Newnes

INCLUDES

FREE
NEWNES ONLINE
MEMBERSHIP

SOFTWARE DEVELOPMENT FOR EMBEDDED MULTI-CORE SYSTEMS

A Practical Guide for Using Embedded Intel® Architecture

- Covers the latest Intel® Multi-core Architecture
- Details Scalar Optimization and Parallel Optimization
- Discusses the benefits of using Multi-core for embedded systems

Max Domeika

Software Development for Embedded Multi-core Systems

Software Development for Embedded Multi-core Systems

A Practical Guide Using Embedded Intel® Architecture

Max Domeika

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier

Newnes

Cover image by iStockphoto
Newnes is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Library of Congress Cataloging-in-Publication Data
Domeika, Max.
Software development for embedded multi-core systems : a practical guide using embedded Intel architecture / Max Domeika.
p. cm.
ISBN 978-0-7506-8539-9
1. Multiprocessors. 2. Embedded computer systems. 3. Electronic data processing—Distributed processing. 4. Computer software—Development. I. Title.
QA76.5.D638 2008
004′.35—dc22 2008006618

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN: 978-0-7506-8539-9

Typeset by Charon Tec Ltd (A Macmillan Company), Chennai, India
www.charontec.com

Printed in the United States of America
08 09 10 11 10 9 8 7 6 5 4 3 2 1

Contents

Preface .... *ix*

Acknowledgments .... *xiii*

Chapter 1: Introduction ***1***
1.1 Motivation 3
1.2 The Advent of Multi-core Processors 4
1.3 Multiprocessor Systems Are Not New 4
1.4 Applications Will Need to be Multi-threaded 6
1.5 Software Burden or Opportunity 8
1.6 What is Embedded? 10
1.7 What is Unique About Embedded? 13
Chapter Summary 14

Chapter 2: Basic System and Processor Architecture ***17***
Key Points 17
2.1 Performance 19
2.2 Brief History of Embedded Intel® Architecture Processors 20
2.3 Embedded Trends and Near Term Processor Impact 37
2.4 Tutorial on x86 Assembly Language 39
Chapter Summary 53
Related Reading 54

Chapter 3: Multi-core Processors and Embedded ***55***
Key Points 55
3.1 Motivation for Multi-core Processors 56
3.2 Multi-core Processor Architecture 57
3.3 Benefits of Multi-core Processors in Embedded 62
3.4 Embedded Market Segments and Multi-core Processors 63

3.5 Evaluating Performance of Multi-core Processors 69
Chapter Summary 87
Related Reading 88

Chapter 4: Moving to Multi-core Intel Architecture 89
Key Points 89
4.1 Migrating to Intel Architecture 91
4.2 Enabling an SMP OS 111
4.3 Tools for Multi-Core Processor Development 117
Chapter Summary 136
Related Reading 137

Chapter 5: Scalar Optimization and Usability 139
Key Points 139
5.1 Compiler Optimizations 143
5.2 Optimization Process 153
5.3 Usability 161
Chapter Summary 170
Related Reading 170

Chapter 6: Parallel Optimization Using Threads 173
Key Points 173
6.1 Parallelism Primer 175
6.2 Threading Development Cycle 184
Chapter Summary 206
Related Reading 207

Chapter 7: Case Study: Data Decomposition 209
Key Points 209
7.1 A Medical Imaging Data Examiner 209
Chapter Summary 245

Chapter 8: Case Study: Functional Decomposition 247
Key Points 247
8.1 Snort 248

8.2 Analysis 251
8.3 Design and Implement 258
8.4 Snort Debug 280
8.5 Tune 282
Chapter Summary 286

Chapter 9: Virtualization and Partitioning* *287
Key Points 287
9.1 Overview 287
9.2 Virtualization and Partitioning 290
9.3 Techniques and Design Considerations 304
9.4 Telecom Use Case of Virtualization 322
Chapter Summary 342
Related Reading 344

Chapter 10: Getting Ready for Low Power Intel Architecture* *347
Key Points 347
10.1 Architecture 349
10.2 Debugging Embedded Systems 362
Chapter Summary 382

Chapter 11: Summary, Trends, and Conclusions* *385
11.1 Trends 387
11.2 Conclusions 392

Appendix A* *393

Glossary* *394

Index* *411

Preface

At the Fall 2006 Embedded Systems Conference, I was asked by Tiffany Gasbarrini, Acquisitions Editor of Elsevier Technology and Books if I would be interested in writing a book on embedded multi-core. I had just delivered a talk at the conference entitled, "Development and Optimization Techniques for Multi-core SMP" and had given other talks at previous ESCs as well as writing articles on a wide variety of software topics. Write a book – this is certainly a much larger commitment than a presentation or technical article. Needless to say, I accepted the offer and the result is the book that you, the reader, are holding in your hands. My sincere hope is that you will find value in the following pages.

