Overview of Performance Measurement and Analytical Modeling Techniques for Multi-core Processors

Garrison Prinslow, gprinslow@gmail.com (A paper written under the guidance of Prof. Raj Jain)

Download

Abstract:

Multi-core processors can offer significant performance improvements over their single-core counterparts for certain kinds of parallelized tasks, often demanding new programming paradigms to efficiently utilize the complex architecture involved. Multi-core processors also present unique challenges for the performance analyst because the architecture of these processors has significant impacts on the way work is scheduled, memory is allocated, and instructions are executed. When studying the performance of a particular application, the analyst will need a set of techniques to appropriately measure and analytically model the performance of a multi-core processor. This paper provides an introductory overview to multi-core processors, multi-core processor parallelism, performance measurement, and analytical modeling techniques, focusing on multi-core Central Processing Units (CPUs).

Keywords: Multi-core processors, multi-core CPUs, performance measurement, performance modeling and analysis, benchmarking, parallelism, gprof, TAU, profiling.

Table of Contents:

1. Introduction to Multi-Core Processors

Understanding the behavior and architecture of a multi-core processor is necessary to proficiently analyze its performance [Chandramowlishwaran10]. This section defines a multi-core processor, introduces the primary types of multi-core processors used in computing, then compares two of the most prevalent types used in performance computing contexts. The next section introduces three types of parallelism that impact multi-core processor performance. Subsequent sections apply these concepts in a performance analysis context, introducing techniques for performance measurement and analytical modeling. This paper assumes that the analyst is examining the performance of a given application on a multi-core processor, or different types of multi-core processors, seeking to quantify and understand why certain performance

characteristics are observed.

1.1 Definition

This subsection defines a multi-core processor in general terms, discusses why multi-core processors emerged in the mainstream marketplace, and provides examples of common multi-core processors. Accordingly, this will provide the analyst with sufficient background to identify multi-core processors in a system before analyzing the appropriate performance implications.

To define a multi-core processor, a definition of a processor (or "microprocessor") is necessary as well. In computing terms, a processor is a component that reads and executes program instructions; these instructions tell the processor what to do, such as reading data from memory or sending data to an output bus [Microprocessor]. A common type of processor is the Central Processing Unit (CPU). A multi-core processor is generally defined as an integrated circuit to which two or more independent processors (called cores) are attached [Multi-core processor]. This term is distinct from but related to the term multi-CPU, which refers to having multiple CPUs which are not attached to the same integrated circuit [Multi-core processor]. The term uniprocessor generally refers to having one processor per system [Uniprocessor], and that the processor has one core [Keckler09]; it is used to contrast with multiprocessing architectures, i.e. either multi-core, multi-CPU, or both. Figure 1, below, illustrates the basic components of a generic multi-core processor.

