Old Tricks for New Architectures: Teaching CSP for Multi-core Programming

Brian Vinter[1]
[1] eScience centre, University of Copenhagen,
2100 Copenhagen, Denmark
vinter@diku.dk

Abstract. The advent of multi-core processors beyond the trivial two-cores, has spawned a renewed interest from students in learning to program these processors. Many have tried the usual approaches, threads, OpenMP and even MPI, but soon realize that the challenge lies not so much in managing parallel code as planning for data-locality. To help students learn to write highly scalable and portable applications for multi-core architectures of unknown size we have developed a new class, Extreme Multi Programming, that uses Communicating Sequential Processes[1] as the driving design and implementation principle. This paper describes the philosophy and outline of the class, introduces some of the tools that enable students to implement CSP-based applications on multi-core architectures and spends some time covering a few previous exam projects and the lessons we and the students learned from them

Keywords: CSP, Multi-core

1 Introduction

As Moores law continues to increase the number of transistors on a single chip, the imagination of CPU-designers seems expended and the common approach to utilization of the many transistors is to go multi-core. Today any PC- and most game-console CPUs are multi-core. Unfortunately the introduction of multi-core CPUs only adds to our problems and the challenges we faced with ordinary single core CPUs are simply scaled with the number of cores.

The University of Copenhagen thus started a class to get the students up-to-speed with the multi-core world. We start by motivating the introduction of multi-core processors and shows why they are simply enhancing our known problems. We then go on to a proposal for the revitalization of an old and established programming paradigm as a means to address these problems in multi-core processors, Communicating Sequential Processes. Finally we will look into a number of multi-core architectures and show how CSP may be used to program all of them.

1.1 Motivation I

When presented with a multi-core processor most students intuitively think that the problem they are facing is the parallelization of an application to run on the multiple cores. While this of course is partly true we need to make the students appreciate why the problem with programming multi-core architectures is much more focused on memory management then process parallelization. The argument as presented to the students goes as follows:

A modern processor runs just over 3GHz, if we chose 3.3GHz as a typical fast processor then the cycle time for that processor is 1/3 of a nano-second. When we further consider the CPI of a modern processor we can easily find processors with a CPI between 1/4 and 1/3, if we round off coarsely we end up having no problem accepting that the average time for an instruction execution in a processor today is 0.1 nano-second. Consider then the fact that the speed of light allows a signal is approximately 30 cm/ns this mean that on the average execution time of an instruction a light signal will travel 3 cm, this translates into the fact that, seen from the perspective of the CPU, the memory is very far away.

The class continues to cover the motivation behind using the vast majority of a processors transistors for caching and branch-prediction and demonstrates how these techniques are providing increasingly poor return on investment. The motivational class finished with the provocative message that the students who believed that OpenMP was the answer (to programming multi-core architectures) had not understood the question.

1.2 Motivation II

Once that hardware motivation is in place, we go on to motivate why classic multithreaded programming paradigms are unsuited for multi-core programming. The students witness demonstrations why traditional locks are convenient but very sensitive to programmer errors and thus lacks robustness, why synchronized statements in Java are easy to use but are unable to eliminate starvation – and how true monitors can function correctly, robustly and ensure fairness/eliminate starvation.

The next part of the motivation is based on hardware scalability reasoning. Since we get increasingly more transistors per processor we will get increasingly more cores in our multi-core architecture. The logical conclusion from this fact is that writing applications that scale to a fixed number of cores will, in a not so distant future, not be able to utilize a new processor, since this processor will have more cores than the application was designed for, the Intel teraflops research chip is used as a not too distant processor architecture[2].

2 Communication Sequential Processes

Communication Sequential Processes are then introduced as an alternative to the more well known threading and coordination mechanisms. The students are given one (1!) lecture on formal CSP notation and algebra, in the remainder of the class graphical equivalence proofs are used over formal algebraic proofs wherever possible, non the less a number of the students that have signed on to the class will drop out at this point, once the realize that the class is not only on programming techniques.

