On Multicore Software Engineering in the Enterprise Computing Domain

Applies to:

Refers to enterprise software in general. Related to, e.g., ERP, SCM, ABAP

Summary

Given present tools and operating systems, it is anticipated that a large percentage of enterprise applications may not be capable of automatically exploiting the steadily increasing number of processor cores. Current technology and market trends are reflected concluding that enterprise software developers require a fundamental re-thinking: to design and to re-factor their products to embody parallel business logic and to make concurrency a scalable and manageable property. The article discusses the challenges imposed to this mission and surveys a range of parallel programming approaches which may be leveraged to foster the industrial adoption of parallel enterprise software engineering.

Author: Bernd Scheuermann

Company: SAP AG

Created on: 12 September 2012

Author Bio

Bernd Scheuermann holds a diploma in industrial engineering and a doctoral degree in applied computer science from the University of Karlsruhe. He joined SAP Research Center Karlsruhe in 2006. In his current position as Senior Researcher his primary interests are dynamic optimization and parallel algorithms targeting multicore architectures and Field-Programmable Gate Arrays (FPGA).

Table of Contents

Introduction

“Ultimately, the advice I’ll offer is that developers should start thinking about tens, hundreds, and thousands of cores now in their algorithmic development and deployment pipeline. This starts at a pretty early stage of development; usually, the basic logic of the application should be influenced because it drives the asymptotic parallelism behaviors.”

Anwar Ghuloum, Intel Microprocessor Technology Lab [1]

As highlighted by Patrick Leonard, Vice President for Product Development, Rogue Wave Software [2], the “dilemma” is that a large percentage of mission-critical enterprise applications will not “automagically” run faster on multi-core servers. Indeed many will actually run slower. The majority of enterprise applications have not been programmed multi-threaded. Typically, a single-threaded enterprise application will not be able to exploit the additional cores available without risking ordered processing. Hence, most of the cores are likely to remain idle.

Currently, neither the application servers nor the OS are capable of effectively scheduling single-threaded enterprise applications to multiple cores. Presumably, there remains no other solution than programming or re-factoring enterprise software to embody parallel business logic. It is anticipated that parallel programming becomes a very hot issue in enterprise software engineering, and in the long run, vendors convincingly bringing scalable and manageable concurrency to its products will gain advantages in the competitive market. Consequently, it is important that we make it as easy as possible for programmers of enterprise software to exploit the latest developments in multicore architectures, while still making it easy to target future (and perhaps unanticipated) hardware developments.

Current Technology Trends

According to Moore’s law the number of transistors on CPUs doubles every two years. Generally, an increasing number of transistors means higher compute power. Today this trend is still valid even though the clock speed of latest processors has experienced stagnation. Further increasing clock speed comes with a trade-off: Higher clock rates imply higher power consumption. Also cooling such devices requires increasingly more energy. The ratio between compute power and energy does not scale in a linear manner. This phenomenon is also referred to as the “power wall”. Shortly, the additional amount of compute power is too expensive with respect to the electric power needed to achieve it [3].

A second trend can be observed in modern chip production technology which yields steadily smaller chip components through miniaturization. As a consequence more processors can be placed on the chip area which gave rise to the emergence of current multicore devices. The current strategy is to accommodate more small processors on one chip instead of increasing the speed of a single processor. Quad-core and octa-core processors become standard in current desktops and laptops. Additionally, techniques like hyper-threading further increase the potential degree of parallelism as each core may schedule additional hardware-scheduled threads.

The implications for software developers are tremendous: They can no longer rely on clock speed evolution to accelerate the speed of their applications for single CPUs. Moreover, programming for a single processor was based on a deterministic computational model such that software could be understood and be debugged in a straight-forward manner. This has changed significantly. Typically, software programmed for single CPUs does not automatically adapt itself to the higher number of cores available. Developers themselves have to adapt their code, to parallelize it and to distribute its workload efficiently. Software developers need to handle the existing parallelism in order to implement new compute-intensive features and to prevent degrading performance.

The challenges involved with these market and technology trends have also been analyzed by Gartner identifying the Seven Grand Challenges Facing IT published in 2008 [4]. Two of these challenges directly address the problems involved in parallel multicore software development. Gartner reports that parallel programming was required to make efficient use of the multitude of cores available. The challenge, however, relied in dividing a problem into smaller sub-problems which were then to be executed on the individual processors. Consequently, Gartner outlines that the key issues were to effectively break up processes into specific sub-processes, to achieve an efficient multicore task scheduling and to design the architecture of the parallel processing environment.

Parallel Programming Challenges

Typically, when introducing the parallel programming paradigm to software engineering industry, companies have to deal with a range of challenges. These comprise technological obstacles as well as educational challenges which need to be addressed to establish expert developer teams for parallel programming.

