OpenMP performance analysis for many-core platforms with non-uniform memory access

Pablo González de Aledo Marugán[1], Javier González Bayón[1], Pablo Sánchez Espeso[1] and Juan Casal Martín[2]

[1] TEISA, University of Cantabria
Santander, Cantabria 39005, Spain

[2] (SAPEC) Sociedad Anónima de Productos Electrónicos y de Comunicación
Madrid, 28037, Spain

Abstract

One of the first steps in embedded-system design flow is to choose the most efficient implementation of the embedded software application. However, this is difficult to do at the earliest design stages because particular details of the final many-core HW platform are usually unknown and many possible mappings of the software tasks/threads have to be evaluated. This paper presents a complete framework for early performance estimation of parallel programs in many-core platforms. The proposed framework is based on a specific native-simulation approach oriented to many-core platforms, which enables fast simulation and profiling. The software parallelism is specified in OpenMP, a commonly used application software interface (API) for shared-memory parallel programming. In order to support Non-Uniform Memory Access (NUMA) architectures (which are dominant in high-performance many-core platforms), the paper proposes some OpenMP extensions. These extensions improve performance analysis and facilitate the automatic translation from OpenMP to OpenCL (a low-level API for heterogeneous computing), which are commonly used for NUMA programming). Results show that the proposed OpenMP extension and specific parallel modeling techniques provide reliable results even for NUMA architectures.

Keywords: *OpenMP,OpenCL, performance analysis, many-core, NUMA, early estimation.*

1. Introduction

During the last years, embedded computing platforms have been evolving from mono-core architectures to MPSoC (Multi-Processor Systems-on-Chip) and many-core systems (typically, more than 32 cores per chip). These new platforms provide higher computation capabilities with a moderate increase in power consumption. They comply with current requirements of real-time applications such as image-processing, video compression and augmented reality. However, the higher number of cores used in current many-core platforms (for example [1]) exponentially increases the complexity of the design-space exploration process. Additionally, new SoC are replacing hardware accelerators by many-core modules (for example, P2012 many-core IP) that also increase design-space exploration. Therefore, early and faster performance estimation techniques are necessary to guide the embedded-system design flow for many-core architectures. Most many-core architectures group the cores in clusters or sets of processors that share some common local resources. There are two main communication architectures between processing elements in a many-core SoC: bus-based and NoC-based (Network-On-Chip) [2] communication. For inter-cluster communication, NoC is typically used because of its better scalability and power consumption. For intra-cluster communication, a bus is preferred because of its smaller size and better performance. This double hierarchy of communication introduces new problems and challenges in the design of parallel applications because the performances are highly related to how the

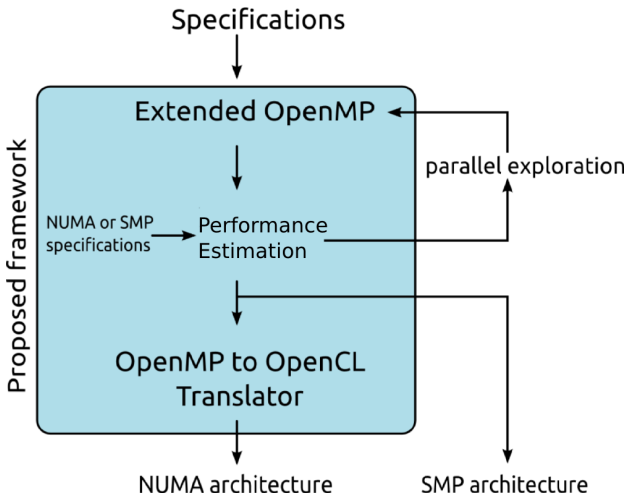

Figure 1: Proposed framework

communication between the cores and the memory is performed. Moreover, memory hierarchy is becoming the main bottleneck of many-core architectures, so it is important to model it accurately. Memory-coherence between each core cache and the main memory, as in ccNUMA architectures [2], is no longer efficient when many cores are transferring data though a NoC. Therefore, caches are sometimes replaced by scratch-pad memories and the responsibility for keeping these memories coherent is delegated to the application program. A profound knowledge of both the HW architecture (communication latency, bandwidth, number of memory banks, etc.) and the application memory requirements (most relevant variables and memory access patterns) is normally required to efficiently implement applications in many-core platforms. This task normally requires efficient, accurate and fast performance analysis frameworks.

