Programming Pearls

Second Edition

Jon Bentley

Programming Pearls

Second Edition

my organization and my company's purchasing department. Purchasing was swamped with similar orders; fifty people in my organization alone had placed individual orders. A friendly chat between my management and purchasing resulted in consolidating those fifty orders into one large order. In addition to easing administrative work for both organizations, this change also sped up one small piece of a computer system by a factor of fifty. A good systems analyst keeps an eye out for such savings, both before and after systems are deployed.

Sometimes good specifications give users a little less than what they thought was needed. In Column 1 we saw how incorporating a few important facts about the input to a sorting program decreased both its run time and its code length by an order of magnitude. Problem specification can have a subtle interaction with efficiency; for example, good error recovery may make a compiler slightly slower, but it usually decreases its overall time by reducing the number of compilations.

System Structure. The decomposition of a large system into modules is probably the single most important factor in determining its performance. After sketching the overall system, the designer should do a simple "back-of-the-envelope" estimate to make sure that its performance is in the right ballpark; such calculations are the subject of Column 7. Because efficiency is much easier to build into a new system than to retrofit into an existing system, performance analysis is crucial during system design.

Algorithms and Data Structures. The keys to a fast module are usually the structures that represent its data and the algorithms that operate on the data. The largest single improvement in Appel's program came from replacing an $O(n^2)$ algorithm with an $O(n \log n)$ algorithm; Columns 2 and 8 describe similar speedups.

Code Tuning. Appel achieved a speedup factor of five by making small changes to code; Column 9 is devoted to that topic.

System Software. Sometimes it is easier to change the software on which a system is built than the system itself. Is a new database system faster for the queries that arise in this system? Would a different operating system be better suited to the real-time constraints of this task? Are all possible compiler optimizations enabled?

Hardware. Faster hardware can increase performance. General-purpose computers are usually fast enough; speedups are available through faster clock speeds on the same processor or multiprocessors. Sound cards, video accelerators and other cards offload work from the central processor onto small, fast, special-purpose processors; game designers are notorious for using those devices for clever speedups. Special-purpose digital signal processors (DSPs), for example, enable inexpensive toys and household appliances to talk. Appel's solution of adding a floating point accelerator to the existing machine was somewhere between the two extremes.

6.3 Principles

Because an ounce of prevention is worth a pound of cure, we should keep in mind an observation Gordon Bell made when he was designing computers for the Digital Equipment Corporation.

> *The cheapest, fastest and most reliable components of a computer system are those that aren't there.*

Those missing components are also the most accurate (they never make mistakes), the most secure (they can't be broken into), and the easiest to design, document, test and maintain. The importance of a simple design can't be overemphasized.

But when performance problems can't be sidestepped, thinking about design levels can help focus a programmer's effort.

If you need a little speedup, work at the best level. Most programmers have their own knee-jerk response to efficiency: "change algorithms" or "tune the queueing discipline" spring quickly to some lips. Before you decide to work at any given level, consider all possible levels and choose the one that delivers the most speedup for the least effort.

If you need a big speedup, work at many levels. Enormous speedups like Appel's are achieved only by attacking a problem on several different fronts, and they usually take a great deal of effort. When changes on one level are independent of changes on other levels (as they often, but not always, are), the various speedups multiply.

Columns 7, 8 and 9 discuss speedups at three different design levels; keep perspective as you consider the individual speedups.

6.4 Problems

1. Assume that computers are now 1000 times faster than when Appel did his experiments. Using the same total computing time (about a day), how will the problem size n increase for the $O(n^2)$ and $O(n \log n)$ algorithms?
2. Discuss speedups at various design levels for some of the following problems: factoring 500-digit integers, Fourier analysis, simulating VLSI circuits, and searching a large text file for a given string. Discuss the dependencies of the proposed speedups.
3. Appel found that changing from double-precision arithmetic to single-precision arithmetic doubled the speed of his program. Choose an appropriate test and measure that speedup on your system.
4. This column concentrates on run-time efficiency. Other common measures of performance include fault-tolerance, reliability, security, cost, cost/performance ratio, accuracy, and robustness to user error. Discuss how each of these problems can be attacked at several design levels.

5. Discuss the costs of employing state-of-the-art technologies at the various design levels. Include all relevant measures of cost, including development time (calendar and personnel), maintainability and dollar cost.
6. An old and popular saying claims that "efficiency is secondary to correctness — a program's speed is immaterial if its answers are wrong". True or false?
7. Discuss how problems in everyday life, such as injuries suffered in automobile accidents, can be addressed at different levels.

6.5 Further Reading

Butler Lampson's "Hints for Computer System Design" appeared in *IEEE Software 1*, 1, January 1984. Many of the hints deal with performance; his paper is particularly strong at integrated hardware-software system design. As this book goes to press, a copy of the paper is available at *www.research.microsoft.com/~lampson/*.

COLUMN 7: THE BACK OF THE ENVELOPE

It was in the middle of a fascinating conversation on software engineering that Bob Martin asked me, "How much water flows out of the Mississippi River in a day?" Because I had found his comments up to that point deeply insightful, I politely stifled my true response and said, "Pardon me?" When he asked again I realized that I had no choice but to humor the poor fellow, who had obviously cracked under the pressures of running a large software shop.

My response went something like this. I figured that near its mouth the river was about a mile wide and maybe twenty feet deep (or about one two-hundred-and-fiftieth of a mile). I guessed that the rate of flow was five miles an hour, or a hundred and twenty miles per day. Multiplying

$$1 \text{ mile} \times 1/250 \text{ mile} \times 120 \text{ miles/day} \approx 1/2 \text{ mile}^3/\text{day}$$

showed that the river discharged about half a cubic mile of water per day, to within an order of magnitude. But so what?

