Parallel Programming for Multi-core Architectures

Jean-Marc MOREL [a] for the ParMA Project Partners [b]
[a] *Bull SAS, France*
[b] *France, Germany, Spain, UK*

Abstract. This paper gives an overview of the European ITEA2 project named ParMA ("Parallel Programming for Multi-core Architectures"), which aims at developing advanced technologies for exploiting multi-core architectures both for conventional High Performance Computing (HPC) and for Embedded Computing. It presents the main components of the ParMA technology and describes how application developers applied it and improved the performance of their applications.

Keywords. Keywords. parallel programming, multi-core, performance analysis, code optimization, (embedded) high-performance computing.

1. Introduction

The ParMA project has been proposed in 2006 when it was acknowledged that the "free lunch was over" as Herb Sutter stressed it in his famous article [1]: "because the microprocessor serial processing speed is reaching a physical limit, processor manufacturers will turn en masse to hyperthreading and multi-core architectures, forcing software developers to develop massively multithreaded programs to harness the power of such processors". Indeed, the number of cores per chip doubles every 18 months, instead of clock frequency as shown in Figure 1.

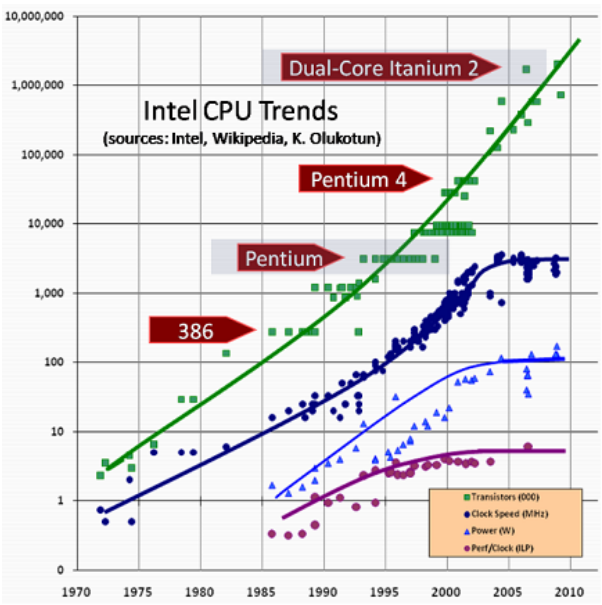


Figure 1.: Intel CPU Introductions

On the other hand, research and industry do need more powerful and more cost effective technical and scientific computing systems either to solve big challenges (e.g. global warming studies, bio-informatics) or to design new products in a more effective manner (e.g. using car crash simulation to spare the cost of hardware prototypes). Efficient parallel computing appears to be paramount to meet these challenges.

Therefore, the ParMA project (see `http://www.parma-itea2.org/`) has been launched mid-2007 to develop and leverage parallel programming techniques and tools enabling to take advantage of multi-core architectures. The aim is to build a software development environment intended, on the one hand, to significantly improve the performance of HPC applications (thus the modelling and simulation of more complex models) and, on the other hand, to prepare to develop power- intensive embedded applications that could not be envisaged so far.

In the following sections, we present the ParMA approach, the main components of the ParMA technology (focusing on the elements that are not presented in the companion papers [2,3,4,5,6,7]), and the first results obtained on industrial applications, while the list of partners with their main contribution is given in Table 1 on the last page.

2. The ParMA Approach

The advent of various types of massively and multilevel parallel architectures presents several real challenges:

- Since most existing HPC applications are designed as a set of independent processes communicating and synchronising through calls to MPI (Message Passing Interface) primitives with each process being bound to a processor, programming paradigms and tools must be developed to help restructure these applications so that they can fully benefit from the multilevel parallelism offered by new architectures;
- Due to the huge number of threads that multi-core processors will be able to run in parallel, dramatic improvements must be achieved in the way threads are allocated and monitored to minimise memory access – i.e. data exchanges between threads;
- Because of the increased complexity resulting from the huge number of threads, more powerful programming, debugging and performance analysis tools are needed . They should also be more easy to use, and should help optimize applications using different forms of parallel architectures in various contexts (e.g. SMP, NUMA, MPSoC).
- Since embedded systems will also be built with multi-core processors, multilevel parallel design, programming and execution models must be defined and tools must be developed accordingly; including tools to design efficient interconnecting networks for MPSoC.
- Because each domain – simulation, virtual reality, avionics, etc. – has its own characteristics and specific constraints, the proposed approaches must be experimented and validated with different applications from various domains.