One of the simplest target-independent paradigms for parallelizing a sequential code or creating a parallel program is OpenMP. It provides a multithreaded application implementation in a shared memory environment. Moreover, it has the advantage that it is easily portable and enforces gradual transformations of the code. There are other parallel programming paradigms, such as OpenCL, but the programmer must explicitly define the code parallelization and the communication between concurrent elements. This detailed specification can slow down the design process at the early stages of the design flow.

One of the biggest problems of application mapping in many-core platforms is the large number of possible task/thread bindings. Therefore, a framework is necessary to model the structure and resources of many-core platforms including their complex NUMA memory hierarchy. The framework would also enable a rapid simulation and application performance analysis at the early stages of the design process. It is also desirable that OpenMP can be used to program NUMA architectures in an efficient and reliable way.

The framework presented in this work aims to facilitate the

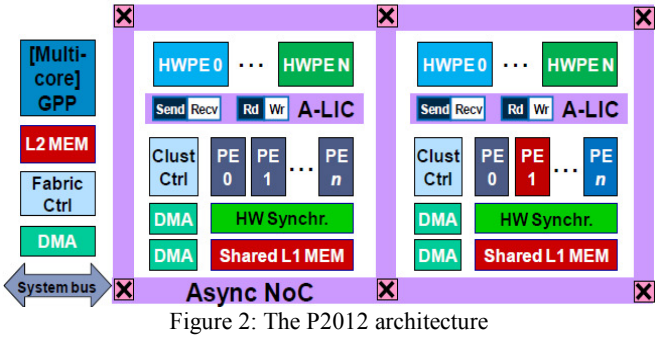

Figure 2: The P2012 architecture

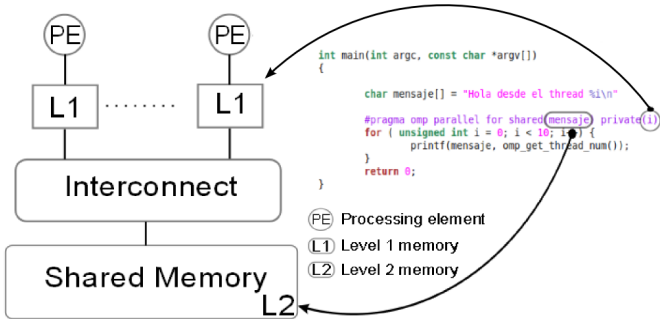

Figure 3: Inference of location based on semantic meaning

design process for a many-core platform allowing rapid simulations of different parallel configurations. This framework is shown in Figure 1 and it integrates performance analysis (with improved memory models) and source-to-source code transformation tools. The contributions of the paper can be summarized as follows:

- It proposes some extensions to the OpenMP standard (which is oriented to Symmetric Multiprocessing architectures, SMP) to deal with NUMA architectures. It also proposes the use of OpenMP extension to provide information about the virtual localization of variables in the memory hierarchy.
- It introduces a performance analysis tool that is based on native simulation and is able to simulate OpenMP applications on embedded systems targeting many-core NUMA platforms. It provides fast and sufficiently accurate results to enable a thorough exploration of many different parallelizations.
- It introduces a source-to-source transformation tool that provides automatic OpenMP to OpenCL program translation. This is useful because it enables the design space in OpenMP to be explored (which is more amenable to trying different program parallelizations), and it also automates the error-prone translation when the final parallelization have been established.

The paper is organized as follows. In Section II, the state of the art is analyzed. In section III, the ST-HORM platform (in which tests will be performed) is presented. The proposed framework is presented in Section IV and some simulation results are shown in Section V. Finally, the conclusions are drawn in Section VI.

