

Re: A chip too far? Where is your solution Mr Larkin?

Re: A chip too far? Where is your solution Mr Larkin?

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2008-08/msg03373.html>

- *From:* Martin Brown <|||newspam|||@nezumi.demon.co.uk>
 - *Date:* Tue, 19 Aug 2008 20:53:38 +0100
-

John Larkin wrote:

On Tue, 19 Aug 2008 17:23:58 +0100, Martin Brown
<|||newspam|||@nezumi.demon.co.uk> wrote:

John Larkin wrote:

On Tue, 19 Aug 2008 15:07:51 GMT, Jan Panteltje
<pNaonStpealmtje@xxxxxxxxxx> wrote:

A chip too far? Where is your solution Mr
Larkin?

http://money.cnn.com/2008/08/13/technology/microchips_copeland.fortune/index.htm

Blowing in the wind, all blowing in the
wind.

1. A new programming language, accompanied with a new way of teaching programming.

That might help a bit. But there are a lot of very important legacy applications already out there. Like it or not we are tied into the existing applications codebase.

2. A new OS, which uses a nanokernel approach and wastes processors in the name of reliability.

Can't fault the idea of keeping the Ring0 protected kernel as small as possible, or giving users the least privileges needed to do their job. But wasting performance for reliability though is never going to fly – not least because most times it would not have the desired effect.

Hardware enforced memory protection for threads on a timesliced CPU can be made every bit as reliable as giving each one a physical CPU just a

Re: A chip too far? Where is your solution Mr Larkin?

Re: A chip too far? Where is your solution Mr Larkin?

SMOP.

3. Ultimately, a new multicore cpu architecture that exerts much more hardware control over the things that programmers tend to screw up.

You mean like enforcing bounds checking on array and pointer access?

Pointers have to be the first thing to go. And of course array bounds should be checked, but all that can do is crash a program, which was going to crash sooner or later anyhow.

Pointers are OK provided you can only trash your own memory space. Anything else and you should get squashed flat.

We are programming antique hardware in antique languages.

Speak for yourself. I program in the language that best fits the problem (or is dictated by the clients requirements). Sometimes both. The fastest way to develop really tricky algorithms is in a language where human errors and typos are mostly caught at compile time. The final version is then translated back into C (which is mostly what people seem to want). ISTR NAGLIB used this approach a very long time ago (when FORTRAN was the main language of choice for scientific computing).

I wish that a more robust language had won the language wars, but it is too late now to cry over spilt milk.

It will take a long time to fix things, because the current computer culture – academics, programmers, cpu makers, Microsoft – will fight for its survival.

Academics are a lot more sanguine about things than you seem to think.

The ideal programming language would be wordy, plodding, not a bit exciting – sort of like Cobol – and no fun for research or for coding.

In the sense that the compiler protects the average user from themselves I have to agree with you. But you actually do need really good visualisation skills and a logical mathematical mind to produce reliable code. Good intentions are insufficient.

That's the point: we have to make programming not fun if we want it to be reliable. Modern programming attracts the wrong kind of people to

Re: A chip too far? Where is your solution Mr Larkin?

Re: A chip too far? Where is your solution Mr Larkin?

teach and do reliable programming.

Even that isn't clear. I do have some sympathy for your argument that hardware engineers in general tend to be better at thinking through the details, unit testing and planning. But nobody wants to pay for truly reliable software when the law of diminishing returns sets in. You might want to take a look at Donald Knuths literate programming model used for TeX. It has one of the lowest bug rates of any program ever written.

And hardware implementation isn't really as robust as you like to pretend with your rose tinted glasses on. Various CPUs have had nasty bugs the worst one costing Intel a cool \$0.5B (despite being largely irrelevant for most users). I had a specimen dodgy PII – it even made a tiny difference for one regression test.

So far no-one has produced a full CPU with a formal proof of correctness. Components have been stratin with the Cyrix x87 copro. ISTR some bad liability lawsuits resulted from the Viper CPU debacle.

<http://www.cs.berkeley.edu/~gporter/pubs/roleofproof.html>

Most that I know believe the market gets what the market deserves. If people will buy vacuous buggy software with a cute interface that crashes when the wind changes direction then manufacturers will oblige.

When later versions of programs are slower and buggier, "the market" will oblige by not upgrading. We have seen this here a few times – a

Not enough to change the market dynamics – at least so far.

new rev is worse than the current one – so we drop support and stick with the one that works. Or we go to a free equivalent, like PDF utilities or LT Spice, and dump the bloatware.

XL2007 is a case in point. Slower, unreliable and in places downright flakey. I would not even recommend it to my worst enemy, but some of my clients insist on having the latest and greatest despite being given clear advice to avoid. XL2003 is a local optimum, as was NT3.51 – even a chaotic process can sometimes produce real gems...

Intel at least has had the good sense not to deploy Vista. YMMV

I notice that you quietly editted out all references to the ill fated Transputer hardware and its naturally parallel Occam language. That was a serious early attempt to adopt your world model. It failed big time.

Regards,

Martin Brown

** Posted from <http://www.teranews.com> **

.

Re: A chip too far? Where is your solution Mr Larkin?