

UCBTEST vs. Pentium

David Hough

ABSTRACT

University of California research into systematic verification of computer arithmetic has been put into a form in which anybody could in principle test any computer arithmetic.

So it's exceedingly unlikely that another microprocessor will come to market with bugs that UCBTEST software might detect, or with uninitialized entries in a divider's PLA. But to avoid the situation of generals always fighting the last war, what can be done to detect, or better prevent, the **next** arithmetic flaw?

Number Theory in Action

In the summer of 1988, Prof. W. Kahan of the University of California, Berkeley, presented a series of Floating-Point Indoctrination Lectures at Sun Microsystems in Mountain View, CA. As one offshoot of that lecture series, some of the attendees and some of Kahan's former students translated several unpublished number-theoretic methods for testing floating-point arithmetic into actual working programs. Programs were devised to test multiplication, division, and sqrt, by artfully generating test cases whose results are nearly halfway between representable numbers, and so are likely to be rounded incorrectly if the underlying arithmetic implementation is not provably correct according to ANSI/IEEE 754, the IEEE Standard for Binary Floating-Point Arithmetic.

Those programs were calibrated for a number of test cases that ran tolerably long on the Sun-3 systems then common, and were incorporated into Sun's automatic testing regimes for new hardware, compilers, and libraries.

In the fall of 1994, in the process of testing a new faster software implementation of quadruple-precision floating-point arithmetic, some bottlenecks in the arithmetic test programs were identified and eliminated, which in combination with the general rapid advance in computer technology, caused the test programs to run too quickly to be timed reliably under Unix. Increasing the number of test cases from ten thousand to one million seemed about right, so the source files were modified accordingly.

Meanwhile, news of the Pentium division flaw was migrating from computer bulletin boards to *EE Times* to the front page of the *San Jose Mercury News*. Prof. Kahan had served as a consultant to Intel over many years and his test methods must have been known there. Why hadn't they turned up the Pentium flaw years earlier?

Ironically, the single-precision division test applied to a flawed Pentium chip found evidence of the flaw after 750000 tests - and would have been overlooked at the previous level of 10000 tests.

UCBTEST

Since the Pentium crisis at its height cast a cloud of uncertainty over all current computer systems, Sun decided to make publicly available the multiplication, division, and sqrt test programs mentioned above. Send a message "send index from ucctest" to netlib@research.att.com for further details. In addition, versions of the infamous Paranoia test of computer arithmetic, the Berkeley Elementary Functions test programs, and several other computer arithmetic tests have been included in UCBTEST.

In principle, anybody could use these programs to test the arithmetic of a C or Fortran compiler. In practice, it's more complicated than that. By misapplying the programs it is possible to "discover"

all sorts of non-existent problems. Neither does a lack of apparent problems prove that all is well. But in conjunction with other methods, UCBTEST allows manufacturers to increase their confidence in their designs. And UCBTEST provides one of the few ways that end users, with no knowledge about the details of circuit design, may increase their confidence in manufacturer's claims about the computer systems they buy.

Confidence Games

How can one know that a computer circuit or software meets its specifications?

Exhaustive Tests

IEEE 754 single-precision functions of one single-precision operand can be tested exhaustively, with every possible operand value, usually against double precision, in an acceptable amount of time. Not everybody bothers to do this, since this approach is hopeless if there is more than one operand or double precision is involved.

Proofs

Proofs may be the most satisfactory method, if they can be found, if they can be simple enough to be understood, and if they apply fully to the problem at hand. It is not always possible to find a satisfactory proof. If a satisfactory proof seems to require modifying the design, does that reflect on the design or on the proof? Changing the design will delay coming to market - is that a fair price to pay for insuring against a problem that may not exist? Perhaps a more skillful analyst could find a subtler proof. But if the proof becomes more complicated than the algorithm, why believe it better; and what about the Pentium division algorithm, for which the "hard part" had been rigorously proved implicitly assuming a correctly initialized PLA - but the fault lay in an simple process that had not been so well scrutinized.

Random Tests

Random testing of inputs to computer arithmetic has its place, but it is a very uneconomical method of turning up problems, because the devil is usually in the details of what happens at various kinds of algorithmic boundaries, and points on the boundaries are a very small fraction of all points. Random testing of billions of operands didn't find the Pentium flaw.

