Dan Negrut

Primer:
Elements of Processor Architecture.
The Hardware/Software Interplay.

Supplemental material provided for
ECE/ME/EMA/CS 759

October 23, 2013

Table of Contents

Preface: Purpose and Structure 4
Document Structure 5
Acknowledgments 6
1. Sequential Computing: Basic Architecture and Software Model 7
1.1. From high level language to executable instructions 7
1.2. From transistors to the CPU 10
1.2.1. The CPU's control unit 10
1.2.2. The CPU's arithmetic logic unit 13
1.2.3. Registers 17
1.2.4. The Fetch-Decode-Execute sequence 19
1.2.5. The Bus 22
1.2.6. Execution Pipelining 23
1.3. The Clock and Execution Performance Metrics. Superscalar Processors 27
1.4. Bandwidth and latency issues in data movement 31
1.5. Data Storage 32
1.5.1. Static RAM (SRAM) and Dynamic RAM (DRAM) 33
1.6. The CPU – Main Memory Communication 34
1.7. Cache Memory 36
1.7.1. Caching Strategies 37
1.7.2. Reading and Writing Data in the Presence of Caches 42
1.8. Virtual memory 43
1.8.1. Introduction, Overview, and Nomenclature 43
1.8.2. Virtual Memory as an Enabler of Caching for Secondary Memory 45
1.8.3. The Anatomy of a Virtual Address. Address Translation. 45
1.8.4. The Anatomy of the Virtual Memory 48
1.8.5. How Multitasking if Facilitated by Memory Virtualization 50
1.8.6. Handling Page Faults 52
1.8.7. The Translation Lookaside Buffer (TLB) 54
1.8.8. Memory Access: The Big Picture 54

1.9. From Code to an Executable Program 56
1.9.1. The Compiler and Related Issues 58
1.9.2. The Assembler and Related Issues 63
1.9.3. The Linker and Related Issues 67
1.9.4. The Loader and Related Issues 71
2. Parallel Computing: The Basic Architecture and Software Model 73
2.1. Introduction. Parallel Execution vs. Parallel Computing 73
2.2. Increasing Execution Speed through Instruction Level Parallelism 76
2.3. Superscalar Architectures. Hyper-Threading 78
2.4. The Three Walls to Sequential Computing 80
2.4.1. The Memory Wall 82
2.4.2. The frequency/power wall 85
2.4.3. The Instruction Level Parallelism Wall 85
2.4.4. Concluding Remarks 86
2.5. Multi-Core Chips 86
2.5.1. The Intel Haswell Architecture 87
2.5.2. The Nvidia Fermi Architecture 94
References 107

Preface: Purpose and Structure

This primer is intended to help graduate students in disciplines such as Mechanical Engineering, Chemistry, Civil Engineering, or Biology with answering a simple question: "How can parallel computing augment my research and its impact?". These are individuals who took a C, C++, or Java programming class who understand the power of Data Analytics and Computational Science but lack the skills to leverage existing hardware to solve, for instance, a climate modeling problem at a resolution that offers unprecedented level of detail; providing the computational support to sequence the human genome and identify relevant mutations in matters of seconds; enable real time 3D visualization of the human heart; perform an electronic structure computation that backs up an accurate molecular dynamics simulation at the nanoscale; or resolve through direct numerical simulation two phase flows and spray dynamics for improved internal combustion engine design. A lot of emphasis has been recently placed in Science and Engineering on using modeling and computer simulation as alternatives to experimental and testing efforts. A validated computer program indeed can serve as a very effective and cost-conscious alternative to experimental tests in guiding and accelerating scientific discovery and engineering innovation. Moreover, recent advances in data sensing and collection has led to a deluge of raw data that needs to be processed, interpreted, and leveraged in fields as diverse as health care, social networks, and genetics.

