wdv-notes 334
Stand: 22.JAN.1995 (3.)
Wiss.Datenverarbeitung
FREIE UNIVERSITÄT BERLIN

The Frequency and Propagation of Errors Caused by the Pentium FDIV Defect.

Theo

Das Pentium-Desaster der Computerindustrie [1–3] wurde schon bald nach seinem Bekanntwerden mit dem Super-GAU des Kernkraftwerks Tschernobyl von 1986 verglichen [u. a. 2, 4]. Das hat Kritik hervorgerufen.

Der Tschernobyl-Vergleich gilt natürlich nicht im streng physikalischen Sinn, sondern bezieht sich auf die Machbarkeit, die Testbarkeit, die Qualität von komplexen technischen Produkten, und natürlich auch auf die Glaubwürdigkeit der Aussagen der produzierenden Industrie.

Professor Pratt vom Computer Science Dept. der Stanford University konnte nun aber auch zeigen, daß sogar eine direkte Analogie zu einer außer Kontrolle geratenen Kettenreaktion besteht. In einem sehr eindrucksvollen Poster (publiziert in *comp.sys.intel, sci.math.num-analysis* und *comp.arch* [5]) demonstrierte er an einem Modell (Programm), daß der Pentium-Defekt – entgegen allen Prognosen von Intel [6] – die Daten einer einfachen Datenbank bereits nach 280 Modellzyklen – *nach nur 32 Minuten* – bis zur völligen Unbrauchbarkeit zerstört hat.

Der Pentium-Hersteller Intel war selbst bisher nicht in der Lage eine solche wegweisende Analyse abzuliefern. –

Chain Reaction in Pentiums

By VAUGHAN R. PRATT
Stanford University
pratt@cs.stanford.edu
Posted to UseNet (Internet) – Dec 30, 1994

We give an empirically determined upper bound on the time it would take a chain reaction in a critical mass of Pentiums to melt down a database, together with the program used to determine it.

In the physical sense it is absurd to compare the Pentium with Chernobyl. I've been sitting beside mine for weeks; had it been remotely comparable to Chernobyl I wouldn't be able to lift a finger to type.

Of course that's not what Henry had in mind [5], he meant a suitable analogy in the data processing world. What I found myself wondering was, just how broad does this analogy have to be in order to be apt? On investigating this I found that the analogy can be taken far more literally than even M. Lane presumably had in mind [5].

Chernobyl happened when its operators lost control of the reactor's chain reaction while conducting an ill-considered experiment. When the world's Pentiums ship each other flawed data, a natural question to ask is *whether the data processing equivalent of a sustainable and uncontrolled chain reaction is possible*. Continuing the analogy, is the Pentium's equivalent of a nuclear half-life (*the time between errors*) sufficiently short, and the decay products (*the error magnitudes*) sufficiently strong that a chain reaction in a sufficiently dense mass of Pentiums (relative to the number of participating non-Pentiums) could both start in a reasonable time and sustain itself?

Now we have seen many examples that trigger the bug with a probability greater than the official one in nine billion [6]. The most questionable examples are those based on specific quotients such as 4195835/3145727 and 4.999999/14.999999, or somewhat more generic computations such as x/(d.d–d.d); many have wondered just how often these examples are really encountered in everyday computation, while many others are convinced that they are quite likely enough to present a real risk. Also questionable are the examples that embed some understanding of the nonuniformities in the distribution of the bad operands, e.g. my small-bruised-integer measurements. Just how at-risk really is a typical program that is ignorant of those nonuniformities?

With these concerns in mind, I designed the following "lab experiment," with a Pentium as the lab. The design of the experiment is not at all specific to the Pentium, and is simple enough both to implement in a reasonable time and to at least partially understand its nature. Its purpose is to allow one to observe the rate at which errors appear, and to study their propagation.

The Model

The upshot was that errors appeared about a thousand times sooner than the random model predicts, and they propagated energetically enough to sustain a chain reaction essentially analogous to those taking place in atomic bombs and nuclear reactors.