Why This Book?

Embedded multi-core software development is the grand theme of this book and certainly played the largest role during content development. That said, the advent of multi-core is not occurring in a vacuum; the embedded landscape is changing as other technologies intermingle and create new opportunities. For example, the intermingling of multi-core and virtualization enable the running of multiple operating systems on one system at the same time and the ability for each operating system to potentially have full access to all processor cores with minimal drop off in performance. The increase in the number of transistors available in a given processor package is leading to integration the likes of which have not been seen previously; converged architectures and low power multi-core processors combining cores of different functionality are increasing in number. It is important to start thinking now about what future opportunities exist as technology evolves. For this reason, this book also covers emerging trends in the embedded market segments outside of pure multi-core processors.

When approaching topics, I am a believer in fundamentals. There are two reasons. First, it is very difficult to understand advanced topics without having a firm grounding in the basics. Second, advanced topics apply to decreasing numbers of people. I was at an instrumentation device company discussing multi-core development tools and the topic turned to 8-bit code optimization. I mentioned a processor issue termed partial register stalls and then found myself discussing in detail how the problem occurs and the innermost workings of the cause inside the processor (register renaming to eliminate false dependencies, lack of hardware mechanisms to track renamed values contained in different partial registers). I then realized while the person to whom I was discussing was thoroughly interested, the rest of the people in the room were lost and no longer paying attention. It would have been better to say that partial register stalls could be an issue in 8-bit code. Details on the problem can be found in the optimization guide.

My book will therefore tend to focus on fundamentals and the old KISS[1] principle:

- What are the high level details of X?
- What is the process for performing Y?

Thanks. Now show me a step-by-step example to apply the knowledge that I can reapply to my particular development problem.

That is the simple formula for this book:

1. Provide sufficient information, no more and no less.
2. Frame the information within a process for applying the information.
3. Discuss a case study that provides practical step-by-step instructions to help with your embedded multi-core projects.

Intended Audience

The intended audience includes employees at companies working in the embedded market segments who are grappling with how to take advantage of multi-core processors for their respective products. The intended audience is predominately embedded software development engineers; however, the information is approachable enough for less day-to-day technical embedded engineers such as those in marketing and management.

[1] KISS = Keep It Simple, Stupid.

Readers of all experience and technical levels should derive the following benefits from the information in this book:

- A broad understanding of multi-core processors and the challenges and opportunities faced in the embedded market segments.
- A comprehensive glossary of relevant multi-core and architecture terms.

Technical engineers should derive the following additional benefits:

- A good understanding of the optimization process of single processors and multi-core processors.
- Detailed case studies showing practical step-by-step advice on how to leverage multi-core processors for your embedded applications.
- References to more detailed documentation for leveraging multi-core processors specific to the task at hand. For example, if I were doing a virtualization project, what are the steps and what specific manuals do I need for the detailed information?

The book focuses on practical advice at the expense of theoretical knowledge. This means that if a large amount of theoretical knowledge is required to discuss an area or a large number of facts are needed then this book will provide a brief discussion of the area and provide references to the books that provide more detailed knowledge. This book strives to cover the key material that will get developers to the root of the problem, which is taking advantage of multi-core processors.

Acknowledgments

There are many individuals to acknowledge. First, I'd like to thank Rachel Roumeliotis for her work as editor.

I also need to acknowledge and thank the following contributors to this work:

- Jamel Tayeb for authoring Chapter 9 – Virtualization and Partitioning. Your expertise on partitioning is very much appreciated.
- Arun Raghunath for authoring Chapter 8 – Case Study: Functional Decomposition. Thank you for figuring out how to perform flow pinning and the detailed analysis performed using Intel® Thread Checker. Thanks also to Shwetha Doss for contributions to the chapter.
- Markus Levy, Shay Gal-On, and Jeff Biers for input on the benchmark section of Chapter 3.
- Lori Matassa for contributions to big endian and little endian issues and OS migration challenges in Chapter 4.
- Clay Breshears for his contribution of the tools overview in Chapter 4.
- Harry Singh for co-writing the MySQL case study that appears in Chapter 5.
- Bob Chesebrough for his contribution on the *Usability* section in Chapter 5.
- Lerie Kane for her contributions to Chapter 6.
- Rajshree Chabukswar for her contributions of miscellaneous power utilization techniques appearing in Chapter 10.
- Rob Mueller for his contributions of embedded debugging in Chapter 10.
- Lee Van Gundy for help in proofreading, his many suggestions to make the reading more understandable, and for the BLTK case study.