Wrong Target Tests

Certain kinds of devious tests have a way of turning up problems that they were not even looking for. If you don't know what kind of problem you are looking for - the situation of most end users - all tests are probably pointed at the wrong target. But some verification is better than none. The UCBTEST programs may be in this category for many users. The UCBTEST division test, run in single-precision rounding mode, turns up the Pentium flaw in less than a minute. But run in the more usual double-precision or extended-precision rounding modes, the same program will not find the Pentium flaw in an amount of time you'd be willing to wait. This program was designed to test nearly halfway cases, corresponding to boundaries in the result, while the Pentium flaw was related to boundaries in the plane representing the input operands; that this program happened to find the flaw in single precision should be considered remarkable, not its failure to find any flaw in higher precision. It may be that Pentium flaw cases are "denser" somehow in single precision: does the lower density of single precision operands increase the chance that a nearly halfway result case is also on a critical operand boundary? Who would have thought of that in advance?

Targeted Tests

Ideally, given a particular algorithm, one would devise a specific test for its weak spots and run that. But that depends on being able to identify an algorithm's weak spots - did you think to check the PLA entries? Prof. Kahan is developing such a test program specifically for PLA-based dividers - but not specifically for Pentium - that finds the Pentium flaw in a fraction of a second. By the time such tests become available to end users, they are likely to be proving that last year's bug is not in this year's products - an outcome that the manufacturers have usually taken care to prevent.

Realistic Application Tests

The real proof is in the running of the realistic applications that people buy computers to run, so another approach is to test a large variety of realistic applications with massive amounts of real data, and compare the results to known good results or apply internal self-consistency checks. These programs were never intended to be arithmetic tests, or hardware or compiler or library tests, yet some prove remarkably effective at turning up problems. Even so, large numbers of such tests failed to notice the Pentium flaw. The Pentium flaw was discovered by Prof. Nicely after his program's internal self-consistency checks failed. Millions of end users insure that even a very rare Pentium flaw will eventually be found. The flaws of systems with fewer users may evade detection for the life of those products. And unfortunately some manufacturers have not outgrown some of Intel's first approaches to the Pentium problem: keep problems secret in the hope they'll never be discovered, or if found out, argue that the problems don't matter, or all systems have flaws, or manufacturers don't have an ongoing responsibility to insure that previously shipped products meet previously published specifications. These approaches sometimes work in the technical market, but graduation to the mass consumer market carries additional risks along with its rewards, as Intel discovered.

Yet taken together, the diverse test methods outlined above reinforce each other, and one measure of the quality of an arithmetic implementation, or any other hardware and software, is how many different tests have been applied to it by the manufacturer and by a number of external testers with different points of view. But having ten people run the same tests ten times adds little information. And that leads to the larger question: since computer product lifetimes seldom exceed two years, and exhaustive testing can't be completed in that period of time, how long should testing be conducted before a product is released to production? The UCBTEST division program might find the Pentium flaw with double-precision rounding if allowed to run for a week - or a month - or a year, perhaps, but who knows for sure?

Independent Confirmation

Being perfectly reproducible and context-free, the Pentium flaw is anachronistic. Far more typical these days are bugs that are practically irreproducible because they require very rare alignments of apparently unrelated events. This is not as much a problem for batch computations, which can be repeated, as for real-time control systems. But in either case the final conclusion is the same: any kind of decision should be confirmed by independent means. In the most important cases of decisions made with the aid of computers, "independent means" may need to encompass a different machine, with a different instruction set, running a different operating system, with a program written in a different computer language, implementing a different algorithm; all done by a different investigator on another continent who thinks in a different natural language.

Thus when writing a program to test base conversion, for instance, **at least** take care to use test algorithms utterly different from any likely to be used in the base conversion functions to be tested. So even slow, simple algorithms have their place in Computer Science - for testing fast complicated algorithms.

Cost of Testing

The Pentium bug may cost Intel, its system-vendor customers, and their end-user customers more money than the entire cumulative budget of the UCB EECS department since its founding. Yet on the whole, the high-tech industry centered in California seems curiously indifferent to the decline of the state's system of public education. Organizations interested in funding further work in computer arithmetic testing and verification might contact the Berkeley Engineering Fund in the College of Engineering.