2. State of the art

Nowadays, some simulation and performance analysis frameworks use ISSs (Instruction Set Simulators) for evaluation of applications that are executed in multi or

many-core platforms. The work in [4] introduces a simulation framework for multi-cores based on a cycle-accurate simulator at ISS level. In [5], the authors introduce GePoP, an ISS-based simulation framework oriented to the P2012 platform (many-core NUMA architecture). This tool only works for a specific architecture and its adaptation to other platforms requires a great effort (a new ISS has to be developed).

Virtual platforms based on dynamic binary translation, as in [6][7], can also be used to simulate many-core systems. However, this virtual platform approach does not enable reliable design space exploration or performance profiling.

Jung et al. [8] developed a technique that allows the modeling of multi-core processors at thread-level, overlooking instruction-level and micro-architectural details. This idea is similar to our proposal, although we present specific details about the implementation of the simulation framework. In [8], only the instructions that cause the threads to stall or interchange data with other threads are considered in detail. The other instructions are simulated by means of native performance estimation. In [9], the authors present a technique that automatically measures the performance of arbitrary parts of a program on a multiprocessor embedded system. With this framework only certain parts of the code can be analyzed, so the programmer must have a thorough knowledge of the application to profile the most relevant portions of the code. In [10], an initial version of an automatic scheduler on heterogeneous architectures for task-to-processors assignment and a model of parallel algorithms on heterogeneous architectures are presented.

It is well known that OpenMP is a very suitable option for fine parallelization algorithms as was proved in [11]. Several works present extensions of OpenMP to allow the translation of programs to NUMA architectures. For example, the work in [10] proposes the first source-to-source compiler that is able to automatically convert from OpenMP to CUDA. The work proposed in [13] uses an OMPi parser in order to generate transformed code that runs over a CUDA runtime. OMPi [14] is a source-to-source compiler that accepts the OpenMP standard. It implements parallelism over the POSIX pthreads library, but it has not been designed with other libraries in mind. The work in [15] extends OpenMP to enable its use in NUMA machines although it is oriented to FORTRAN programs. All these translators are aimed at code conversion, but they do not provide the specification infrastructure that is needed to allow rapid explorations of different parallel options and memory configurations. Furthermore, even though they support CUDA as a programming API for NUMA architectures, they do not enable conversion to OpenCL. The proposed tool can perform this translation while providing NUMA architectures with accurate estimations.

3. The P2012 (ST-HORM) platform

As was previously commented, the proposed tool aims to facilitate the SW application implementation in a generic many-core platform. In order to compare results, a specific many-core platform has been selected. The target is the ST-HORM (P2012) **[1]** platform, which is currently under joint development by STMicroelectronics and CEA. The P2012 computing fabric is highly modular, as it is based on multiple clusters implemented with independent power and clock domains. As depicted in Figure 2, clusters are connected via a high-performance fully-asynchronous network-on-chip (NoC), which provides scalable bandwidth and robust communication across different power and clock domains [1].

Each cluster features up to 16 tightly-coupled processing elements (PE) sharing uncached multi-banked level-1 data memories (L1 memory), individual cached instruction memories, a multi-channel advanced DMA engine, and specialized hardware for synchronization and scheduling acceleration. Each PE is a customizable 32-bit RISC processor which could include vector units, a floating-point unit and special-purpose instructions.

From the software viewpoint, all processors in the P2012 have full visibility of all the memories (shared memory architecture) with no aliasing: hence, it is possible for the processors in one cluster to load and store directly in remote L1 memories in other clusters. The same holds for global memory (L2 memory) and external-host memory. A relaxed memory consistency model is hardware-supported through memory barrier instructions. Synchronous and asynchronous DMA-assisted memory copy functions are the favored way to hide the latency in accessing remote memories.

