

# Re: How to develop a random number generation device

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg03703.html>

---

- *From:* David Brown <david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>
  - *Date:* Wed, 19 Sep 2007 23:06:17 +0200
- 

John Larkin wrote:

On Tue, 18 Sep 2007 18:15:07 +0200, David Brown  
<david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

John Larkin wrote:

On Mon, 17 Sep 2007 23:04:03 +0200, David Brown  
<david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

John Larkin wrote:

On Mon, 17 Sep 2007  
18:40:35 +0200, David  
Brown  
<david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>  
wrote:

John Larkin  
wrote:

On  
Sun,  
16  
Sep  
2007  
22:07:42  
+0200,  
David  
Brown  
<david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>  
wrote:

Re: How to develop a random number generation device

John  
Larkin  
wrote:

On  
Sun,  
16  
Sep  
2007  
11:33:21  
-0700,  
MooseFET  
<kensmith@xxxxxxxx>  
wrote:

On  
Sep  
15,  
11:09  
am,  
John  
Larkin  
<jjlar...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>  
wrote:  
[...]

architecture.  
In  
a  
few  
years  
we'll  
have,  
say,  
1024  
processors  
on  
a  
chip,  
and  
something  
new  
will  
be  
required  
to  
manage  
them.  
It  
will

Re: How to develop a random number generation device

be  
a  
thousand  
times  
simpler  
and  
more  
reliable  
than  
Windows.

I  
think  
that  
the  
number  
of  
virtual  
cores  
will  
grow  
faster  
than  
the  
number  
fo  
real  
cores.  
With  
extra  
register  
banks  
and  
a  
bit  
of  
clever  
design,  
a  
single  
ALU  
can  
look  
like  
two  
slightly  
slower  
ones.

I  
expect

Re: How to develop a random number generation device

to  
see  
multicore  
machines  
with  
less  
actual  
floating  
point  
ALUs  
than  
actual  
integer  
ALUs.

Sounds  
sort  
of  
like  
Sun's  
Niagra  
chips,  
which  
have  
(IIRC)  
8  
cores,  
each  
with  
4  
threads,  
but  
only  
a  
few  
floating  
point  
units.  
For  
things  
like  
web  
serving,  
it's  
ideal.

Yup.  
Low-horsepower  
tasks

Re: How to develop a random number generation device

can  
just  
be  
a  
thread  
on  
a  
multithread  
core,  
and  
many  
little  
tasks  
don't  
need  
a  
dedicated  
floating-point  
unit.

My  
point/fantasy  
is  
that  
OS  
design  
should  
change  
radically  
if  
many,  
many  
real  
or  
virtual  
CPUs  
are  
available.  
One  
CPU  
would  
be  
the  
manager,  
and  
every  
task,  
process,  
or  
driver  
could

Re: How to develop a random number generation device

have  
its  
own,  
totally  
confined  
and  
protected,  
CPU,  
and  
there  
would  
be  
no  
context  
switching  
ever,  
and  
few  
interrupts  
in  
fact.

That's  
not  
going  
to  
work  
for  
Linux,  
anyway  
–  
there  
is  
a  
utility  
thread  
spawned  
per  
cpu  
at  
the  
moment  
(work  
is  
underway  
to  
avoid  
this,  
because  
it

Re: How to develop a random number generation device

is  
a  
bit  
of  
a  
pain  
when  
you  
have  
thousands  
of  
cpus  
in  
one  
box).

However,  
there  
is  
no  
point  
in  
having  
a  
cpu  
(or  
even  
a  
virtual  
cpu)  
dedicated  
to  
each  
task.  
Many  
sorts  
of  
tasks  
spend  
a  
lot  
of  
time  
sleeping  
while  
waiting  
for  
other  
events  
–  
a

Re: How to develop a random number generation device

cpu  
in  
this  
state  
is  
a  
waste  
of  
resources.

Only  
if  
you  
think  
of  
a  
CPU  
as  
a  
valuable  
resource.  
As  
silicon  
shrinks,  
a  
CPU  
becomes  
a  
minor  
bit  
of  
real  
estate.  
It  
makes  
sense  
to  
use  
it  
when  
there's  
something  
to  
do,  
and  
put  
it  
to  
sleep  
when  
there's

## Re: How to develop a random number generation device

not.  
Lots  
of  
power  
gets  
saved  
by  
not  
doing  
context  
switches.