The experiment simulated a computational world of indeterminate scope, one that could correspond either to a distributed database run by many computers or a centralized one run by one computer. This database's knowledge is obtained partly from "axioms" or "sensory input", i.e. clean data from outside the scope of the experiment, and partly by "deduction," i.e. computation performed within its scope. The latter is where the Pentium-induced damage arises.

I had originally considered taking the "sensory input" to consist of financial amounts such as $9.98. However the more complicated the source the more suspect it becomes in terms of reflecting a possible bias towards Pentium-bad numbers. So in the end I settled for a constant incoming stream of 1's, on the theory that if the number 1 all by itself is Pentium-bad, things are much worse than we thought!

"Deduction" was simply the inference of new numbers from previously obtained ones via arithmetic operations chosen at random. All divisions were by flawed Pentium, corresponding to all computers participating in the maintenance of this database being such. I did not investigate in this experiment what lesser density of Pentiums might still constitute a critical mass for a chain reaction.

The Database

In more detail, I first set up a numeric database consisting of a single number, namely 1. I then set a flawed Pentium (not mine, it's been fixed) to work adding new numbers to this database. These numbers came from two sources. The first source was the database itself; two numbers were selected at random from the database, then combined by an operation randomly selected from among the four operations of addition, subtraction, multiplication, and division, then stored in the database. The second source was a stream of 1's, constituting "fresh data" moderating any chain reaction that might start up like a graphite rod in a reactor core. These sources were weighted equally for the groups plus-minus, times-divide, and the ones, one third each. This was achieved simply by viewing these as six sources: four for the arithmetic operations and two for the stream of 1's treated as a pair of such streams. These six sources were sampled randomly, more precisely pseudorandomly with a uniform distribution using the Unix library integer rand() function.

I did not want the database to fill up with improbably large or tiny numbers, whose absurd sizes might raise eyebrows; on the other hand I also did not want an artificial cliff in the distribution, whose precise location might raise questions. I therefore wrote a macro ablg(x) to compute (int)(fabs(log_2(x)/32)) fast and used it in ((rnd = (rnd%15)+1)%(ablg(cl)+ablg(di)+1) == 0) as a fast stochastic filter of high-magnitude-exponent numbers whose effect is first felt on positive numbers lying outside $[2^{-31}, 2^{32}]$ (and symmetrically for negative numbers). This filter rolls off gently, with probability 1/3 of retaining a number in $[2^{32}, 2^{96}]$ (and symmetrically in $[2^{-95}, 2^{-31}]$), down to 1/15 for $[2^{416}, 2^{480}]$ (in general 1/(2n+1) for $[2^{32(2n-1)}, 2^{32(2n+1)}]$ for n=1,...,7), and probability 0 thereafter (via the %15).

After the database fills up, the program creates space for each new number by overwriting a randomly selected old number. Although I have 32 Mb, I set the storage limit to 750,000 double-precision numbers (.75*8*2 = 12 Mbytes, there being two copies of the database) so that the program would run on 16 Mb machines without swapping (the accesses are highly nonlocal and would cause thrashing if swapping happened).

To measure errors I ran two such databases, clean and dirty, in lockstep, performing the same operations on both using the same operand addresses and making all decisions identically for both (e.g. the above stochastic

reactor.c

Das von V. R. Pratt für diese Analyse geschriebene Quellenprogramm *reactor.c* findet man nun unter *http://www.khd-research.net/Tech/Computer/PBug/pbug_chain_react.txt*.

Weitere Analysen von V. Pratt sind auf *boole.stanford.edu* publiziert.

Die 1. Edition erschien am 31. Dezember 1994.

filter resolves disagreement between the clean and dirty numbers by in effect taking their geometric mean). The only difference is that clean division uses the Mathworks' Matlab solution. (For this sort of work a dirty Pentium is much faster than a clean one because it is much faster to emulate clean division on a dirty Pentium than vice versa.)