A key aspect to achieve performance gain in an application running on the P2012 platform is efficient management of accesses to the global memory. These are performed without any cache management, so they are subject to high latencies. It is the responsibility of the programmer to ensure that the variables are close to the PE that uses them most, so no high latencies are incurred when accessing these variables. The proposed framework can help in this task by enabling the evaluation of different memory access patterns.

4. Proposed framework for early estimation in parallel applications

The framework shown in Figure 1 is described in detailin this section. It is composed of three main parts: specification with an OpenMP extension, performance analysis and code transformation. OpenMP has been extended with several “pragmas” (specific compiler

directives that are inserted into the source code) that provide information that is necessary to efficiently map applications in NUMA architectures. Additionally, a virtual platform based on native simulation provides rapid estimation of parallel program performance to guide the application-to-core mapping. Finally, the OpenMP to OpenCL translator enables the application to be ported to different many-core platforms and software development environments.

4.1 OpenMP extension

As was previously mentioned, one of the most important aspects to consider when creating applications for many-core platforms is the efficient use of the hierarchical memory levels and DMAs (see P2012 section). However, a standardized API to specify DMA transfers or to assign specific data localization for variables is not available yet. This paper proposes an approach that uses the standard "shared" and "private" pragmas of OpenMP to assign every variable to a particular memory in the memory hierarchy. Basically, the proposed approach adds an additional semantic to the standard OpenMP pragmas. For many-core platforms, the proposed approach assumes that "shared" variables are located in memory L2 while "private" variables are located in the cluster memory L1 (see Figure 3). This provides a generic and practical way to assign variables in different platforms which facilitates performance analysis. In order to assign location to a task in micros and clusters, which is also necessary to estimate data movement, a double hierarchy of openMP pragmas is used. The first level of parallelism distributes tasks among clusters, while the second one distributes tasks inside each cluster (see Figure 4).

In addition, specific pragmas have been created to enable the translation from OpenMP to OpenCL and improve performance analysis. It is important to remember that while OpenMP is focused on SMP architectures, OpenCL is focused on NUMA architectures, so information about task and data location have to be provided. The new pragmas are:

- "**#pragma begin kernel** <kernel_name>": This statement indicates that the following section of the code will be executed in a processing element. This section is estimated with the model of the core of the processing element.
- "**#pragma begin kernel call** <kernel_name>": This statement is replaced by an OpenCL kernel call. In terms of performance estimation, it is replaced by a time annotation that models the execution time of the OpenCL kernel create function
- "**#pragma OpenCL setup** <kernel_name>": It is replaced by the OpenCL function that loads the kernel in the processing elements.
- **"#pragma begin DMA transaction** <source_line> <source_row> <dest_x> <dest_y> <linelength> <linewidth>": In OpenMP DMAs are modeled as parallel tasks which perform the data translation between different memories. The current version of the translator does not infer the DMA from these parallel tasks –a DMA can be described in many different ways–, so a pragma is needed to specify these memory transactions.
- "**#pragma OpenCL release**": When the program is finished, this releases the useless OpenCL object and