To meet the above challenges, ParMA proposed a holistic approach and we split the technical work into four inter-connected work packages as shown in Figure 2.

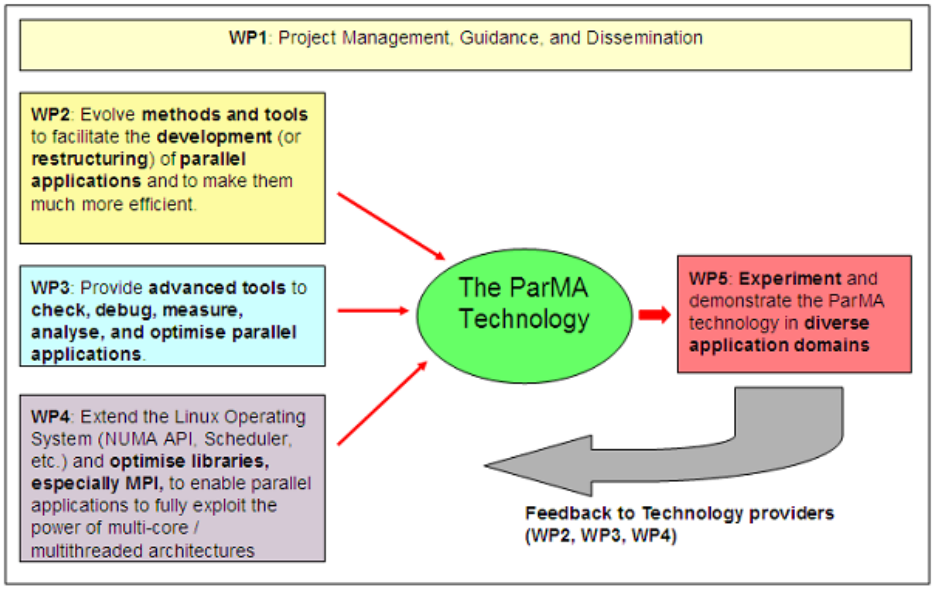


Figure 2. The ParMA cooperation schema

3. The ParMA Technology

As it is acknowledged that there is no silver bullet to port a compute-intensive application on a multi-core architecture and get better performance, several key enablers need to be combined and leveraged that include:

- Algorithms that are more accurate and converge faster
- Methods and tools to detect and extract parallelism (thread extraction),
- New parallel programming models and extended directives to express parallelism,
- Easy to use and scalable performance-analysis tools,
- Powerful correctness checker and debugger,
- Code optimization tools,
- Optimized libraries for multithread / multi-core, and
- Enhanced software infrastructure with optimized thread management, job scheduling, etc.

Thus, the ParMA project organized to advance the state-of-the-art in these various aspects:

3.1. Algorithms that are more accurate and converge faster

Several partners recognized the need to start by identifying critical algorithms that could not take advantage from parallelism or could not scale properly and to replace them by new ones which can well exploit multi-core architectures thereby yielding faster and more accurate results. CEA- LIST for instance, developed innovative algorithms of collision detection (based on Bounding Volume Hierarchy) and object motion management.

They designed and tested a task calibration scheme to get a set of homogeneous tasks (with respect to compute time) matching the number of available cores while limiting the overhead to pay for synchronisation and task management. The main challenge was to organize the tasks (threads) to get optimal load balancing.

In an industrial test case carried out on an 8 core machine with an initial speed-up of 1.7, their extraction parallelism (using a mix of temporal coherency and geometrical properties) enabled to deliver a speed-up of 6.9.

Several other similar experiences have been made that, in particular, led to adopting solvers that are more efficient and more scalable on multi-core architectures.

3.2. Methods and tools to detect and extract parallelism

Methods and tools to detect and extract parallelism are of course much needed to parallelize a sequential application. Therefore CAPS (http://www.caps-entreprise.com/) developed a thread extraction tool that helps identify possible threads using profiling information: hot paths and memory transfers (data size and dependencies). This extraction facility is part of the HMPP tool which enables to map domain specific and critical computations on specialized processors (hardware accelerators such as GPU or FPGA) while generic computations are allocated on generic cores (CPUs). This mapping is specified by mean of OpenMP-like compiler directives that are used to generate codelets that encapsulate hardware-specific code, while determining which data are local or shared to minimize data transfers.