CPUs \*are\*  
a valuable  
resource –  
modern cpu  
cores take  
up a lot of  
space, even  
when you  
exclude  
things like  
the cache  
(which take  
more space,  
but cost less  
per mm<sup>2</sup>  
since you  
can design  
in a bit of  
redundancy  
and thus  
tolerate  
some  
faults).

The more  
CPUs you  
have, the  
more time  
and space it  
costs to  
keep caches  
and  
memory  
accesses  
coherent.  
There are  
some sorts  
of

## Re: How to develop a random number generation device

architectures  
which work  
well with  
multiple  
CPU cores,  
but these  
are not  
suitable for  
general  
purpose  
computing.

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

Re: How to develop a random number generation device

and  
reliable.

I would be very surprised to see a system where the number of CPU cores was greater than the number of processes. I expect to see the number of cores increase, especially for server systems, but I don't expect to see systems where it is planned and expected that most cores will sleep most of the time.

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.

You can certainly get 1024 CPUs on a chip – there are chips available today with hundreds of cores. But there are big questions about what you can do with such a device – they are specialised systems. To make use of something like that – you'd

## Re: How to develop a random number generation device

need a highly parallel problem (most desktop applications have trouble making good use of two cores – and it takes a really big web site or mail gateway to scale well beyond about 16 cores). You also have to consider the bandwidth to feed these cores, and be careful that there are no memory conflicts (since cache coherency does not scale well enough).

No, no, NO. You seem to be assuming that we'd use multiple cores the way Windows would use multiple cores. I'm not talking about solving big math problems; I'm talking about assigning one core to be a disk controller, one to do an Ethernet/stack interface, one to be a printer driver, one to be the GUI, one to run each user application, and one to be the system manager, the true tiny kernel and nothing else. Everything is dynamically loadable, unloadable, and restartable. If a core is underemployed, it sleeps or runs slower; who cares if transistors are wasted? This would not be a specialized system, it would be a perfectly general OS with applications, but no process would hog the machine, no process could crash anything else, and it would be fundamentally reliable.

That would be an absurd setup. There is some justification for wanting multiple simple cores in server systems (hence the Sun Niagara chips), but not for a desktop system. The requirements for a disk controller, a browser, and Doom are totally different. With a few fast cores like today's machines, combined with dedicated hardware (on the graphics card), you get a pretty good system that can handle any of these. With your system, you'd get a chip with a couple of cores running flat out (but without a hope of competing with a ten year old PC, as they could not have comparable bandwidth, cache, or computing resources in each core), along with a few hundred cores doing practically nothing. In fact, most of the cores would \*never\* be used – they are only there in case someone wants to do a few extra things at the same time since you need a core per process.

This is not about performance; hardly anybody needs gigaflops. It's all about reliability.

## Re: How to develop a random number generation device

Until you can come up with some sort of justification, however vague, as to why you think one cpu per process is more reliable than context switches, this whole discussion is useless.

You define yourself by the ideas you refuse to consider. So I suppose you'll still be running Windows 20 years from now.

I run windows (on desktops) and Linux (on a desktop, a laptop, and a bunch of servers, and on a fairly high-reliability automation system I am working on), and I'd use something else if I needed an OS in my embedded systems. If something better came along, I'd use that – whatever is the right tool for the job.

The relevant saying is "keep an open mind, but not so open that your brains fall out". I'm happy to accept that doing things in hardware is often more reliable than doing things in software (I work with small embedded systems – I know when reliability is important, and I know about achieving it in practical systems). But what I am not willing to accept is claims that you alone understand the way to make all computers reliable, using a hardware design that is obviously (to me, anyway) impractical, and you offer no justification beyond repeating claims that "hardware is always more reliable than software", and therefore you can practically guarantee that the future of computing will be dominated by single task per core processors.

I believe I have been open minded – I've tried to point out the problems with your ideas, and why I think it is impractical to design such chips, and why they would be impractical for general purpose computing even if they were made. I've repeatedly asked for justification for your claims, and received none of relevance. I am more than willing to discuss these ideas more if you can justify them – but until then, I'll continue to view massively multi-core chips as useful for some specialised tasks but inappropriate for general purpose (and desktop in particular) computing.

I seem to remember previous discussions reaching similar conclusions – you had a pretty way-out theory, leading to an interesting discussion but ending with me giving up in frustration, and you calling me closed-minded. These sorts of ideas are good for making people think, but scientific minds are naturally sceptical until given solid evidence and justification.

mvh.,

David

.