance instrumentation, measurement, analysis, and visualization of large-scale parallel computer systems and applications. The success of the TAU project represents the combined efforts of researchers at the University of Oregon and colleagues at the Research Centre Juelich and Los Alamos National Laboratory.

TAU provides an API that allows programmers to manually annotate the source code of the program. Source level instrumentation can be placed at any point in the program and it allows a direct association between language and program-level semantics and performance measurements. Using cross-language bindings, TAU provides its API in C++, C, Fortran, Java, and Python languages.

4. Case Study - Mandelbrot

Code that generates the Mandelbrot set is a favourite target for evaluating performance in embedded systems. Embedded systems generally require a high amount of image processing to perform and the Mandelbrot set can be adjusted to demand the computer power necessary for evaluation. Beside this, the Mandelbrot set is familiar to most users and the code required to generate the images is quite simple.

Beginning with the application requirements, the software engineer models the application with the help of VisualRTXC. The modelling step is carried out at two levels. First, the level of the system where the software engineer represents the executing modules (tasks, threads and exceptions) and their kernel primitives, as well as all communications and synchronizations. The second level refers to the modelling of the executing modules, and is carried out after it has been made explicit at the first level of modelling.

Figure 1 shows the Layer Diagram and Figure 2 the Code Diagram.

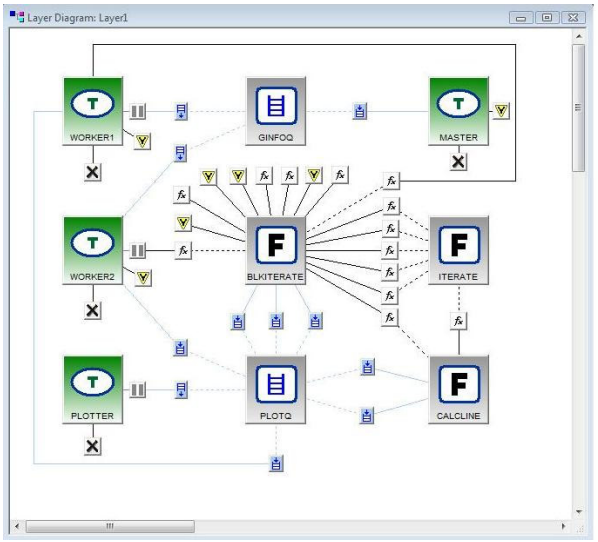

Figure 1 - Layer Diagram

Figure 2 shows the code that is generated automatically together with the code that was inserted by hand. The code was executed using the simulator of the Real-Time Kernel RTXC™ (RTXC is a Trademark of Quadros Systems Inc.) and VisualRTXC. A Core2 Duo processor was used to run the simulator of the real-time kernel RTXC with Windows Vista.

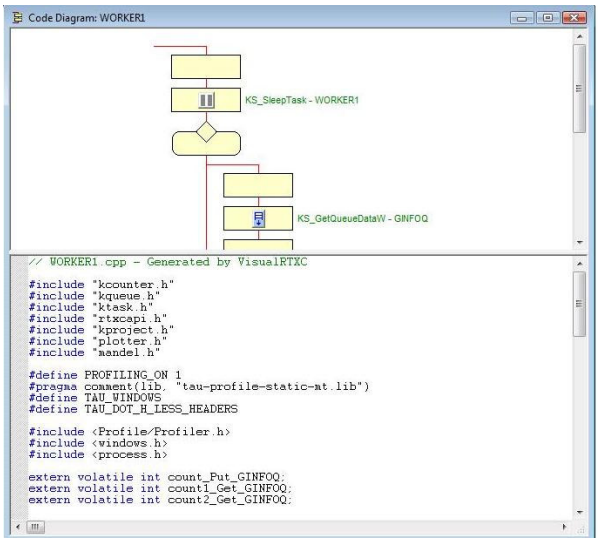

Figure 2 - Code Diagram

The code was instrumented with commands of the performance tool TAU. The output of TAU can be seen in Figure 3.

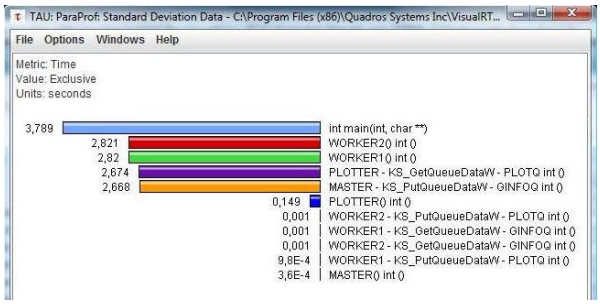

Figure 3 – TAU ParaProf

The total execution time was 7.067 seconds. Analyzing the times provided by the performance tool, it is possible to see a delay in the manipulation of queues (fourth and fifth line in Figure 3). The delay in task PLOTTER makes sense, because it will take the dots only after these have been calculated. However, the delay in task MASTER is due to the size of the queue which is quite short. Increasing the size of the GINFOQ queue and running the application again, results on the performance shown in Figure 4.