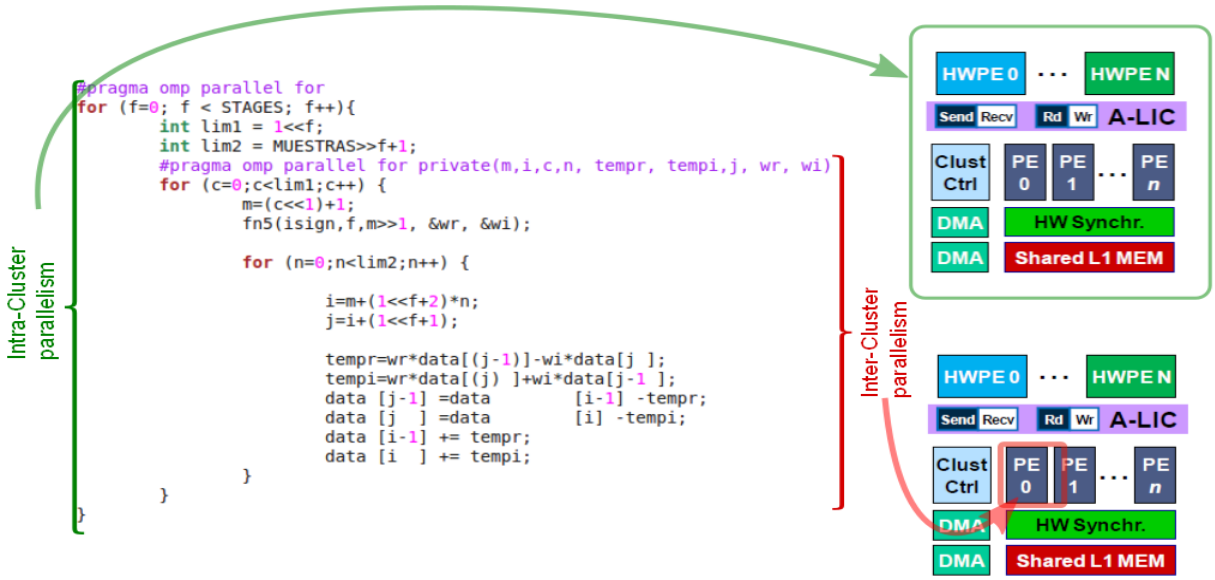

Figure 4: Mapping from threads to clusters and microcontrollers

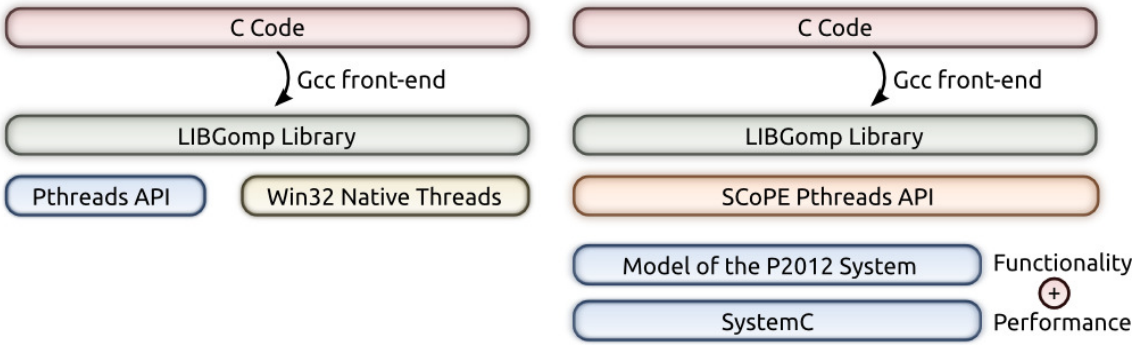

Figure 5: Parallelism modeling in the virtual Platform

frees allocated buffers for DMA transactions.

4.2 Performance analysis infrastructure for many-core platforms

In this work, a virtual platform based on native simulation [16] is used to provide fast and accurate performance estimation of many-core HW/SW systems. In native simulation, the application source code is annotated with performance-oriented code that is target-platform dependent [16]. During execution, the annotated code enables the estimation of the power consumption and application-code execution time in the target many-core platform. The annotated code is compiled with a native compiler of the host computer in which the simulation (annotated code execution) is performed. Thus, cross-compilation is not needed in native-simulation-based virtual platforms. Additionally, these platforms allow the embedded software development process to be started even before the HW platform is completely defined because a limited number of high-level HW-platform parameters are needed. In the current version of the tool, it is possible to model the following elements of many-core architectures:

- Processor: it is characterized by defining the number of cycles that each instruction consumes. Additionally, power consumption could also be assigned to every statement.
- Bus interconnection: it is modeled by a SystemC generic bus model.
- Memories: the model of the memory hierarchy parameterizes memory response delay and memory size. Both data and instruction caches can be simulated, although data caches are not used in the P2012 many-core virtual platform, as the physical platform does not include them (see Section III).
- NoC: The tool provides two simulation approaches for the NoC. The first approach provides a high-level SystemC model where NoC transactions are modeled as blocks of data that flow from one node to another towards a virtual path. The second approach provides a low-level cycle-accurate simulation in which all the micro-architectural details of the router crossbar and switches are modeled. A unique network interface is provided for both approaches and a designer can use the former (if fast simulations are the main goal) or the latter (if simulation accuracy is the main goal) approach.

