



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

Larkin?

[http://money.cnn.com/2008/08/13/technology/microchips\\_co](http://money.cnn.com/2008/08/13/technology/microchips_co)

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 –

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

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.

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

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

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.

So, a hundred years from now, everybody will still be programming in C++? Writing billion-line programs with millions of bugs?

I didn't say that at all. Hand written code has a worryingly high defect rate of about 0.2% (average) and as high as 5% worst case (IBM OS/360 study). So either you have to make every line do more by improving the languages to match specific application domains or weed out the human errors sooner by more sophisticated static code analysis.

Almost every language from the birth of computing with a significant codebase is still going strong more than half a century later. The languages have evolved a bit and standards committees have haggled over the small print but they still exist. And they are still used.

Cobol, Fortran, Algol, Basic, Lisp all have their roots in the late 1950's.

I fully expect new kids on the block like Java and SQL and the various web markup languages to last at least a similar length of time.

Something to look forward to.

If I had to bet how software engineering will evolve in the future then I would say that formal specifications will become more common for mission critical work. Z and VDM that we have now are effective, but very few people can read or work with them and manipulate proofs.

Tools to support them and automatically convert the formal spec to a users view of how the interface will look and from that a conversion to functional code. Every path also has to handle changes and allow proof of correctness to be applied back to the specification. There are research projects trying to do this. It is distinctly non-trivial.

And yet again you quietly snip away the embarrassing failure of the Transputer and Occam which followed your proposed massively parallel

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

hardware route though expensively and somewhat ahead of its time.

I have not proposed a "massively parallel hardware route". I have observed that most CPU makers are going multicore, and that hundreds of cores, and maybe 1000 threads, will be available soon. And I have speculated on how that might affect OS design and programming. What's remarkable to me is how entrenched the current bloated and broken programming models are in the minds of programmers. Virtually everyone has responded to my speculations by telling me how good things are now, and how things will not change. So much for "design."

As far as failed architectures go, lots of good stuff didn't make it. 68K is mostly gone. PPC ditto, as far as mainstream apps go. VAX and Alpha were wonderful. Modern structured languages used to be the rage. All have been blown away by Intel and Microsoft, x86 and Windows and C, the worst of the litter.

John

.