

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/msg03353.html>

---

- *From:* John Larkin <[jjlarkin@xx](mailto:jjlarkin@xx)>
  - *Date:* Tue, 19 Aug 2008 11:22:54 -0700
- 

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

John Larkin wrote:

On Tue, 19 Aug 2008 15:07:51 GMT, Jan Panteltje  
<[pNaonStpealmtje@xxxxxxxxxx](mailto: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?postve](http://money.cnn.com/2008/08/13/technology/microchips_copeland.fortune/index.htm?postve)

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 SMOP.

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

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.

We are programming antique hardware in antique languages.

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. 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 teach and do reliable programming.

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 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.

John

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