- Charles Roberson and Shay Gal-On for a detailed technical review of several chapters.
- David Kreitzer, David Kanter, Jeff Meisel, Kerry Johnson, and Stephen Blair-chappell for review and input on various subsections of the book.

Thank you, Joe Wolf, for supporting my work on this project. It has been a pleasure working on your team for the past 4 years.

This book is in large part a representation of my experiences over the past 20 years in the industry so I would be remiss to not acknowledge and thank my mentors throughout my career – Dr. Jerry Kerrick, Mark DeVries, Dr. Edward Page, Dr. Gene Tagliarini, Dr. Mark Smotherman, and Andy Glew.

I especially appreciate the patience, support, and love of my wife, Michelle, and my kids, James, Caleb, and Max Jr. I owe them a vacation somewhere after allowing me the sacrifice of my time while writing many nights and many weekends.

CHAPTER 1

Introduction

The proceeding conversation is a characterization of many discussions I've had with engineers over the past couple of years as I've attempted to communicate the value of multi-core processors and the tools that enable them. This conversation also serves as motivation for the rest of this chapter.

A software engineer at a print imaging company asked me, "What can customers do with quad-core processors?" At first I grappled with the question thinking to a time where I did not have an answer. "I don't know," was my first impulse, but I held that comment to myself. I quickly collected my thoughts and recalled a time when I sought an answer to this very question:

- Multiple processors have been available on computer systems for years.
- Multi-core processors enable the same benefit as multiprocessors except at a reduced cost.

I remembered my graduate school days in the lab when banks of machines were fully utilized for the graphics students' ray-tracing project. I replied back, "Well, many applications can benefit from the horsepower made available through multi-core processors. A simple example is image processing where the work can be split between the different cores."

The engineer then stated, "Yeah, I can see some applications that would benefit, but aren't there just a limited few?"

My thoughts went to swarms of typical computer users running word processors or browsing the internet and not in immediate need of multi-core processors let alone the fastest single core processors available. I then thought the following:

- Who was it that said 640 kilobytes of computer memory is all anyone would ever need?
- Systems with multiple central processing units (CPUs) have not been targeted to the mass market before so developers have not had time to really develop applications that can benefit.

I said, "This is a classic chicken-and-egg problem. Engineers tend to be creative in finding ways to use the extra horsepower given to them. Microprocessor vendors want customers to see value from multi-core because value equates to price. I'm sure there will be some iteration as developers learn and apply more, tools mature and make it easier, and over time a greater number of cores become available on a given system. We will all push the envelope and discover just which applications will be able to take advantage of multi-core processors and how much."

The engineer next commented, "You mentioned 'developers learn.' What would I need to learn – as if I'm not overloaded already?"

At this point, I certainly didn't want to discourage the engineer, but also wanted to be direct and honest so ran through in my mind the list of things to say:

- Parallel programming will become mainstream and require software engineers to be fluent in the design and development of multi-threaded programs.
- Parallel programming places more of the stability and performance burden on the software and the software engineer who must coordinate communication and control of the processor cores.

"Many of the benefits to be derived from multi-core processors require software changes. The developers making the changes need to understand potential problem areas when it comes to parallel programming."

"Like what?" the overworked engineer asked knowing full well that he would not like the answer.

"Things like data races, synchronization and the challenges involved with it, workload balance, etc. These are topics for another day," I suggested.

Having satisfied this line of questioning, my software engineering colleague looked at me and asked, "Well what about embedded? I can see where multi-core processing can help in server farms rendering movies or serving web queries, but how can embedded applications take advantage of multi-core?"

Whenever someone mentions embedded, my first wonder is – what does he or she mean by "embedded"? Here's why:

- Embedded has connotations of "dumb" devices needing only legacy technology performing simple functions not much more complicated than those performed by a pocket calculator.
- The two applications could be considered embedded. The machines doing the actual work may look like standard personal computers, but they are fixed in function.