4.3 Parallelism modeling in the virtual platform

OpenMP is an application program interface that provides concurrency support. During compilation, some additional function calls are inserted by the compiler to create and manage concurrent threads. These additional functions are implemented in the "LibGOMP" library: the OpenMP runtime library. The proposed tool integrates a modified version of the "LibGOMP" library that enables the simulation of thread management functions while analyzing system performance. Therefore, when the "LibGOMP" library calls the thread "create function", there is another call to the function that implements this behavior in the virtual platform. This function keeps track of the execution time that is needed for each thread creation in the target platform (see figure 5). The performance estimation framework is also able to convert OpenMP directives to internal functions calls. It also uses its internal simulator to map the tasks to particular processors in the cores of the target platform. Finally, the parallel platform model is able to consider different costs for different memory transactions.

4.4 Translation of OpenMP to OpenCL

During the last years, several programming APIs have been developed to program multi- and many-core platforms, OpenMP and OpenCL being the most generally used. OpenMP has evolved towards SMP architectures, in which all processor cores access the same shared memory with a similar latency for any data. On the other hand, OpenCL is a framework for building parallel applications

Table 1: Simulation results (proposed framework vs GePoP)

Number of cores	Matrix multiplication	Image Difference Collaborative	FFT Correlator	1D-FFT	Warp	Mpeg predictor
1	30%	8.5%	-	-	-	-
2	10%	12%	-	-	-	-
4	6.7%	4.4%	-	-	-	-
8	18%	0.5%	-	-	-	-
16	28%	2.1%	7.4%	9.1%	6.8%	8.6%

that are portable across heterogeneous parallel platforms. It provides a common hardware abstraction layer across different many-core architectures. OpenCL defines its own memory hierarchy and memory consistency model and is more oriented to NUMA architectures where the memory access time depends on the memory location.

The last component in the proposed framework is an OpenMP-to-OpenCL translator. OpenMP has been widely used so there are a lot of libraries and application implementations that use this API. The translator enables these applications to be ported to OpenCL. One of the main problems of this translation is related to memory access. Even when the platform architecture is transparent for the OpenCL API, the memory model remains exposed to the programmer, who must explicitly program DMA transactions in order to provide data to the OpenCL kernel. Even though these two programming paradigms have evolved towards such different platforms, the fork-join model of OpenMP can also fit any many-core architecture if each thread is assigned to a core. Taking these elements into account, there are two important issues in the OpenMP-OpenCL translation. One is the workload partitioning. Each interaction of an OpenMP program must be associated with an OpenCL thread. The other issue is how to perform the data mapping. The information that can be inferred in OpenMP pragmas about data allocation must be converted to explicit OpenCL DMA function calls between the CPU and the other heterogeneous processors. These issues are deal with in several steps by the proposed translator:

1. OpenCL kernels need to be identified. This is performed taking into account parallel OpenMP sections. When this operation is done, a mapping needs to be done to provide every OpenMP task with a physical location (specifically, the cluster and microcontroller coordinates). For this purpose, the OpenMP nested directive is used, enabling two levels of parallelism in nested loops. This was shown in Figure 4.
2. Variables need to be allocated in memory. To accomplish this, shared and private pragmas are used as commented in section A.
3. DMA transfer implementation. Before and after a kernel call, DMA transactions have to be introduced in the code. "Pragma DMA transaction" is provided as a way to indicate to the tool that a DMA transaction needs to translate data from the host controller and the processing elements.
4. The kernel calling piece of code is created. This code is responsible for calling the kernel, distributing its code among a set of microcontrollers.