At that point Martin picked up from his desk a proposal for the communication system that his organization was building for the Summer Olympic games, and went through a similar sequence of calculations. He estimated one key parameter as we spoke by measuring the time required to send himself a one-character piece of mail. The rest of his numbers were straight from the proposal and therefore quite precise. His calculations were just as simple as those about the Mississippi River and much more revealing. They showed that, under generous assumptions, the proposed system could work only if there were at least a hundred and twenty seconds in each minute. He had sent the design back to the drawing board the previous day. (The conversation took place about a year before the event, and the final system was used during the Olympics without a hitch.)

That was Bob Martin's wonderful (if eccentric) way of introducing the engineering technique of "back-of-the-envelope" calculations. The idea is standard fare in engineering schools and is bread and butter for most practicing engineers. Unfortunately, it is too often neglected in computing.

7.1 Basic Skills

These reminders can be helpful in making back-of-the-envelope calculations.

Two Answers Are Better Than One. When I asked Peter Weinberger how much water flows out of the Mississippi per day, he responded, "As much as flows in." He then estimated that the Mississippi basin was about 1000 by 1000 miles, and that the annual runoff from rainfall there was about one foot (or one five-thousandth of a mile). That gives

$$1000 \text{ miles} \times 1000 \text{ miles} \times 1/5000 \text{ mile/year} \approx 200 \text{ miles}^3\text{/year}$$

$$200 \text{ miles}^3\text{/year} \,/\, 400 \text{ days/year} \approx 1/2 \text{ mile}^3\text{/day}$$

or a little more than half a cubic mile per day. It's important to double check all calculations, and especially so for quick ones.

As a cheating triple check, an almanac reported that the river's discharge is 640,000 cubic feet per second. Working from that gives

$$640{,}000 \text{ ft}^3\text{/sec} \times 3600 \text{ secs/hr} \approx 2.3\times10^9 \text{ ft}^3\text{/hr}$$

$$2.3\times10^9 \text{ ft}^3\text{/hr} \times 24 \text{ hrs/day} \approx 6\times10^{10} \text{ ft}^3\text{/day}$$

$$6\times10^{10} \text{ ft}^3\text{/day} \,/\, (5000 \text{ ft/mile})^3 \approx 6\times10^{10} \text{ ft}^3\text{/day} \,/\, (125\times10^9 \text{ ft}^3\text{/mile}^3)$$

$$\approx 60/125 \text{ mile}^3\text{/day}$$

$$\approx 1/2 \text{ mile}^3\text{/day}$$

The proximity of the two estimates to one another, and especially to the almanac's answer, is a fine example of sheer dumb luck.

Quick Checks. Polya devotes three pages of his *How to Solve It* to "Test by Dimension", which he describes as a "well-known, quick and efficient means to check geometrical or physical formulas". The first rule is that the dimensions in a sum must be the same, which is in turn the dimension of the sum — you can add feet together to get feet, but you can't add seconds to pounds. The second rule is that the dimension of a product is the product of the dimensions. The examples above obey both rules; multiplying

$$(\text{miles} + \text{miles}) \times \text{miles} \times \text{miles/day} = \text{miles}^3\text{/day}$$

has the right form, apart from any constants.

A simple table can help you keep track of dimensions in complicated expressions like those above. To perform Weinberger's calculation, we first write down the three original factors.

1000 miles	1000 miles	1 mile
		5000 year

Next we simplify the expression by cancelling terms, which shows that the output is 200 miles^3/year.

~~1000~~	~~miles~~	~~1000~~	~~miles~~	~~1~~	~~mile~~	200	mile^3
				~~5000~~	year		

Now we multiply by the identity (well, almost) that there are 400 days per year.

~~1000~~	~~miles~~	~~1000~~	~~miles~~	~~1~~	~~mile~~	200	mile^3		year
				~~5000~~	year			400	days

Cancellation yields the (by now familiar) answer of half a cubic mile per day.

~~1000~~	~~miles~~	~~1000~~	~~miles~~	~~1~~	~~mile~~	~~200~~	mile^3		~~year~~	1
				~~5000~~	~~year~~			~~400~~	days	2

These tabular calculations help you keep track of dimensions.

Dimension tests check the form of equations. Check your multiplications and divisions with an old trick from slide rule days: independently compute the leading digit and the exponent. One can make several quick checks for addition.

3142	3142	3142
2718	2718	2718
+1123	+1123	+1123
983	6982	6973

The first sum has too few digits and the second sum errs in the least significant digit. The technique of "casting out nines" reveals the error in the third example: the digits in the summands sum to 8 modulo 9, while those in the answer sum to 7 modulo 9. In a correct addition, the sums of the digits are equal after "casting out" groups of digits that sum to nine.

Above all, don't forget common sense: be suspicious of any calculations that show that the Mississippi River discharges 100 gallons of water per day.

Rules of Thumb. I first learned the "Rule of 72" in a course on accounting. Assume that you invest a sum of money for y years at an interest rate of r percent per year. The financial version of the rule says that if $r \times y = 72$, then your money will roughly double. The approximation is quite accurate: investing \$1000 at 6 percent interest for 12 years gives \$2012, and \$1000 at 8 percent for 9 years gives \$1999.

The Rule of 72 is handy for estimating the growth of any exponential process. If a bacterial colony in a dish grows at the rate of three percent per hour, then it doubles in size every day. And doubling brings programmers back to familiar rules of thumb: because $2^{10} = 1024$, ten doublings is about a thousand, twenty doublings is about a million, and thirty doublings is about a billion.

Suppose that an exponential program takes ten seconds to solve a problem of size $n = 40$, and that increasing n by one increases the run time by 12 percent (we probably