2.1 Programming CSP

The following lectures are a mixture of designing applications using CSP and practical programming using CSP. The first CSP tool that is used is JCSP[3] where the students soon realize that writing a multithreaded program with JCSP is in fact easier than using standard Java Threads. Once the students are comfortable with JCSP the model is extended beyond the multi-core world by introducing the network enabled version of JCSP[4] and the students see how their, correctly designed, multi-threaded CSP programs can utilize computational clusters by simply adding a network manager to the initiation of the application.

To demonstrate that Java is not the only target language for CSP, PyCSP[5] is introduced as the absolutely most easy way of writing CSP applications and C++CSP[6] is introduced as a means of using CSP in a programming language that provides much more low level control of the application.

Finally a commercial CSP product, JIBU[7], is introduced. JIBU demonstrates the ability of using CSP in a commercial production environment and shows how even Windows based applications can easily be implemented using CSP techniques. A funny side-story is the fact that JIBU in fact started as a CSP project the very first time this class was ever given. The students working on the project allowed themselves to be completely sidetracked by the project and ended up dropping out of University to market JIBU as a product.

3 Multi-core hardware

Once the students are all able to write complex applications using CSP, including the use of equivalences and compositional semantics, the class continues to cover several multi-core architectures and how CSP may be mapped onto them.

The architectures that are covered are

- Intel and AMD x86 multi-core processors
- SUN T1 and T2
- Xenon (Microsoft Xbox 360)
- CELL-BE

4 Exam Projects

The students are graded based on two programming assignments that they bring home for 30 days before handing in a report on their design, implementation, performance and general findings. The examination projects are designed in such a way that one application is very natural in its concurrency while the other is focused on parallelizing an existing sequential algorithm. The students may implement the exam in any programming language they choose. They are also allowed to use a language that has no CSP extension library as long as they keen with the CSP design principles.

Last year the projects were

- Digital circuit simulation
- Pawn-chess game

The idea behind the digital circuit simulation was to enable the students to demonstrate that they master the concurrency aspects of CSP as well as the equivalences and compositional semantics, i.e. one sub-task in the project was to make an 8-bit full-adder using Boolean logic processes and by writing the adder as native code, and then demonstrate the equivalence between the two. The pawn-chess, chosen over a full chess-game because the rules are much more easily implemented, enables the students to consider the level of concurrency that is of interest.

This year the exam projects are both more eScience oriented and are defined as:

- Ant-hill simulator
- Smith-Waterman sequence alignment

The motivation behind these problems is similar to those of last year, the Ant-hill simulation allows the students to model complex behavior in a natural concurrent model with open ended parallelism, while the Smith-Waterman parallelization allows them to demonstrate control over complex dependencies in a parallel application.

5 Conclusions

We have designed and taught a class where CSP is used as the principle means of coordination for parallel applications for multi-core architectures. Overall the class has been a success and we have multiple examples of former students who decide to use CSP based design and implementation for later projects in their education, and who have successfully introduced the paradigm to industrial workplaces.

References

1. Hoare, C.A.R.: Communication Sequential Processes. Communications of the ACM, 21(8):666--667 (1978)
2. http://techresearch.intel.com/articles/Tera-Scale/1449.htm
3. Moore, James. Native JCSP - the CSP for Java library with a Low-Overhead CSP Kernel, Proc. Of Communicating Process Architectures 2000, 263—274 (2000)
4. Welsh, P.H and Vinter, B. Cluster Computing and JCSP Networking, Proc. Of Communicating Process Architectures 2002, 203—222 (2002)
5. Anshus, O. J., Bjørndalen, J. M and Vinter, B. PyCSP - Communicating Sequential Processes for Python, Proc. Of Communicating Process Architectures 2007, 229—248 (2007)
6. Brown, N. and Welsh, P. H. An Introduction to the Kent C++CSP Library, Proc. Of Communicating Process Architectures 2003, 139—156 (2003)
7. http://axon7.com/