3.3. New parallel programming models and enriched directives to express parallelism

Evolving parallel programming models to help application developers express parallelism to get optimal binary code on multi-core architectures is also an important research path in ParMA. In addition to HMPP directives mentioned above, two extensions have been designed and are being implemented:

- The STEP approach [4] to transform an OpenMP program to an MPI program so that it can be executed on distributed-memory platforms.
- The introduction of data parallelism in the OASIS parallel time-triggered programming and execution model for embedded system, evaluated with an intense-computing application provided by Dassault Aviation [5].

3.4. Easy to use and scalable performance-analysis tools

Easy to use and scalable performance-analysis tools are of the utmost importance to help developers understand how their parallel application behaves on a given multi -core architecture. These tools point out the performance bottlenecks, and help identify how to remove these bottlenecks and more generally how to optimize the code or reduce the time spent in inter- processes communications. See [7] for more details about the performance toolset (Scalasca + Vampir) that has been extended, enhanced, and integrated in ParMA.

3.5. Powerful correctness checker and debugger

Powerful correctness checker and debugger is also a much needed category of tools when dealing with highly parallel code, in particular when this code must execute in various contexts (on different platforms, using different MPI libraries, etc.). Whereas various tools exist that address different problems (for instance DDT from Allinea to debug the code and Marmot from HLRS and TU Dresden to check MPI correctness), it appears

tedious for the user to run them separately. ParMA enabled to efficiently integrate and combine these tools to ease their use. In particular, an "Interface for Integrated MPI Correctness Checking" [6] was designed, allowing to easily combine any MPI correctness checker with performance analysis tools.

3.6. Code optimisation tools

Parallelism enables to distribute the work on different processors but optimal performance cannot be obtained without code optimisation. First, the developer needs facilities to identify which resources are saturated (e.g. the memory bandwidth) and which part of the code should be changed. Then, guidance is needed to find out what should be changed (e.g. the algorithm, the data layout, the directives, or the compiler options) and in which way. UVSQ designed a semi- automated methodology to analyze performance and guide the optimization process [2] using both static analysis (with the MAQAO tool) and dynamic analysis (with Hardware Performance monitoring and memory traces).

3.7. Optimized libraries for multithread / multi-core

As an important part of the executing time is spent in functions from various libraries, for instance the BLAS[1] library that contains routines for performing basic vector and matrix operations, optimizing performance-critical and complex routines such as DGEMM (matrix-matrix multiply) is both challenging and rewarding. So, UVSQ explored the impact of data prefetching on multi-core Xeon platform and devised efficient workarounds that led to an optimized parallel version of two building block functions, `matrix_init` and `memcpy2D` which perform significantly better than the Intel£s implementation. Thus, these functions can be used to replace the corresponding routines of the Intel MKL when there is an important need of them, typically in iterative solvers.

3.8. Optimized software infrastructure

An optimized software infrastructure is more than needed to ensure efficient communications (data exchanges) among interrelated processes and threads that execute concurrently. For example, an optimized implementation of MPI (Message Passing Interface) must be topology- aware in order to use the most efficient way of moving data between two threads depending on their relative position in the hierarchical architecture (an HPC cluster being usually composed of several groups / islands of computing-nodes, each node containing several sockets, each socket containing several cores, and each core being possibly multithreaded). For intra-node communication for instance, Bull designed an efficient shared and cache memory management that minimizes buffer copy and optimizes bandwidth. To be optimal, different protocols depending on the size of the message to be transmitted are implemented. Other optimizations include (1) improved scheduling policies, (2) synchronisation of daemons on all nodes to reduce their disturbances thus improving MPI collective operations, and (3) enhanced resources and job management to ensure that all cores of a cluster are consistently and optimally exploited.

The need for efficient communications also exists in embedded systems where multicore processors are making their way. As traditional bus-based interconnects prove to be insufficient to support the increasing volume of data exchange between the multiple processors on a MPSoC (Multi-Processor System-on-Chip), inter-processors communi-

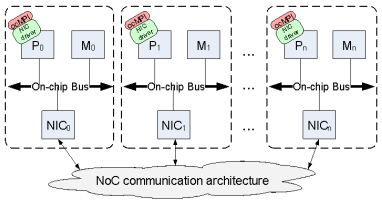
[1] BLAS stands for Basic Linear Algebra Subprograms. It is part of the Intel Math Kernel Library (MKL)

cations in embedded systems should now be based on the Network-on-Chip (NoC) concept. UAB developed the NoCMaker tool that helps NoC developers to design an efficient network for a given embedded system architecture and the set of applications that will run on it. NoCMaker enables to simulate and analyze not only the performance of the designed NoC but also its area and energy consumption because those who embrace the manycore wave also expect superior energy efficiency (MOPS/mW).

