

# Efficient Memory Programming

David Loshin

Efficient Memory Programming is written by David Loshin, and published by McGraw-Hill, 1998, hardcover, ISBN 0-07-038868-7, 320 pp., \$50.00.

As little as ten years ago, if you wanted to optimize the execution speed of your program, you counted instruction cycles and modified your code to minimize the number of cycles the instructions required. With today's highly pipelined superscaler RISC processors with multiple levels of memory caches, virtual memory, and speculative execution, the optimization techniques of years past are not only insufficient, they may even produce a negative impact on your code's execution speed. *Efficient Memory Programming* is David Loshin's attempt to explain how these various artifacts inter-relate and affect code execution speed, as well as to provide concrete examples of techniques that will increase execution speeds in these environments.

The book is organized in a hierarchical manner that mimics modern processor architectures. This provides for a very logical approach that builds upon previous chapters and makes it very easy for the reader to move down the memory hierarchy and through the optimization process. From the highest level, the book is organized into three sections; the hardware, the software (which is titled Performance), and advanced topics, such as cooperating processes, network communications, and client/server computing.

At the next lower level, the hardware section starts with a discussion of the CPU, including the basics of data and control hazards, superscaler architectures, superpipelining, and branch optimizations and predictions. Mr. Loshin then works his way down the memory hierarchy, starting with registers, then working through multiple levels of cache memory, main memory, virtual memory, and disks and file systems.

The software section starts out with a chapter on optimization basics where the author discusses concepts such as control flow, various kinds of optimizations, and reduction of space, code size, complexity and latency.

This is followed by a chapter on analyzing the performance of a code stream. Various techniques with their strengths and weaknesses are presented, along with sage advice on when to stop optimizing and when to not to invest the effort optimize at all.

The remaining chapters in this section deal with data access and dependency analysis, advanced optimization techniques, and optimizations for concurrency, including threads, synchronization, instruction scheduling, and software pipelining.

The book is filled with example code and various architectural examples from the Intel P6. The author also provides a summary at the end of each chapter to reinforce the concepts contained within.

Unfortunately, there are several problems with the book that detract from its usefulness. The organization within the chapters could use some careful editing. There are multiple instances of a concept or diagram appearing with the necessary explanatory text occurring eight to ten pages later. Additionally, although there are several examples based on the Intel P6 processor, I generally found them unsatisfying. They either needed to be placed earlier in the text with the explanatory text then building on the example or they needed to have more text immediately following the example to tie it into the concept the author was attempting to illustrate.

Another issue I had with the book was the author's inability to reach an appropriate level for the explanation of many of the concepts. Mr. Loshin apparently couldn't decide whether to give a high level explanation or delve into the lower level details, so in many instances, he settled for a murky middle ground that is confusing to the reader.

There are also multiple errors in the technical explanations that should have been caught during the proof and edit cycle. Although most of them are not critical, they do detract from the book as the reader tries to determine what was really meant. An example from page 80 illustrates this type of error. "Because segments are of variable length, two registers must be used in a pure segmentation system. A base register is used to indicate the base address in memory where the segment is loaded. A second register holds the length of the register so that the system will not attempt to access a memory location that exceeds the boundary of the segment." While an astute reader will recognize that the last usage of the word "register" should have been "segment", the reader of a book of this nature should not have to be particularly astute. The assumption is that the reader is not an expert in the topic and is reading the book to better understand the subject matter.

Even worse are the code examples. Over half of the code examples are either wrong or do not match the explanation in the corresponding text. For example, page 16 presents the assembly language for a loop that is supposed to compute:

```
for(i = 0; i <= 100; i++)
    c(i) = a(i) + b(i);
```

Instead, the assembly code computes:

```
for(i = 0; i <= 100; i++)
    c(0) = a(i) + b(i);
```

Although this is a relatively minor error, it unfortunately presages what is to come later. This kind of error becomes particularly frustrating when the author tries to explain optimization techniques. The majority of the code examples in this part of the book change variable names in midstream, change the program logic, or just plain compute the wrong answer. For instance, the example on index set splitting shows a loop that runs

from 0 to  $N - 1$ . After the author splits it, the two split loops run from 0 to  $N/2 - 1$  and from  $N/2 + 1$  to  $N - 1$ . Many of the code examples have similar "off by one" index problems.

Another example is the code to illustrate software pipelining. The pipelined version computes the wrong answer. The next to the last two iterations of the original loop are not executed in the pipelined version. This is particularly distressing since the discussion of many of the optimization techniques rely on the reader understanding the code examples since insufficient explanation on implementing the technique is provided in the text.

Although this book covers an important topic that many programmers should understand, I cannot recommend it in its present state. The book needs a careful editing to correct the problems mentioned previously. Until such editing takes place, if you are interested in the subject matter presented, *Computer Architecture, A Quantitative Approach* by John Hennessey and David Patterson ( Morgan Kaufmann Publishers) covers the same material at a much greater level of detail and, more importantly, without the errors.

Copyright © 1999 by Kenneth R. Frazer