Chapter 6
Multicore and Manycore: Hybrid Computing Architectures and Applications

Pedro Valero-Lara
University of Manchester, UK

Manuel Prieto-Matías
Universidad Complutense de Madrid, Spain

Abel Paz-Gallardo
Research Center for Energy, Environment and Technology, Spain

Alfredo Pinelli
City London University, UK

Erich L Foster
Università della Svizzera Italiana, Italy

Johan Jansson
Basque Center for Applied Mathematics, Spain

ABSTRACT

This chapter presents an overview of the evolution of computer architecture, giving special attention on those advances which have promoted the current hybrid systems. After that, we focus on the most extended programming strategies applied to hybrid computing. In fact, programming is one of the most important challenges for this kind of systems, as it requires a high knowledge of the hardware to achieve a good performance. Current hybrid systems are basically composed by three components, two processors (multicore and manycore) and an interconnection bus (PCIe), which connects both processors. Each of the components must be managed differently. After presenting the particular features of current hybrid systems, authors focus on introducing some approaches to exploit simultaneously each of the components. Finally, to clarify how to program in these platforms, two cases studies are presented and analyzed in deep. At the end of the chapter, authors outline the main insights behind hybrid computing and introduce upcoming advances in hybrid systems.

INTRODUCTION

"Moore's Law" (Moore, 1965) is probably the most cited prediction in the computing era. It was in 1965, when one of Intel's co-founders, Gordon Moore, realized that, with the advances achieved in the integration technology, it was economically to double the number of transistors per chip every 18

DOI: 10.4018/978-1-5225-0287-6.ch006

months. This prediction will possibly reach a limit someday, as (Krauss, & Starkman, 2004) stated, due to the *Bekenstein bound*, but as (The International Technology Roadmap for Semiconductors, 2013) confirmed, it is expected to hold true for the next few years.

Nothing about processing performance was said by Gordon Moore when he made his prediction. His observation only links integration levels to production cost (Själander, Martonosi, & Kaxiras, 2014). However, a few years later (Dennard, Gaensslen, Rideout, Bassous, & LeBlanc, 1974) from IBM, articulated a set of rules (i.e., Dennard scaling) that link transistor size with performance and power. The key observation they made was that smaller transistors can switch quickly at lower supply voltages, resulting in more power-efficient circuits and keeping the power density constant (Själander, Martonosi, & Kaxiras, 2014). For about four decades, Moore's law coupled with Dennard scaling have prescribed that every technology generation have more transistors that are, not only smaller, but are also much faster and more energy efficient. However, the transistor scale has also brought new challenges. In fact, with the introduction of the 45 nm node, the insulation became too thin, and power leakage has been a huge problem ever since.

Computer architects have been able to increase the performance of processors, even without increasing their area and power. Indeed, during the 80's and early 90's, actual processor performance increased faster than Moore's law and Dennard scaling predicted (Hennesy, & Patterson, 2011). The surplus of transistors was used by computer architects to integrate complex techniques to hide memory access latency and extract instruction level parallelism (ILP). Some relevant examples were out-of-order execution, branch prediction and speculative execution, register renaming, non-blocking caches or memory dependence prediction. Notoriously, all of them were able to improve performance while maintaining the conventional Von Neumann computational model. In other words, these techniques were virtually invisible to software, avoiding the need to rewrite the applications and letting them improve their performance as technology scaled (Hennesy, & Patterson, 2011).

The breakdown of Dennard scaling prompted the switch to multicore and multithreaded architectures, which has been the driving force in computational technology over the last decade. Broadly speaking, the semiconductor industry has abandoned complex cores in favor of integrating more cores on the same chip (Geer, 2005). Further, hardware multithreading has become essential to mask long-latency operations such as main memory accesses (Nemirovsky, & Tullsen, 2013). The assumption of this new paradigm is that as the number of processors/threads on a chip doubles, the performance of a scalable parallel program will also continue to improve. However, there is now a major problem at the software level. In contrast to previous generations, programmers are in charge of exposing the parallelism in their applications and need it to improve performance; this is not a simple task. Despite more than 40 years' experience with parallel computers, we know that parallel programs are usually difficult to design, implement and debug, and their performance do not always scale well with the number of cores/threads. Despite Amdahl law (Amdahl, 1967) introduces several inconveniences to efficiently extract sufficient parallelism from many applications, the Gustafson law (Gustafson, 1988) defends that any application can be reformulated to take advantage of a higher number of processors. However, the reformulation or porting of a particular application (programming) to a new and bigger computer system can become a non-trivial task.

The first general-purpose processor that included multiple processing cores on the same die was released by IBM in 2001, the IBM POWER4 processor (Olukotun, 2007). Since then, multi-core processors have become the norm. In fact, the major way to improve the performance of high-end processors has been to add support for more threads, either by increasing the number of cores per chip, or through