5. Simulation results

The benchmark set includes a battery of tests that are used for GePoP validation such as a "matrix multiplication" (it performs a multiplication of two 16*16 matrices with a divide-and-conquer algorithm), and "Image Difference Collaborative" (it obtains the difference between two images). Other examples (that are not included in the GePoP SDK) are obtained from well-known software applications such as "FFT-Correlator" (it obtains the correlation of a template with an image by means of its FFT-transforms), "1-D FFT" (it performs a radix-2 FFT with a parallelized Cooley-Turkey algorithm) and "WARP" (it calculates a geometrical transformation of an image) and an example of an industrial design, an MPEG motion estimation algorithm.

The OpenMP code of these examples has been analyzed with the proposed performance analysis framework. Additionally, the same code was translated to OpenCL, using the proposed source-code translator. The OpenCL code is then simulated with the GePoP simulator, which is provided with the platform SDK. GePoP is based on an ISS simulator. However, it is not currently possible to change the number of cores in GePoP. This requires the modification of all the software layers in order to take into account the real number of cores, although for the two GePoP examples (matrix multiplication and image difference collaborative), it is possible to change the number of threads that can work since they do not enqueue several work-groups at the same time.

The results obtained are presented in Table 1. As can be observed, the proposed framework provides simulation

results with errors lower than 30% and with an average error of 12%. On average, the speedup archived with the proposed framework, compared to GePoP for the proposed examples is 16. Since GePoP is based on the ISS approach, it has a fixed simulation time even when different cores are used (idle cores are simulated even if they do nothing). In contrast, the proposed framework is able to reduce its simulation time depending on the number of cores. In this case, the speedup obtained is one order of magnitude for a multi-core architecture.

6. Conclusions

This paper presents a virtual platform that enables a reliable design-space exploration for many-core platforms. There are three important elements in the proposed tool that facilitate the design process. One is the use of OpenMP as a programming paradigm and the extension of this API with new pragmas that allow a thorough memory exploration. Another contribution is the development of a virtual platform that enables performance evaluation and code simulation before a real hardware prototype is available. Finally, a translator from OpenMP to OpenCL allows applications for shared memory environments to be ported to NUMA architectures.
The proposed framework has been tested with a battery of tests that involve complex operations with some degree of parallelism. It can be observed that the tool is able to capture and estimate the most relevant aspects of the execution of these examples, helping the designer task by providing rapid estimations of the performance of the final system when neither the program, nor the platform are as yet completely finished and available.

Acknowledgments

This work has been funded by CA-104 COBRA (MITyC TSI-020400-2010-82) and DREAMS (TEC2001-28666-C04-02)

References

[1] Platform 2012: A Many-core Programmable Accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology. CEA, STMicroelectronics, Nov. 2010.

[2] Eduard Fernandez-Alonso, David Castells-Rufas, Jaume Joven and Jordi Carrabina, “Survey of NoC and Programming Models Proposals for MPSoC”, International Journal of Computer Science Issues (IJCSI); March 2012, Vol. 9 Issue 2, p176.

[3] Ribeiro, C.P, Mehaut, J.-F and Carissimi, A., “Memory Affinity Management for Numerical Scientific Applications over Multi-core Multiprocessor with Hierarchical Memory”, IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum, 2010, pp. 1-4.

[4] Uehara, K., Sato, S., Miyoshi, T., Kise, K, “A study of an infrastructure for research and development of many-core processors”, International Conference on Parallel and Distributed Computing, Applications and Technologies, 2009, pp. 414-419.

[5] E. Gebrewahid, Zain-ul-Abdin, and B. Svensson, “Mapping Occam-pi programs to a Many-core Architecture”, Fourth Swedish Workshop on Multicore Computing, 2011.

[6] G. Wang, “Power analysis and optimizations for GPU architecture using a power simulator”, ICACTE International Conference on Advanced Computer Theory and Engineering, 2010, pp. 1619-1623.

