Re: How to develop a random number generation device

Source: http://sci.tech−archive.net/Archive/sci.electronics.design/2007−09/msg03261.html

- *From*: John Larkin <jjlarkin@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
- *Date*: Mon, 17 Sep 2007 13:17:16 −0700

On Mon, 17 Sep 2007 14:54:38 −0500, Vladimir Vassilevsky
<antispam_bogus@xxxxxxxxxxx> wrote:

> John Larkin wrote:
>
> > My point is that large numbers of CPU cores
> > *will* become common and
> > cheap, and we need a new type of OS to take
> > advantage of this new
> > reality. Done right, it could be simple and
> > astoundingly secure and
> > reliable.
>
> Dear John,
> Try developing a perfect OS of your own. I did. That was a very
> enlightening experience of why certain things have to be done by the
> certain ways. Particular questions welcome.

I did write three RTOS's, one for the 6800, one for the PDP−11, one for the LSI−11. As far as I know, they were perfect, in that they ran damned fast and had no bugs. The 6800 version included a token ring LAN thing, which I invented independently in about 1974.

> > Well, I remember 64−bit static rams, and 256−bit DRAMS. I can't see
> > any reason we couldn't have 256 or 1024 cpu's on a chip, especially if
> > a lot of them are simple integer RISC machines.

> 1024 CPUs = 1048576 software interfaces and a hell of the bus arbitration.

No worse a software interface than if each process was running on a single shared CPU; much less, in fact, since irrevelant interrupts, swapping, and context switches aren't going on. Each process absolutely owns a CPU and only interacts with other processes when *it* needs to, probably through shared memory and semaphores.

As far as bus arbitration goes, they all just share a central cache on the chip, with a single bus going out to dram. Cache coherence becomes trivial.

> > I'd be happy to waste a little silicon if I could have an OS that
> > doesn't crash and that doesn't go to sleep for seconds at a time for
> > no obvious reason.
>
> The weak link is a developer. It is obviously more difficult to develop multicore stuff; hence it is a higher probability of flaws.

Putting a few hundred RISC cores on a chip, connecting to a central cache, is easy. You only have to get it right once. In our world, incredibly complex hardware just works, and modestly complex software is usually a bag of worms. Clearly we need the hardware to help the software.

> > Multiple cores gives absolutely no benefits in terms of reliability or
> > stability − indeed, it opens all sorts of possibilities for hard−to−debug race conditions.
>
> Especially if you remember about the 50−page silicon erratas for pretty much any modern CPU.

Intel, maybe. Are any of the RISC machines that bad? But my PC doesn't have hardware problems, it has software problems.

> > They don't if you insist on running a copy of a bloated OS on each. A system designed, from scratch, to run on a pool of cheap CPUs could be incredibly reliable.
>
> What do you think in particular would be better for a typical desktop applications?

Oh, 256 CPUs and, say, 32 FPUs should be plenty.

> > It's gonna happen.
>
> You have to listen to the screams of the SEL software developers...

Of course lots of software people won't like this. Well, they had their chance and blew it.

John

.