I responded, "One definition of embedded is fixed function which describes the machines running the two applications you mention. Regardless, besides the data parallel applications you mention, there are other techniques to parallelize work common in embedded applications. Functional decomposition is one technique or you can partition cores in an asymmetric fashion."

"Huh?" the software engineer asked.

At this point, I realized that continuing the discussion would require detail and time that neither of us really wanted to spend at this point so I quickly brought up a different topic. "Let's not talk too much shop today. How are the kids?" I asked.

1.1 Motivation

The questions raised in the previous conversation include:

- What are multi-core processors and what benefits do they provide?
- What applications can benefit from multi-core processors and how do you derive the benefit?
- What are the challenges when applying multi-core processors? How do you overcome them?
- What is unique about the embedded market segments with regard to multi-core processors?

Many of the terms used in the conversation may not be familiar to the reader and this is intentional. The reader is encouraged to look up any unfamiliar term in the glossary or hold off until the terms are introduced and explained in detail in later portions of the book. The rest of this chapter looks at each of the key points mentioned in the conversation and provides a little more detail as well as setting the tone for the rest of the book. The following chapters expound on the questions and answers in even greater detail.

1.2 The Advent of Multi-core Processors

A *multi-core processor* consists of multiple *central processing units* (CPUs) residing in one physical package and interfaced to a motherboard. Multi-core processors have been introduced by semiconductor manufacturers across multiple market segments. The basic motivation is *performance* – using multi-core processors can result in faster execution time, increased throughput, and lower power usage for embedded *applications*. The expectation is that the ratio of multi-core processors sold to single core processors sold will trend even higher over time as the technical needs and economics make sense in increasing numbers of market segments. For example, in late 2006 a barrier was crossed when Intel® began selling more multi-core processors than single core processors in the desktop and server market segments. Single core processors still have a place where absolute cost is prioritized over performance, but again the expectation is that the continuing march of technology will enable multi-core processors to meet the needs of currently out-of-reach market segments.

1.3 Multiprocessor Systems Are Not New

A *multiprocessor system* consists of multiple processors residing within one system. The processors that make up a multiprocessor system may be single core or multi-core processors. Figure 1.1 shows three different system layouts, a single core/single processor system, a multiprocessor system, and a multiprocessor/multi-core system.

Multiprocessor systems, which are systems containing multiple processors, have been available for many years. For example, pick up just about any book on the history of computers and you can read about the early Cray [1] machines or the Illiac IV [2]. The first widely available multiprocessor systems employing *x86 processors* were the Intel iPSC systems of the late 1980s, which configured a set of Intel® i386™ processors in a cube formation. The challenge in programming these systems was how to efficiently split the work between multiple processors each with its own memory. The same challenge exists in

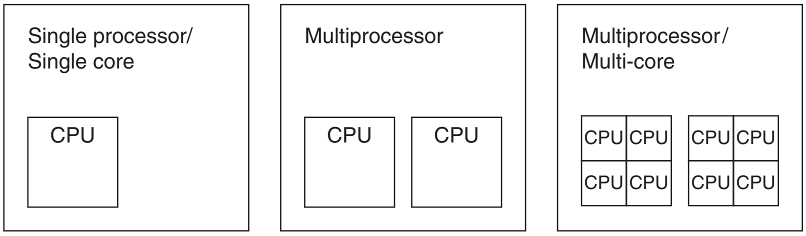

Figure 1.1: Three system configurations

today's multi-core systems configured in an asymmetric layout where each processor has a different view of the system. The first widely available dual processor *IA-32 architecture* system where memory is shared was based upon the Pentium® processor launched in 1994. One of the main challenges in programming these systems was the coordination of access to shared data by the multiple processors. The same challenge exists in today's multi-core processor systems when running under a *shared memory* environment.

Increased performance was the motivation for developing multiprocessor systems in the past and the same reason multi-core systems are being developed today. The same relative benefits of past multiprocessor systems are seen in today's multi-core systems. These benefits are summarized as:

- Faster execution time
- Increased throughput

In the early 1990s, a group of thirty 60 *Megahertz* (MHz) Pentium processors with each processor computing approximately 5 million floating-point operations a second (MFLOPS) amounted in total to about 150 MFLOPS of processing power. The processing power of this pool of machines could be tied together using an *Application Programming Interface* (API) such as Parallel Virtual Machine [3] (PVM) to complete complicated ray-tracing algorithms.