The purpose of this primer is to help the student get a basic understanding of the architecture of modern processors and the hardware and software context within which they operate. This effort was motivated by the belief that in order to write good software a basic understanding of the hardware that will execute one's program will help. Understanding how the hardware works enables one to visualize the process of parallel execution. This skill takes time to master but it is good to possess since it allows for: (a) expeditious design of parallel programs; and (b) writing software that runs fast and correctly. In other words, it enables one to get it done and have it done right.

As the title suggests, this is a primer. The presentation follows a 10-80 approach: with 10% of the discussion required to exhaustively cover a topic the hope is that the most useful 80% of the concepts will be understood. Depth was not a priority, nor was breadth. The priority was in providing enough information to help with the process of designing fast and correct parallel code. As such, this text does not attempt to present a comprehensive landscape of the topics covered. My knowledge would not allow it, and you would probably not be interested on a treatise on the topic of instruction level parallelism, for instance.

The community revolving around computers approaches them from three perspectives. There are colleagues seeking answers to difficult issues, e.g., how to design a better instruction architecture set, how to improve pipelining and out of order execution, how a memory consistency model should be formulated. They will find this primer too basic and short on details. At the other end of the spectrum, there are colleagues for which computers provide a means to an end. The computer can be used to watch a movie, send email, or run an off-the-shelf commercial Computer Aided Engineering program to produce data that is subsequently processed to yield insights in engineering design. This textbook is meant to assist the group in between, colleagues who need a *basic understanding and intuition* into the

inner workings of a multiprocessor computer to develop software that can solve larger domain specific problems faster. Recent trends suggest that the required discipline boundary crossing by engineers and scientists into a territory that traditionally has been identified with the Computer Science and/or Computer Engineering fields is increasingly desirable. This primer attempts to smoothen the discipline boundary crossing, a process as worthwhile as frustrating at times. The role of parallel computing is bound to increase and there is a clear need for engineers and scientists who are savvy computer users.

Document Structure

In its structure, this primer reflects a trend that obdurately held for a decade: sequential computing model has passed its high-water mark owing to realities in the microprocessor industry. Parallel computing is riding a rising tide owing to better use of power per amount of computation, better ability to hide memory latencies with useful computation, and a simpler microarchitecture that does not need to handle complexities associated with instruction level parallelism.

The selection of topics in this primer caters to students with no formal training in computer architecture or operating systems. The primer tries to cover these two topics at a pace that would take four weeks of class to go through the material. In-depth coverage of each of these two topics typically takes two semester courses and PhD theses continue to be inspired by problems in these two areas. As such, it should come as no surprise if after reading the section on buses (not the yellow things that take kids to school but the conduits for information passing in a computer system), one will be left with unanswered questions. Rather than providing exhaustive coverage of each topic, the goal is to introduce the concepts that combine to form a framework in which the process of parallel execution can be understood and visualized, the two ultimate goals of any engineer or scientist who makes the computer a means to an end.

Presently, the primer is organized in two chapters. The first provides an overview of critical building blocks that combine to make the computer work. The discussion first introduces the Von Neumann computing model and the concept of instruction and transistor. The machine instruction holds a work order that eventually will lead to a choreographed interplay of groups of transistors to fulfill the action spelled out by this instruction. The fetch-decode-execute cycle is analyzed, which requires the prior discussion of the concepts of control unit, arithmetic logic unit, and registers. Physical memory issues are covered next, which brings to the front the concepts of memory hierarchy, random access memory and cache along with two key attributes: latency and bandwidth. The interplay between hardware and software is the topic of a separate section, which discusses the virtual memory and a support cast that includes translation-lookaside-buffers, addressing techniques, and memory organization. A final section is dedicated to the laborious process of starting with source code and obtaining a program that might rely on third party libraries at link time.

The second chapter builds on the first one and provides an overview of one CPU processor; i.e., Intel's Haswell, and one GPU card; i.e. Nvidia's Fermi GTX480, in an exercise that places in context the concepts of the first chapter. The chapter sets off with a discussion of the three walls in sequential computing: memory, power, and instruction level parallelism, that are generally regarded as responsible for a gradual embracing of microarchitecture designs aimed at parallel computing. The steady and fast