Technological Challenges

Writing parallel algorithms there exists a range of problems commonly not present when developing sequential code. Such problems comprise [5]:

- Race conditions: As two different parallel procedures may access shared objects, the order in which calculations are evaluated and the order of communication becomes relevant and may determine the output of the parallel algorithm.
- Locking: Parallel programs using shared memory must employ locking mechanisms in order to avoid inconsistent results. Such locking can be a costly and error-prone activity.
- Deadlocks and Livelocks: Dependencies may produce a blocking of the program execution which must be prevented.
- Decomposition: The problems addressed by the software must be partitioned appropriately so as to derive a set of parallel tasks.
- Granularity: The previously mentioned decomposition must be done in a way to achieve a suited degree of granularity. Too fine-grained and algorithms tend to become inefficient due to surplus administrative routines. Too coarse-grained and programs may not be able to utilize the available computing resources.
- Scalability: Programs should exhibit a near-optimal scaling behavior to flexibly use an increasing number of processors or cores.
- Load balancing: Suitably scheduling tasks and processes to the available cores is essential to ensure the desired QoS.

Parallel Programming Expertise

Clearly being able to effectively exploit multicore systems is an essential requirement to today's and future developers of enterprise software. One of the main challenges is to identify or to create a programming model which comes with a suitable level of abstraction but still ensuring efficient utilization of the hardware resources. To classically-trained developers it is already very difficult to profit from the computation power offered. Unfortunately, only few highly-skilled programmers are actually exposed to parallel programming techniques.

In their report "Seven Grand Challenges Facing IT" [4], Gartner addresses the need to increase programmer productivity 100-fold. The authors emphasize that business' and society's demands for software development increased whereas the amount of qualified developers and software experts declined. Consequently, the productivity per programmer has to be improved.

Fostering Industrial Adoption of Parallel Multicore Programming

From the emergence of the digital computer until very recently, hardware manufacturers have been able to rely on the effects of Moore's law to produce processors with steadily increasing clock frequencies. Accordingly, enterprise software industry benefited from every new processor generation in a more or less automatic fashion. As a consequence, developers remained bound to the inherently sequential von Neumann programming model as this became established since the very beginning of the digital computing era. Although programming language technology evolved over time (e.g. structured programming, object-oriented programming, abstract modeling), still these improvements were mainly driven by the need to make increasingly large software systems more manageable instead of effectively utilizing available hardware capabilities.

Presently, these circumstances are changing, and actually changing at a very high pace. Forthcoming multicore systems will evolve from today's common dual-core or quad-core architecture to systems holding hundreds or thousands of cores in near future. Next to the rapidly increasing degree of parallelism, cores tend to become more heterogeneous combining various specialized cores on a single processor. Unfortunately, the evolution of software engineering tools lags behind the revolution seen in the multicore hardware sector. This gap between software and hardware maturity continues to grow, and it becomes obvious that the von Neumann programming model has reached its limits.

It is obvious that enterprise software industry needs to introduce parallel programming to their development departments, although it must be ensured to supply the proper form of parallelism for the right problem along with the appropriate tools supporting it. Whereas some parallel codes may inherently require complex and error-prone interaction among threads, others may expose a comparatively simple logic. In any case, the parallel programming principles introduced should be able to express such logic at any degree of complexity.

Subsequently, it is motivated to examine four different (non-exclusive) approaches to parallel programming and its applicability in the enterprise software domain: functional programming, parallel programming patterns, auto-tuning and domain-specific environments.

Functional Programming

It is proposed to investigate into the concept of functional programming as an alternate high-level engineering approach for enterprise software. Functional programming languages, like e.g. Haskell or SaC[1] (see also related EU project ADVANCE [5]) expose various advantages with respect to parallel evaluation. A key property is that such languages come without side effects. Therefore parallel execution is always safe and decomposition is facilitated. The order, in which programs are executed, becomes irrelevant and the output of the program will always be the same. Indeed the results will even be identical with the sequential counterpart. Additionally, it is ensured that when a sequential algorithm terminates, also the parallel variant will terminate. Another important aspect regarding industrial adoption is that such programs can be debugged sequentially which is unreached by other parallel programming paradigms and may lead to huge savings. The parallel implementation will have impact on behavioral aspects only, e.g. memory usage or runtime performance. The order of IO-operations is entirely described by the functional program such that race conditions or unexpected outputs cannot appear. Likewise deadlocks cannot appear either because the data and control structures of functional languages avoid unresolved dependencies between tasks. Locking and synchronization can largely be controlled within the implementation itself. Finally, functional programming facilitates implementing control processes to dynamically adapt granularity at execution time which in turn provides advanced tuning capabilities with respect to load balancing and dynamic scaling [6].