Today, a single Intel® Core™ 2 Quad processor delivers on the order of 30,000 MFLOPS and a single Intel® Core™ 2 Duo processor delivers on the order of 15,000 MFLOPS. These machines are tied together using PVM or Message Passing Interface [4] (MPI) and complete the same ray-tracing algorithms working on larger problem sizes and finishing them in faster times than single core/single processor systems.

The Dual-Core Intel® Xeon® Processor 5100 series is an example of a multi-core/multi-processor that features two dual-core Core™ processors in one system. Figure 1.2 is a sample embedded platform that employs this particular dual-core dual processor.

1.4 Applications Will Need to be Multi-threaded

Paul Otellini, CEO of Intel Corporation, stated the following at the Fall 2003 Intel Developer Forum:

> We will go from putting Hyper-threading Technology in our products to bringing dual-core capability in our mainstream client microprocessors over time. For the software developers out there, you need to assume that threading is pervasive.

This forward-looking statement serves as encouragement and a warning that to take maximum advantage of the performance benefits of future processors you will need to take action. There are three options to choose from when considering what to do with multi-core processors:

1. Do nothing
2. Multi-task or Partition
3. Multi-thread

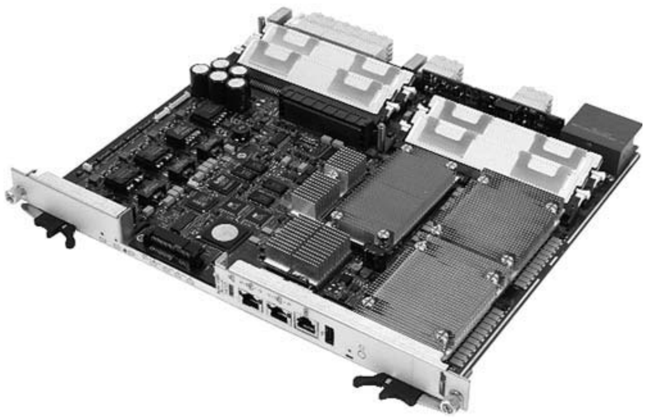

Figure 1.2: Intel NetStructure® MPCBL0050 single board computer

The first option, "Do nothing," maintains the same legacy software with no changes to accommodate multi-core processors. This option will result in minimal performance increases because the code will not take advantage of the multiple cores and only take advantage of the incremental increases in performance offered through successive generations of improvements to the *microarchitecture* and the software tools that optimize for them.

The second option is to multi-task or partition. Multi-tasking is the ability to run multiple processes at the same time. *Partitioning* is the activity of assigning cores to run specific operating systems (OSes). Multi-tasking and partitioning reap performance benefits from multi-core processors. For embedded applications, partitioning is a key technique that can lead to substantial improvements in performance or reductions in cost.

The final option is to multi-thread your application. Multi-threading is one of the main routes to acquiring the performance benefits of multi-core processors. Multi-threading requires designing applications in such a way that the work can be completed by independent workers functioning in the same sandbox. In multi-threaded applications, the workers are the individual processor cores and the sandbox represents the application data and memory.

Figure 1.3 is a scenario showing two classes of software developers responding to the shift to multi-core processors and their obtained application performance over time. The *x*-axis represents time, and the *y*-axis represents application performance. The top line labeled "Platform Potential" represents the uppermost bound for performance of a given platform and is the ceiling for application performance. In general, it is impossible to perfectly optimize your code for a given processor and so the middle line represents the attained performance for developers who invest resources in optimizing. The bottom

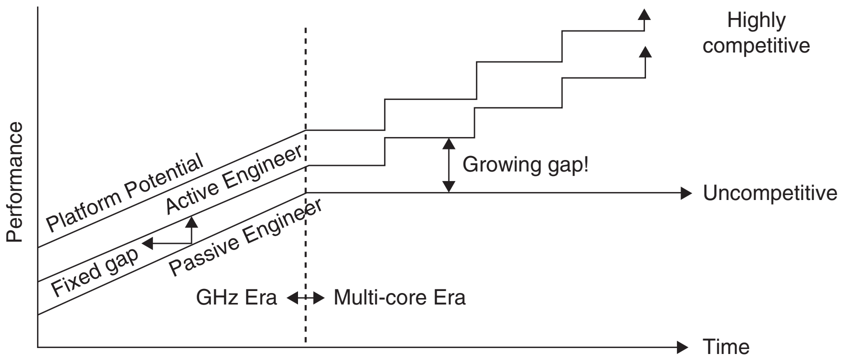

Figure 1.3: Taking advantage of multi-core processors