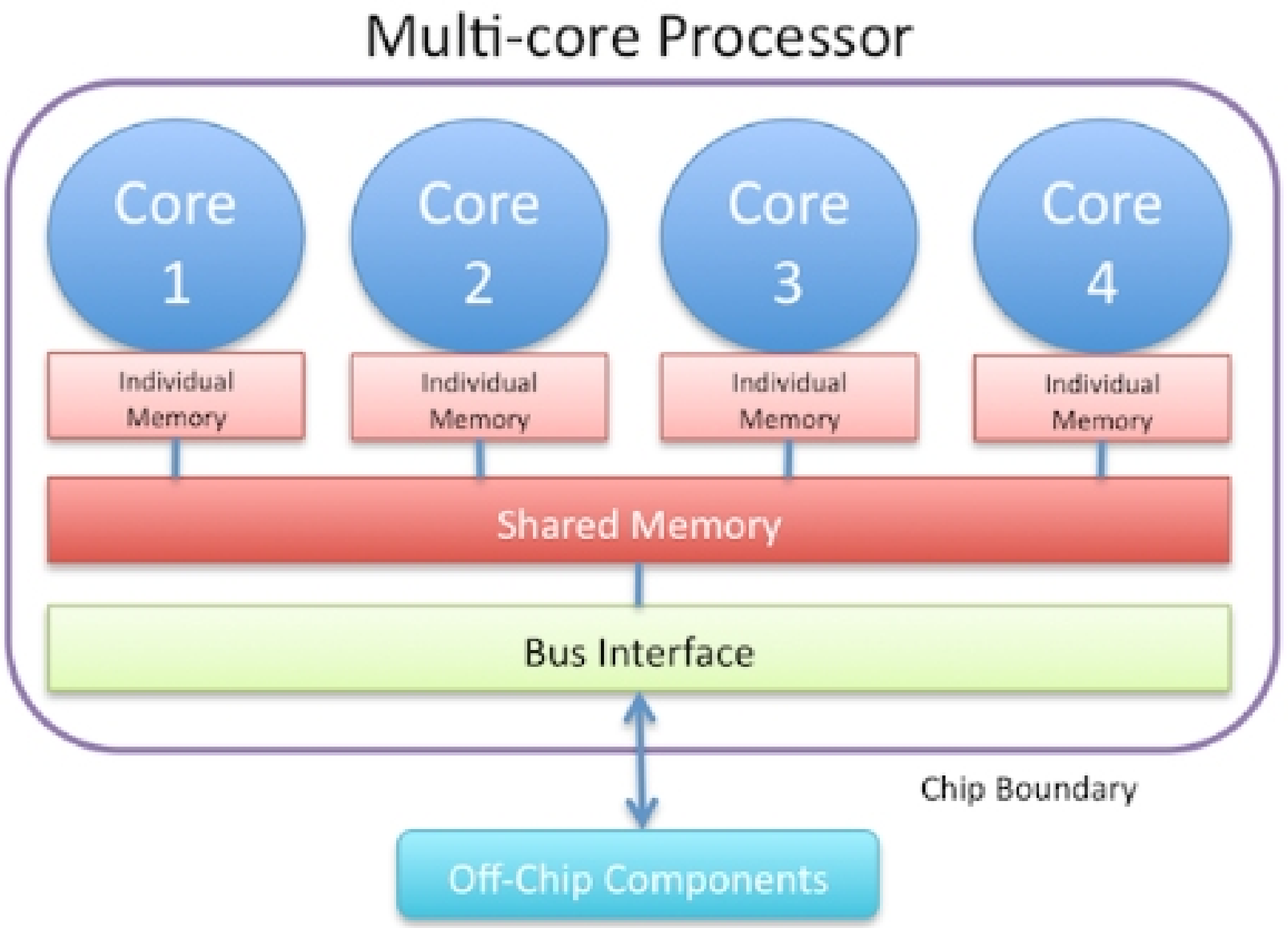

Figure 1: A basic block diagram of a generic multi-core processor

Multi-core processors emerged in the computing industry from uniprocessor technology as a method to achieve greater performance through parallelism rather than raw clock speed. Over the last 30 years, the computer industry developed faster and faster uniprocessors, though this pursuit is drawing to a close due to the limits of transistor scaling, power requirements, and heat dissipation [Keckler09]. Because single-threaded cores are reaching a plateau of clock frequency, chip manufacturers have turned to multi-core processors to enhance performance using parallelism [Keckler09].

1.2 Examples

So far this paper has discussed multi-core processors in a generic sense, as there are many specific types of multi-core processors serving varying functions in computing. Though it is common to refer to multi-core CPUs, other examples include Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs).