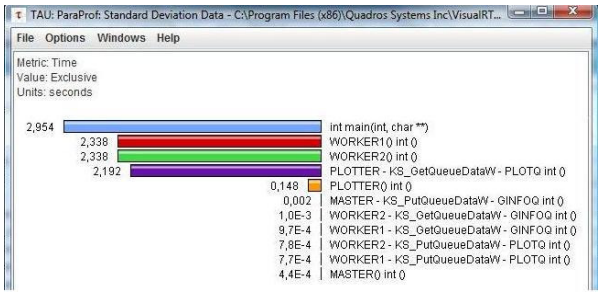

Figure 4 – TAU ParaProf

We can see that the delay imposed to the GINFOQ queue now is only 0.002 seconds. The total execution time of Mandelbrot after the tuning is 5.850 seconds.

5. Conclusion

The exclusive use of general purpose tools, such as compilers and text editors, is not adequate for managing the complexity of most multicore embedded systems. Providing a development process with the help of a programming environment with facilities aimed at the development of these systems, represents a significant step to reduce the drawbacks that make the implementation one of the biggest bottlenecks during the design of those kinds of systems.

We have shown that with two existing tools, it is possible to develop multicore systems in an efficient manner. Our approach for the process of development is carried out by a combination of a graphical tool with a performance tool. Using a graphical tool, it is possible to better understand the communication/synchronization of tasks/threads, while the performance tool allows tuning the system.

The possibility of quick experimentation makes the proposed development process quite adequate for the creation of prototypes of the multicore embedded application during the initial stages of development. Through the creation of prototypes, problems existing in the design can be detected and fixed early, minimizing the consequences of them.

In addition, the division of the graphical representation into layers allows the application to be structured as a hierarchy of subsystems. This modular approach makes our process ideal for the development of large multicore embedded applications, where it is possible to represent each part of the system within a single diagram.

References

[1] Gross, J.L., Yellen, J. 2005. Graph Theory and its Applications, Chapman & Hall/CRC.

[2] Berger, A.S. 2001. Embedded Systems Design: An Introduction to Processes, Tools and Techniques, CMP Books

[3]www.labs.google.com/papers/mapreduceosdi04.pdf.

[4] www.supertech.csail.mit.edu/cilk/.

[5] http://upc.gwu.edu/

[6] www.research.ibm.com/x10

[7] parasol.tamu.edu/groups/rwergergroup/research/stapl/

[8] Ribeiro, J.R.P., Silva, N.C., Moron, C.E. A Visual Environment for the Development of Parallel Real-Time Programs. Lecture Notes in Computer Science, v. 1388, p. 994-1014, 1998.

[9] Justo, G. 1996. PVMGraph: A Graphical Editor for the Design of PVM Programs, Technical Report, University of Westminster.

[10] Itzkovitz, A., Schuster, A., Wolfovich, L. 1996. Millipede: Towards Standard Interface for Virtual Parallel Machines on Top of Distributed Environment, Technical Report 9607, Technion liT.

[11] Heinze, F., Schäfers, L., Scheidler, C., Obelöer, W. 1997. Trapper: Eliminating Performance Bottlenecks in a Parallel Embedded Application, In: IEEE Parallel & Distributed Technology: Systems & Technology, 5, Proceeding p. 28-37.

[12] Kacsuk, P., Dózsa, G., Fadgyas, T. 1996. Designing Parallel Programs by the Graphical Language GRAPNEL, Special Issue of the Euromicro Journal: Parallel Systems Engineering.

[13] Kacsuk, P., Dózsa, G., Kovács, J., Lovas, R., Podhorszki, N., Balaton, Z., Gombás, G. 2003. P-GRADE: a Grid Programming Environment, In: Journal of Grid Computing, 1, Proceedings p. 171-197.

[14] Douglass, B.P. 1998, Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison Wesley.

[15] VisualRTXC - http://www.quadrosbrasil.com.br

[16] TAU, University of Oregon, LANL, and RCJ ZAM, http://www.cs.uoregon.edu/research/tau/home.php

[17] Eclipse 2008. Eclipse - an open development platform, URL: http://www.eclipse.org.