Finally, to enable Indra to develop a custom on-chip SDR (Software Defined Radio, an example of power-intensive embedded application), Robotiker designed and implemented a NoC-based MPSoC platform using a FPGA[2].

It has the following components:

– Soft-core IP processors (P_i)
  Xilinx Microblaze soft-core processor
– Distributed memory subsystem (M_i)
– Network Interface Controller (NICi)
– Driver for Network Interface Controller
– NoC communication architecture
– On-Chip Message Passing Interface (ocMPI)



Note that this on-chip message passing interface implements a subset of the MPI standard which has been adapted to the embedded system environment [8].

The embedded system developers also adopted OTF (the Open Trace Format) which has been designed as the common trace format for the performance analysis tools (e.g. Vampir), thereby enabling to also benefit from these tools.

4. First Results Obtained on Industrial Applications

ITEA projects being industry-driven, the validation and uptake of the developed technology by industrial partners is quite important. As such, the ParMA project includes several numerical simulation software editors and several compute-intensive application developers who started to experiment with the initial version of the ParMA tools. They run a series of performance measurements to identify performance bottlenecks, understand the behavior of their code on multi-core architecture, assess its scalability, and when possible compare the performance of MPI versus OpenMP code version. Most often, these results were analyzed with the help of HPC experts from the labs who in addition gave some hints on how the code should be restructured or the compilation options changed to get better performance. Of course several cycles (measure, analyze, optimize the code) are generally needed to obtain satisfactory results. Hereafter are some examples.

CEA-LIST / LSI developed an efficient parallel dose exposure simulation and, using Vampir and Scalasca to identify bottlenecks, succeeded to reach 0.4s response time on a 16-core platform (instead of 5s on a scalar machine), thus enabling user interactivity. However, scalability can be further improved.

GNS GNS restructured the assembly phase of its metal forming simulation application INDEED (INnovative DEEp Drawing) and got improved performance (30%) by minimizing the sequential code in the assembly phase and by a better load balancing.

[2] An FPGA (Field-Programmable Gate Array) is a semiconductor device containing programmable logic components called “logic blocks”, and programmable interconnects.

Besides, the combined use of Scalasca and Vampir enabled GNS to optimize the adaptive mesh refinement and load balancing phases. See more details in [7].

MAGMA had two versions of its MAGMAsolid solver: (1) A SMP version (i.e. for shared-memory architecture only) that did not scale well (speedup limited to 3.5 on 8 or 16 cores) and (2) an MPI version that scaled well on distributed memory architecture with a large number of processors (speedup $\sim$45x using 64 cores) but was about four times slower than the SMP version on a 1- to 8-core shared memory architecture. Thus, MAGMA application developers carried out a detailed performance analysis of the MPI version to get a unique and efficient product. They first identified that they should switch from a CG (Conjugate Gradient) based solver to an ADI (Alternating Direction Implicit) based solver ($\sim$2x faster). Then, they found out that they could improve the cache efficiency by reorganizing the data structures (eliminating 6 temporary arrays) and enable the compiler to better optimize the code by removing function calls in inner loops. Still with the help of the performance analysis tools, MAGMA also obtained 3.5x performance increase for its MAGMAfill module by redesigning central communication routines using asynchronous communication and by optimizing the checkpointing strategy. As a result, MAGMA customers are now provided with a unique scalable solution (2x to 4x faster) on whatever architecture they use.

RECOM's objective in ParMA was to adopt a hybrid parallel programming model, allowing for the combination of distributed memory parallelization across nodes and data parallel execution within the nodes. For this, an OpenMP- and MPI-Implementation, as well as a hybrid parallel version using a combined MPI- and OpenMP-parallel programming model was implemented and installed on the project platform where detailed performance analysis could be carried out using different combinations of MPI processes and OpenMP threads in each MPI process. For instance, using Vampir, RECOM application developers identified and removed a synchronization bottleneck, obtaining a 10% performance improvement. They also studied with the help of the MAQAO tool how to improve memory access as well as memory utilisation for some memory-intense routines, and could achieve performance gains of about 34% for the entire application. See more details in [3].