Modern GPUs have hundreds of cores, though these cores are significantly different from a CPU core. For example,

NVIDIA's GeForce GTX 580 has 512 Compute Unified Device Architecture (CUDA) cores operating at 1544 MHz [NVIDIA:GTX 580]. AMD's competing Radeon HD 6970 has 1536 stream processors operating at 880 MHz [AMD:HD 6970]. GPUs are primarily designed to accelerate 3D graphics, but starting as early as 1999, computer scientists and researchers recognized that the relatively high floating point performance in GPUs could be used to accelerate other scientific computational applications [NVIDIA:GPU Computing]. The term General Purpose computing/computation on GPUs (GPGPU) now refers to the use of GPUs for other applications besides traditional 3D graphics [Zibula09].

FPGAs might also be considered multi-core processors because this specialized hardware may be programmed to employ multiple "soft" microprocessor cores on one FPGA board [Multi-core processor]. The flexibility of reconfigurable hardware allows the FPGA board's configuration to be optimized for individual applications, rather than a traditional CPU which must have a universal instruction set. An example of an advanced multi-core CPU implemented on an FPGA board is the Open Scalable Processor Architecture (OpenSPARC) project; the T2 version implements eight 64-bit cores with individual Level 1 (L1) and shared Level 2 (L2) cache, and can run programs similarly to a traditional CPU [Sun:OpenSPARC T2].

Multi-core processors also exist in other forms, such as Digital Signal Processors (DSPs), game consoles, and networking hardware [Multi-core processor]. This paper will focus on multi-core CPUs, but it is important for the analyst to recognize the similarities and differences of multi-core CPUs to other multi-core processors, such as GPUs and FPGAs.

1.3 Multi-Core CPUs versus GPUs

Recent studies focused on optimizing the performance of certain types of applications have claimed from 25x to 100x or more performance speedup by utilizing GPUs instead of CPUs for the primary computation task [Lee10]. While this paper will be focusing on multi-core CPU performance, it is important to briefly contrast this with GPGPU performance, as the analyst may need to consider both areas considering the prevalence of GPGPU approaches in high performance computing.

While the speedup numbers of these studies using a GPGPU strategy are dramatic, CPUs and GPUs are designed to be efficient at significantly different types of tasks. The architectural differences between CPUs and GPUs cause CPUs to perform better on latency-sensitive, partially sequential, single sets of tasks [Lee10]. In contrast, GPUs perform better with latency-tolerant, highly parallel and independent tasks [Lee10]. Depending on the application being studied, the overall performance may be better or worse on a GPU, or the latencies may be unacceptable even if throughput is relatively high.

This section defined a multi-core processor then compared the key types of multi-core processors for performance computing. Accordingly, the analyst may identify the relevant types of multi-core processors involved in a particular problem. The next section will introduce the types of parallelism employed by multi-core CPUs to increase performance, and subsequent sections will apply these concepts to performance measurement and analytical modeling.

2. Parallelism and Performance in Multi-Core CPUs

Because multi-core CPUs exploit parallelism to enhance performance, an understanding of the key types of parallelism is important to analyzing performance. Parallelism is a complex topic but a basic understanding of three types of parallelism is sufficient here. Instruction-level parallelism, thread-level parallelism, and data-level parallelism are all employed by various multi-core CPU architectures, and have different impacts on performance that must be understood to conduct thorough performance analysis.

2.1 Instruction-Level Parallelism

The first key type of parallelism, instruction-level parallelism (or ILP), involves executing certain instructions of a program simultaneously which would otherwise be executed sequentially [Goossens10], which may positively impact performance depending on the instruction mix in the application.

Most modern CPUs utilize instruction-level parallelization techniques such as pipelining, superscalar execution, prediction, out-of-order execution, dynamic branch prediction or address speculation [Goossens10]. However, only certain portions of a given program's instruction set may be suitable for instruction-level parallelization, as a simple example illustrates below in Figure 2. Because steps 1 and 2 of the sequential operation are independent of each other, a processor employing instruction-level parallelism can run instructions 1.A. and 1.B. simultaneously and thereby reduce the operation cycles to complete the operation by 33%. The last step must be executed sequentially in either case, however, as it is dependent on the two prior steps.