[7] O. Almer, I. Böhm, T. E. Von Koch, B. Franke, S. Kyle, V. Seeker, C. Thompson and N. Topham, “Scalable multi-core simulation using parallel dynamic binary translation” , International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2011, pp. 190-199.

[8] H. Jung, M. Ju, H. Che, “A theoretical framework for design space exploration of many-core processors”, IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2011, pp. 117-125.

[9] M. Lattuada, C. Pilato, A. Tumeo and F. Ferrandi, “Performance modeling of parallel applications on MPSoCs”, International Symposium on System-on-Chip, 2009, pp. 64-67.

[10] Laura De Giusti, Franco Chichizola, Marcelo Naiouf, Armando De Giusti, Emilio Luque, “Automatic Mapping Tasks to Cores - Evaluating AMTHA Algorithm in Multicore Architectures”, International Journal of Computer Science Issues (IJCSI); March 2010, Vol. 7 Issue 2, p102.

[11] Saravanan, Vijayalakshmi; Radhakrishnan, Mohan; Basavesh, A. S.; Kothari, D. P., “A Comparative Study on Performance Benefits of Multi-core CPUs using OpenMP”, International Journal of Computer Science Issues (IJCSI);Jan2012, Vol. 9 Issue 1, p272.

[12] S. Lee, S.-J. Min, R. Eigenmann, “OpenMP to GPGPU: A compiler framework for automatic translation and optimization.” ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2009.

[13] G. Noaje, C. Jaillet and M. Krajecki, “Source-to-source code translator: OpenMP C to CUDA”, IEEE International Workshop on FTDCS, 2011, pp. 512-519.

[14] V.V. Dimakopoulos and Alkis Georgopoulos, “The OMPi OpenMP/C Compiler”, in Proc. PCI2005, 10th Panhellenic Conference on Informatics, Volos, Greece, Nov. 2005, pp. 153—162

[15] Bircsak J., Craig P., Crowell R., Cvetanovic Z., Harris J., Alexander C., Offner C., "Extending OpenMP for NUMA machines", Proceedings of the 2000 ACM/IEEE,IEEE Computer Society Washington, DC, USA 2000

[16] D. Calvo, P. González, H. Posadas, P. Sánchez, E. Villar, Andrea Acquaviva, Enrico Macii, Claudio Parrella, Mateo Giaconia "SCoPE: SystemC Cosimulation and Performance Estimation. Application to Power and Thermal-Aware Design". University Booth, DATE 11, Grenoble. 2011-03

Pablo González de Aledo studied Telecomunications Engineering in the University of Cantabria and finished his studies in the Network-on-Chip team in ST-Microelectronics Grenoble. In

2012 he obtained a research grant from the Ministry of Education and he is doing research in the Microelectronics Engineering Group improving the modeling and simulation of high-performance, multi-core and heterogeneous platforms.

Javier González-Bayón was born in Santander, Spain, in 1979. He received the Telecommunications Engineering degree from the University of Cantabria (UC) in 2004. He obtained his Ph. D. in the Dept. of Electronic Engineering at the Universidad Politécnica de Madrid in 2011. He is currently working at the Microelectronics Engineering Group in the UC. His main research interests include performance estimation of parallelized programs in shared-memory many-core platforms.

Pablo Sánchez received the Ph.D. in Physics (Electronics) from the University of Cantabria, Santander, Spain, in 1991. He is currently an Associate Professor of Electronic Technology with the Department of Electronic Technology, Engineering and Systems of the University of Cantabria. He is the author of more than 60 international papers and project leader of several ITEA, ARTEMIS, ENIAC and CATRENE projects. His current research interests include Real-time Image Processing and Embedded Systems Design and Verification Methodologies.

Juan Casal has a Technical Telecomunications Engineering degree and a Technical IT Engineering degree. He has worked in signal processing applications for the last 13 years. He has been working in SAPEC since 2002, from 2006 as Project Manager, leading the video encoding algorithm development team. He has represented SAPEC in many Spanish and European R&D projects. He has participated in the development of most of the equipment included in SAPEC's current product portfolio.