Parallel Programming Patterns

Parallel patterns can be considered as a kind of vocabulary for developers. Before implementing a parallel program, developers should reflect about the algorithmic structures, the data types and also about parallel patterns that should be used. It is a prerequisite to train enterprise software developers in the usage of such patterns and to drive the exchange of experiences. A result could be a catalogue of such parallel patterns which are made available for direct re-use. Such parallel patterns differ from rather simple motifs characterizing core computational procedures but being less useful as guidance for code development. Parallel patterns, however, may not offer the indisputable answer to every parallel programming mission. In some cases converting sequential code to a parallel variant may require to develop new parallel structures while challenging the programmer to identify parallelization opportunities, making own high-level decisions on the distribution of data and communication and the required synchronization.

Autotuning

Autotuning approaches enhance parallel programming by providing additional semantic information (e.g. extra-functional requirements or properties) which would otherwise not be available to compilers. Such information, however, must commonly be provided by the programmer which requires additional labor efforts. Nevertheless, the prospective benefits of autotuning, like portability across different machines or generations of machines, may quickly amortize this extra work. It is foreseen that in parallel computing such autotuning systems will become even more profitable than in the sequential case. The effort of annotating code with semantics may to a certain degree be facilitated by means of using code generators, primitives or data parallel operators [7].

[1] Homepage: www.sac-home.org

Domain-specific Environments

Another way to provide an abstraction of parallel hardware is provided by means of domain-specific environments such like domain-specific languages or libraries. Currently a wide range of such tools are used including e.g. Matlab, Labview, Apple Core Audio or the PureMVC framework. Widely used and therefore should be mentioned is Google's Map Reduce framework. Generally, the goal of such tools is to entirely hide the parallel structures and to exploit domain knowledge to automatically parallelize function calls. In practice, however, many users are disappointed with the actual performance of their deployed applications and attempts to fix such problems often remain unsuccessful as the debugging tools still lack the domain knowledge needed. Programming environments should therefore be complemented by domain-aware tools for compilation, debugging and re-factoring. In particular, it should be possible to library developers to supplement code by domain-specific information which support the compiler when translating applications that use their libraries [7].

Conclusion

This SCN article introduced to the current obstacles when adopting multicore software engineering in the enterprise applications domain. Market and technology trends were reflected along with the challenges associated with parallel programming. It was proposed to consider four different approaches and to examine their applicability in the enterprise software domain: functional programming, parallel programming patterns, auto-tuning and domain-specific environments. Such techniques may be selected, combined and applied depending on the respective business application to be developed or re-factored.

Apparently, newly introduced programming models and runtime environments are required to co-exist with currently available software for up to several forthcoming decades. Hence, we cannot punish existing enterprise software stacks, while we should foster and reward the industrial adoption of practices for concurrent software engineering.

Bibliography

[1] Anwar Ghuloum. Unwelcome Advice, Research@Intel blog, http://blogs.intel.com/research/2008/06/30/unwelcome_advice, June 2008.

[2] Patrick Leonard. The Multi-Core Dilemma, Intel Software Blog, http://software.intel.com/en-us/blogs/2007/03/14/the-multi-core-dilemma-by-patrick-leonard, March 2007.

[3] D. Spath, A. Weisbecker, and E. Hebisch, editors. Market Overview of Tools for Multicore Soft-ware Development. Fraunhofer Verlag, 2010.

[4] Gartner. Seven Grand Challenges Facing IT, http://www.gartner.com, April 2008.

[5] C. Grelck, K. Hammond, H. Hertlein, C. Jesshope, R. Kirner, B. Scheuermann, H. Schöner, A. Shafarenko, I. te Boekhorst, V. Wieser. Engineering Concurrent Software Guided by Statistical Performance Analysis. In: K. de Bosschere, E. D'Hollander, G. Joubert, D. Padua, F. Peters, M. Sawyer (eds.). Applications, Tools and Techniques on the Road to Exascale Computing. Advances in Parallel Computing 22, pp. 385-396, IOS Press 2012.

[6] K. Hammond, K. Why Parallel Functional Programming Matters: Panel Statement, In Reliable Software Technologies - Ada-Europe 2011, Lecture Notes in Computer Science, vol. 6652, pages 201-205, Springer, 2011.

[7] S.V. Adve et al. Parallel@Illinois. Parallel Computing Research at Illinois — The UPCRC Agenda. http://www.upcrc.illinois.edu/documents/UPCRC Whitepaper.pdf, November 2008.

Related Content

http://scn.sap.com/docs/DOC-5720

http://scn.sap.com/docs/DOC-31373

http://scn.sap.com/people/oliver.mainka/blog/2009/04/17/if-it-were-a-hundred-times-faster

Copyright