5. Conclusion and Future Work

In this paper we described the overall approach of the ParMA team to consistently advance the various techniques and tools that are needed to speed-up compute-intensive applications and to get them scalable on modern multi-core architectures. Efficient teamwork between HPC application developers and performance tools providers enabled to find out pr oblems, their cause, and most often a solution that significantly improves the performance of the application, up to 4x faster in some cases. So, from now on, application partners can provide their customers with new, very efficient and scalable solutions. Besides, embedded applications developers started to experiment with parallelization of their code, taking advantage from the experience, the techniques, and the tools from the HPC world: they designed ocMPI after MPI, adopted OTF (Open Trace Format), and can benefit from the performance analysis tools such as Vampir. Our plan is to consolidate these early results and to further disseminate them during the few months before the end of the project.

Acknowledgments

This work has partially been supported by the French Ministry of Industry (DGCIS), the German Ministry for Research and Education (BMBF), and the Spanish Ministry of Industry through the ITEA2 project "ParMA" (No 06015, June 2007 – May 2010).

The ParMa Consortium

Ctry	Partner	Category (Industrial, Lab, University)	Main contribution
FR	Bull	I: HPC platform provider	Provide common HPC platform
FR	CAPS	I: Accelerate HPC code using GPU	Develop thread extraction tool
FR	CEA-LIST	L: Safety real-time embedded system	Extend programming model
FR	Dassault Aviation	I: Aircraft design	Experiment with critical code
FR	IT SudParis	U: Institut Telecom Sud Paris	Transform OpenMP to MPI
FR	UVSQ	U: University of Versailles St-Quentin	Optimize code with MAQAO
DE	GNS	I: Metal forming simulation software	Experiment with INDEED
DE	GWT	I: HPC software tool editor	Develop Vampir with TUD
DE	HLRS	U: University of Stuttgart	Develop MPI correctness tool
DE	JSC	L: Jülich Supercomputing Center	Develop Scalasca
DE	MAGMA	I: Casting process simulation software	Experiment with MAGMASoft
DE	RECOM	I: 3D combustion modeling software	Experiment with AIOLOS
DE	TU Dresden	U: Technische Universität Dresden	Develop Vampir with GWT
ES	INDRA	I: Avionics systems (e.g. SDR)	Experiment with Soft. Radio
ES	Robotiker	L: Embedded Systems	Develop MPSoC platform
ES	UAB	U: Universitat autonoma de Barcelona	Develop NoCMaker tool
UK	Allinea	I: HPC software tools Editor	Develop debugger (DDT)

Table 1. The ParMA consortium

.

References

[1] Sutter, H. 2005. The free lunch is over: A fundamental turn toward concurrency in software, Dr. Dobb's Journal, 30(3).

[2] Andres Charif-Rubial, Souad Koliai, William Jalby, Bettina Krammer, Quang Dinh: An Approach to Application Performance Tuning (Proceedings of Parco 2009).

[3] Benedetto Risio, Alexander Berreth, Stephane Zuckerman, Souad Koliai, Mickael Ivascot, William Jalby, Bettina Krammer, Bernd Mohr, Thomas William: How to Accelerate an Application: a Practical Case Study in Combustion Modelling (Proceedings of Parco 2009).

[4] Daniel Millot, Alain Muller, Christian Parrot, Frédérique Silber-Chaussumier: From OpenMP to MPI: first experiments of the STEP source-to-source transformation (Proceedings of Parco 2009).

[5] Christophe AUSSAGUES, Emmanuel OHAYON, Karine BRIFAULT, Quang V. DINH: Using Multi-Core Architectures to Execute High Performance-Oriented Real-Time Applications (Proceedings of Parco 2009).

[6] Tobias Hilbrich, Matthias Jurenz, Hartmut Mix, Holger Brunst, Andreas Knüpfer, Matthias S. Müller, Wolfgang E. Nagel : An Interface for Integrated MPI Correctness Checking (Proceedings of Parco 2009).

[7] Thomas William, Hartmut Mix, Bernd Mohr, Felix Voigtländer, René Menzel: Enhanced Performance Analysis of Multi-core Applications with an Integrated Tool-chain (Proceedings of Parco 2009).

[8] Jaume Joven, David Castells-Rufas: a lightweight MPI-based programming model and its HW support for NoC-based MPSoC. DATE'09 (Design Automation and Test in Europe), April 2009, Nice, France.