Before giving the results of this experiment it is worth considering what might plausibly have been expected. Because *no effort was made to produce Pentium-sensitive numbers* of the kind familiar from the IBM report and my small-bruised-integer response functions [bug1 and bug2 in */pub/FDIV/individual.bugs* on *boole.stanford.edu*], one might expect only one out of every 9 billion divisions [6], and hence one out of every 6*9 = 54 billion numbers produced, would be in error. One would therefore expect to wait about four days for the first error, and only then would the interesting question arise as to whether that error would have time to clone itself sufficiently before it and its first few clones were all overwritten by the steady inbound flow of clean data. Even if the error survived, its expected size would be very small: only one in every 40 or so errors is in the 5th decimal digit, and hence occurs only every 160 days at full speed (or every 40*27,000 ≈ one million years at the 1000 divisions per day rate).

The Results

In fact what happened was that my database effectively "melted down" altogether in 280 "cycles" taking 32 minutes, with 20% of its 750,000 numbers having a relative error more than 0.1 [Ed. Note: > 10%!] and more than 6% having a relative error more than 1 (i.e. meaningless). Although a few more cycles would have seen yet further deterioration, my experiment was programmed to stop at that point: enough is enough!

A "cycle" is defined here as the arrival of just enough numbers to fill the database, less a few filtered-out oversize numbers. On an isolated Pentium a cycle thus consists of 750,000 consecutive operations (including 1's), most of which put a number into the database. My P90 gets through a cycle in 7 seconds; were it not for other issues like locality and propagation delays a network of a thousand Pentiums would accomplish a cycle in 7 milliseconds.

Melt-down *took 32 minutes* on my machine, but if the same operations were to be performed in parallel it could occur in seconds in a sufficiently large and fast network of Pentiums. (Ordinarily a sequential machine can create long sequential dependencies that would invalidate this reasoning by creating long error propagation paths that parallel computers would not have time to develop, but the random access pattern of this reactor creates dependency chains of an expected length sufficiently low as to be consistent with those arising in much more parallel computation, usually an essential component of "embarrassingly parallelizable.")

The Melt-down

Here is the printout detailing the melt-down. The vertical axis is time, increasing downwards linearly; each line is printed after 20 cycles (so line 9 comes at the end of cycle 180). The horizontal axis is space: each of the 50 or so dots on a line denotes 2% of the database, the whole space being ordered left to right by decreasing relative error, with hexadecimal digits denoting precision boundaries: 6 for 10^{-6}, c for 10^{-12}, etc.

```
 1  0123456789a.b.c.d.............................................
 2  0123456789.a.b.c.d............................................
 3  012345.6.7.8.9.a.b.c.d.......................................
 4  012345.6.7.8.9.a.b...c...d....................................
 5  012345.6.7.8.9..a....b.......c....d............................
 6  0123.4.5.6.7.8...9.....a......b.....c..d.......................
 7  012.3.4.5.6.7...8........9......a....b..c.d.....................
 8  012.3.4.5..6....7.......8.......9...a..b.c.d....................
 9  .0.1.2.3.4..5......6.......7.....8...9.a.b.c.d..................
10  .0.1.2.3..4.....5........6....7..8.9.a.b.c.d.....................
11  .0.1.2...3......4.......5.....6..7.8.9.a.b.c.d...................
12  .0.1...2........3......4....5..6.7.8.9.a.b.c.d...................
13  .0....1......2........3...4..5.6.7.8.9.a.b.c.d...................
14  ....0........1......2....3..4.5.6.7.8.9.a.b.c.d...................
```

The dots immediately following a hex digit or at the start of the line actually represent anywhere from a single entry to 2% of the database (whence the slight line-to-line variations in number of dots), the other dots represent exactly 2%. On line 6, the three dots between 8 and 9 indicate that between 4% and 6% of the database has relative errors between 10^{-8} and 10^{-9}, while the single dot between 3 and 4 on line 6 means that there is at least one error in the range 10^{-3} down to 10^{-4}. Absence of a dot, e.g. between 2 and 3 on line 6, means no errors at all in the range 10^{-2} to 10^{-3}. Dots to the right of d denote errors less than 10^{-13}, including correct results. Because 1/3 of the incoming data consists of 1's, which are automatically correct, this region does not shrink below 1/3 of the whole database, which is why d slows down instead of going all the way across the page.

