Getting Your Bits in Order

Igor L. Markov
University of Michigan

 Engineers find EDA software very time-consuming and must wait longer when working with larger chips. For tasks like microprocessor verification, huge server farms and dedicated hardware systems have been constructed. To better use server farms, EDA vendors are now developing new parallel algorithms. However, even with expensive computational resources, EDA software developers must make judicious choices of key algorithms and programming techniques. Some of these techniques are known only to exceptional engineers.

A rule of thumb is that one exceptional software engineer is worth 20 mediocre ones. I was reminded of this rule when teaching an undergraduate course on algorithms and data structures at the University of Michigan. Students had to complete five software projects dealing with the implementation of standard data structures, maze routing, the construction of spanning trees, text processing, and the traveling salesman problem. With two trial submissions per day to an "autograder," students received real-time feedback on the correctness and runtime of their programs, and also compared themselves to their peers.

Year after year, the best student solutions in this course outperformed solutions prepared by teaching assistants, but this was not the case in the Fall 2009 semester. My assistant Mark was an undergraduate student who took this course in the previous semester, yet he could teach me a few things about algorithms. (He later represented Michigan at the ACM International Collegiate Programming Competition and won a gold medal at the world finals.)

On Project 1, Mark’s program was several times faster than those submitted by students, but that was seemingly due to his optimized string I/O, as overall runtimes were small. On Project 2, the gap was even greater, although the best students in class tried hard to match Mark’s runtime, and I/O runtime was not important in this project because it was dwarfed by a branch-and-bound algorithm. Digging through Mark’s code, I identified the critical loop inside branch-and-bound. It began with:

```
for(; i; i = i & (i - 1)) {...}
```

If you consider yourself an expert programmer, pause for a second and think what it does. In the meantime, let me tell you where you can learn such tricks and many more: in the book *Hacker’s Delight* by Henry S. Warren Jr.

The book contains 16 chapters, starting with an introduction that defines an assembly-like instruction set to more concisely express bit-level algorithms (many chapters use C-like code instead of or in addition to those instructions). Bit-level algorithms are heavily used in high-performance data structures and are indispensable in many EDA contexts, including high-level and logic synthesis, formal and simulation-driven verification, detailed routing, and high-performance mask processing. I found that many chapters can be read out of order, and when background is required, the author provides references to the requisite earlier chapters. Chapter 2, “Basics,” discusses standard bitwise operations and their use to perform various tasks, such as compute max and min values without comparisons, rotate shifts, perform multibyte operations, and so on. Chapter 3 focuses on
arithmetic operations that can be efficiently performed with powers of two, such as rounding up or down to the next power of two. Chapter 4 explains efficient bounds checking and bounds propagation. Chapter 5 shows how to count 1 bits, leading and trailing 0s, and how to compute parity. Chapter 6 gives highly efficient techniques for finding the first 0 byte in a string and finding a string of 1 bits of a given length. Chapter 7 covers reversing operations for bits and bytes, and transpositions of bit matrices, as well as general permutations and index transformations.

Chapters 8-12 and 15 cover efficient arithmetic operations. Chapters 8-10 discuss multiword multiplication and multiword integer division, as well as multiplication and division by constants (which can be optimized for these constants). In EDA, such techniques would be useful not only for writing faster programs, but also in high-level synthesis.

Chapter 11 focuses on efficient computation of elementary functions—square and cube roots, logarithms, and exponentiations. In Chapter 12, “Unusual Bases for Number Systems,” Warren readily admits that its material is probably not useful for anything practical, but it definitely has the wow! factor—have you ever tried working with base −2 or base −1 + i? Chapter 15 deals with floating-point arithmetic.

The material of Chapter 13, “Gray Codes,” should be familiar to computer engineers because Karnaugh maps number rows and columns in a gray-code order. Perhaps the reader can try, as an exercise, writing a gray-code generator in several lines of C. In earlier days as a graduate student, I used such a function to implement an optimal hypergraph partitioner in the standard-cell placement tool Capo.

Chapter 14 is perhaps the most high-level in terms of its algorithms, and has a direct relation to the design of circuit test structures (scan-chain layout). It covers the Hilbert’s curve and variants, also known as space-filling curves (see Figure 1).

In other words, we are discussing functions that map the unit segment into the unit square such that the image of the mapping is a curve that “fills” the entire square, up to required precision. The inverse mapping takes an (x,y) point in the unit square and finds the closest point on the curve, in terms of parameter t from the unit segment. Space-filling curves allow us to implement a simple and fast heuristic for solving the traveling salesman problem, which is a well-known technique to order placed flip-flops in a scan-chain (better heuristics exist, though). The construction of space-filling curves is not usually time-critical in EDA, but it isn’t something that a regular Joe, Rahul, or Ming can typically come up with unassisted.

Chapter 16 gives formulas for generating prime numbers, which are used in expandable hash tables and data structures based on them.

Going back to my teaching assistant Mark for a minute: It turns out that Mark had rigged the specification of Project 2 to allow only 26 possible lowercase characters, and no uppercase characters that I initially planned. He then packed subsets of 26 characters in single integers and iterated over such subsets very efficiently using for(;i; i&=i−1) {... To interpret this loop, note that subtracting 1 from the integer i zeros out the least significant 1 bit, regardless of the bit’s position, and replaces all trailing 0s with 1s. Performing bitwise AND (&) on 1 and i−1 thus simply clears the least significant 1 bit of i. With an additional XOR operation and bitwise trickery from chapter 5 of Hacker’s Delight, we can find the position of the cleared bit. Normally, iterating over a subset of integers requires a container data structure, which is many times slower than Mark’s solution, due to additional memory accesses. When this iteration is the bottleneck of branch-and-bound, the resulting improvement is asymptotically significant.

Readers familiar with Donald Knuth’s The Art of Computer Programming will note that some of this material can also be found there. Indeed, Knuth’s oeuvre is encyclopedic, whereas Warren’s book is
practical—Warren uses C where possible and skips most of the proofs. And yet, it features a number of tricks absent from Knuth’s book, including practical techniques for arithmetic operations and a greater variety of bit twiddles. While both books explain how to write exceptionally fast programs, they are silent on how to write programs exceptionally quickly. Perhaps such a book would also be of great value to the industry.

In summary, I highly recommend *Hacker’s Delight* to active software developers and their managers, as well as to researchers concerned about actual performance of their algorithms. Professors teaching courses on digital logic and EDA would find this book a great source of extra-credit assignments for talented students.

Direct questions and comments about this department to Igor Markov, 2260 Hayward St., Ann Arbor, MI 48109-2121; imarkov@eecs.umich.edu.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.
Authoritative
Cutting-Edge
Comprehensive

With over 414,000 articles covering the spectrum of computer science and engineering, CSDL is the definitive resource for academic, corporate, or government libraries. Whether your users are looking for the latest research on today's hot topic, foundational information, or quick answers to a problem, they will find what they need.

Learn more! www.computer.org/library

Your institution may qualify for special subscription discounts. Your institution may also qualify for a FREE 30-day trial of the CSDL.

Email csdl@computer.org for more details.