

On some Potential Research Contributions to the Multi-Core Enterprise

a position paper

Oded Maler

CNRS-VERIMAG
2, av. de Vignate
38610 Gieres
France
Oded.Maler@imag.fr

1 Introduction

This document sketches some topics identified by the author, together with colleagues and industrial partners, as crucial for the success and proliferation of multi-core architectures. The material is based on observations made in the ATHOLE project¹ whose main focus is on low-power multi-core mobile architectures for stream-processing applications. Not all these observations are valid for all potential applications of the multi-core concept.

We start with some motivation. More and more complex electronic gadgets are designed and sold under tough competition constraints. They should satisfy the conflicting goals of *high performance*, *low power*, and *short development time*. Traditionally low-level development (hardware, micro-code) is better for optimizing performance, while high-level software provides for a more flexible and effective development process. One of the attractive features of hardware is its *inherent parallelism* while software (and algorithms in general) is traditionally more of a sequential nature.² The XSTREAM architecture around which the ATHOLE project is centered is based on the following premise: the same silicon area which is dedicated in traditional designs to special-purpose hardware can be allocated differently to yield a computation and communication fabric consisting of computation nodes connected via a network on chip (NoC). The computation nodes are simple general-purpose processors with local memory, augmented perhaps by some special-purpose hardware accelerators obeying the same unified communication regime as the processors.

For such a platform to be viable it needs to combine the relative simplicity and flexibility of software with the efficacy of hardware and this means executing the software in a parallel fashion. In the ATHOLE project we are *not* concerned with the very challenging (and somewhat hopeless) problem of picking an arbitrary sequential program, identifying its maximal parallelism and finding an optimal or satisfactory schedule. Our starting

¹ A project in the framework of the “pole de competitivite” MINALOGIC with the participation of STM, CEA-LETI, THALES, CWS and VERIMAG.

² Of course we do not ignore the large wave of “parallelism” for scientific and other types of computing started in the 80s and other distributed computing paradigms. However it is still fair to say that sequential code in a natural way to write an algorithm.

point are *stream-processing* applications that are described naturally in a *dataflow* style. Under this paradigm an application is viewed as a block diagram or a network of “filters”, a formalism that makes the dependence and independence between tasks visible and hence “exposes” the inherent parallelism. The body of each such filter can be written as a sequential acyclic program obeying some input-output convention.

In principle, once you have such a description, annotated with execution times for each filter, you can apply your favorite *task graph scheduling* algorithm, but there are some problems that renders the straightforward application of standard scheduling algorithms difficult if not impossible. First, we are dealing here with *data-intensive* applications (video, audio, radio) where communicating and transferring data among filters is sometime more significant and resource-consuming than the computations themselves. What is really needed is a model which exposes the *non-parallelism* in the system, that is, pairs of functional blocks that should run on *the same* processor because of the volume of data they exchange. Suppose we have such a model, say a *task-data graph* with annotation for the amount of data transmitted between successive filters, still we cannot apply a scheduling paradigm which is based on *deterministic* allocation over time of processors and communication channels. The reason for this is that the solution identified as the best communication infra structure is the NoC, whose transmission characteristics are more *bulky* and *statistical* in nature. Hence, unlike in critical real-time systems where determinism and predictability are considered almost sacred, in a NoC-based multiprocessor system one has to find new creative ways to combine traditional scheduling with throughput and load-balancing reasoning.

To be more concrete about what we mean by this latter type of reasoning consider deploying a network of filters on a platform consisting of a network of processors. The problem can be addressed by first partitioning the set of tasks among processors (mapping), trying to obtain a “balanced” partition where computation and communication are evenly distributed over the fabric. Nevertheless, such a global reasoning may, in principle, yield inefficient implementations because applying such reasoning we ignore the temporal aspects of the communication and it may be the case that some communication bottlenecks are manifested.

The ideal scenario for developing this type of applications is the following:

1. The application is written as network of filters;
2. The application is compiled into native code while keeping also a representation of its network structure;
3. This structure is annotated with computation time and communication volume;
4. A model of the architecture (or of several instances of the architecture) is given;
5. A mapping/scheduling utility takes both these descriptions and computes a mapping and a schedule;
6. Additional “glueing” code is added to filters according to the mapping/scheduling decisions.

Crucial to such a tool is the ability to reason in abstract high-level terms about tasks and architecture, abstracting from the actual code and seeing tasks/filters as discrete processes that take *some time* to compute and which produce an amount of data. Reasoning

at this level provides for design-space exploration, that is, checking at early stages of the development whether an application can be realized on a given instance of the architecture, or finding the cheapest architecture on which the application can run. While such exploration methods have been applied to other types of hardware-software platform, they will probably be more effective in the multi-core context where one has to choose among similar architectures that may differ only in the number of processes, the communication topology etc. This level of modeling is covered very nicely by the formalism of *timed automata* and its associated tools such as IF.

For reasons of time and space let us stop here mentioning two more points. First, the multi-core architecture provides for a more structured ways to control power consumption by turning processors on and off, reducing their voltage, etc. Secondly, from a theoretical perspective, the whole topic may suggest new (or rediscover) approaches to parallel computational complexity based on models that treat computation and communication as citizens of the same class.

Acknowledgment: I would like to thank my partners in ATHOLE, in particular Bruno Jego and Gilbert Richard from ST, Francois Bertrand from CEA-LETI and Michel Sarlotte from THALES, from whom I learned a lot, as well as the members of the VERIMAG team in the project, including Marius Bozga, Ramzi Ben Salah, Aldric Degorre, Julien Legriël, Jean-Francois Kempf and Selma Saidi.

References

1. Y. Abdeddaim, E. Asarin and O. Maler Scheduling with Timed Automata , *Theoretical Computer Science* 354, 272-300, 2006.
2. L. Benini, G. De Micheli, Networks on Chip: a New SoC Paradigm, *IEEE Computer* 35, 2002.
3. J.L. Bergerand, P. Caspi, D. Pilaud, N. Halbwegs and E. Pilaud, Outline of a Real Time Data Flow Language, *RTSS'85*, 33-42, 1985.
4. S.S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Press, 1996.
5. M. Bozga, S. Graf and L. Mounier, IF-2.0: A Validation Environment for Component-Based Real-Time Systems, *CAV'02*, LNCS 2404, 343-348, 2002.
6. J.T. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt, Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, *Int. J. of Computer Simulation*, 155-182, 1994.
7. A. Darte, Y. Robert, and F. Vivien. *Scheduling and Automatic Parallelization*. Birkhauser Boston, 2000.
8. A. Degorre and O. Maler, On Scheduling Policies for Streams of Structured Jobs, *FORMATS'08*, 141-154, LNCS 5215, 2008.
9. M. Karczmarek, W. Thies, and S. Amarasinghe, Phased Scheduling of Stream Programs, *LCTES'03*, 2003
10. J. Legriël and O. Maler, Meeting Deadlines Cheaply, VERIMAG Technical Report, 2008.
11. H. El-Rewini, T.G. Lewis and H.H. Ali *Task Scheduling in Parallel and Distributed Systems*, Prentice-Hall, 1994
12. M. Snir, Scalable Parallel Systems: Past, Present and Future (from an IBM Perspective), *3rd International Conference on Massively Parallel Processing Using Optical Interconnections*, 33-35, 1996.