The boundaries start out squished together in the upper left. Each boundary takes turns rushing across the no-man's land in the middle, to join its comrades huddling together in the lower right. In terms of error populations this means that the correctness of the database deteriorates explosively; errors blow up by an order of magnitude every twenty cycles or so.

The experiment stops when more than 2% of the data has unit relative error (is meaningless) [Ed. Note: = 100%]; here it had reached 6–8% by the time this stopping condition was examined, at which point at least 20% of the data base had relative errors above 0.1. Had it not been stopped the error explosion would have continued at the same speed and the data base would in a few dozen more cycles be a mixture of 1's and completely meaningless numbers, total melt-down.

Caveats

There are several important caveats in interpreting these results. The most basic caveat is that any chain reaction, whether physical or computational, is highly sensitive to the prevailing conditions. Thus small changes to various parameters may increase or decrease both the speed of onset of the reaction and its ability to sustain itself. In particular I would not expect it would take very much of a dilution, in the form of either non-Pentiums or clean Pentiums contributing to the process, or a different mix of incoming data and operations performed on the data, to moderate this reaction to below the level at which it can sustain itself. Thus few if any real-world computing environments would be likely to provide the right conditions for this chain reaction. With the large number and variety of today's environments however the possibility that some might be at some risk of "going critical" should be kept in mind.

Offsetting these reassuring considerations is the less reassuring possibility that the incoming data may include either many small dollar amounts, shown by the IBM work to be at well above average risk on the Pentium, or my explicitly bruised integers, e.g. rationals as decimals truncated by mandate, or data read from a decimal calculator showing 4.999999 or from an analog-to-digital device that is positioned almost on an integer. When the stream of 1's is replaced by such Pentium-sensitive data the onset of a chain reaction could conceivably be much faster, and/or the critical mass of Pentiums (as measured by the rate of flawed divisions) required to sustain it could conceivably be much lower.

All these variations almost certainly will have some effect. How much awaits further experimental investigation, as does the question of which variations have any likelihood of arising in real computing environments.

As an independent application, by avoiding Pentium-specific knowledge the experiment might make a useful general-purpose FPU test [Ed. Note: Hello Intel, it's possible!] when left to run for a week, provided its "clean" source is obtained from a reference machine.

The Program

Here is the program [Ed. Note: See box on page 1], which may be freely distributed and modified without my permission. It runs under Linux but should require little or no modification for other OS's (on a Pentium of course). It would ease my mind a bit if when you hack it up you put your name, date, and hack at the top, in reverse chronological order, so I get neither the credit nor the blame due you. □

Literatur

Viele der im folgenden aufgezählten Materialien sind auch elektronisch publiziert und stehen als Dateien (Files) auf dem Ftp-Server *ftp.grumed.fu-berlin.de* im Verzeichnis *PC* zum Kopieren via Internet zur Verfügung.

[1] Dittberner, K.-H. (Ed.): „Intel Pentium – The Chip that Redefines Mathematics". FU Berlin (IfP): wdv-notes Nr. 307, 1994–1995.

[2] Dittberner, K.-H.: Intel Pentium – Der eingebaute Divisionsfehler. FU Berlin (IfP): wdv-notes Nr. 327, 1994–1995.

[3] Dittberner, K.-H. (Ed.): Intel Pentium – Questions and (some) answers. FU Berlin (IfP): wdv-notes Nr. 330, 1994–1995.

[4] Computer: Nur fünf Stellen. DER SPIEGEL, Nr. 51, 19. Dez. 1994, Seite 87–88.

[5] Pratt, Vaughan R.: Chain reaction in Pentiums. Published on UseNet, 30. Dec. 1994. – File: PBUG_CHAIN_REACT.TXT.

[6] Sharangpanl, H. P. and Barton, M. L.: Statistical Analysis of Floating Point Flaw in the Pentium Processor (1994). Santa Clara (USA): Intel White Paper, 30. Nov. 1994. – File: INTELS_WHITE_PAPER.PS.ZIP.