DISCUSSION OF PARALLELISM

William Bricken and others
various

A suggestion (CAD developer)

An idea I would like some feedback on is how to make a low-cost platform for real-time cyberspace display.

Specifically, how can multiple processors be used for transforming and displaying a polygonal database? My idea is to have a stack of three of the cheapest 286 clones, all communicating via serial, parallel, or hard-disk controller port. One clone would serve as the database queuer and database transformer, one as the scanline converter for the left eye, and one as the scanline converter for the right eye. Each converter clone would access a double-buffered VGA display. Part of the conversion process would be to keep track of frame coherence. Part of the job of the transformer clone would be to monitor any locator devices being used for the cyberspace application, and to apply that data to the transforms.

Floating point co-processors? Only for the transform clone.

Reply (programming team)

Well, I see some problems with your suggested configuration. The idea of trying to pass enough data through a serial or parallel port to dynamically update a cyberspace environment is pretty much hopeless. You might do better with a couple of Amigas, if super low cost is your objective.

A configuration that I think has more possibilities is a add-in board with an Intel i860 chip (or several in parallel) on board. The i860's computational performance and it's 3D graphics capabilities (which include near real time shading of 50,000 Gourand shaded triangles per second) have a much better chance of supporting the kind of modeling required to implement a reasonable cyberspace environment.

The thing about cyberspace is that almost there is not enough - if you can't update the two views fast enough there's a lag that makes it pretty much unusable. Turns out people actually get nauseous if there's too much lag time.

Discussion (programmers)

> You don't have to pass around the whole world - each machine knows that.
> All you'd have to pass from, say, the machine reading the headtracking
> Polhemus to the machine rendering the other eye, is the few (<< 50) bytes

> of viewpoint position and orientation info.

Doesn't that make the rather drastic assumption that the world is rather static in itself? I kind of expected the cyber-world to be a rather active place...other users, processes, etc...?

Btw, this problem doesn't quite go away even if we have only one processor, since interactions from other users comes from the outside.

Actually, I rather se tightly coupled clusters of machines rendering, possibly many (>8) per frame buffer.

I have no good solution though to the bottle neck.

But here is one wild idea:

A high speed, 32 bit wide, read-only bus, with one sender, the "world manager" and many readers, the "renderers". The data on the bus would basically be updates to the world at some low level (polygon?) to be merged into each rendering machines database.

This is not thought out well yet, let me think some more...

More discussion (programmers)

Where do you draw the line for a parallel display application? For the polygon scan-conversion application that I outlined in the notes, I drew the line at the bucket level.

However, to get the maximum throughput from your pair of processors, the best thing would be to run a software simulation of the entire process and take timings. Vary parameters to see what happens. (For example, does increasing the number of edges have a symmetric effect on both sides?) Then go ahead and write the parallel application.

Simulate Not (William)

We spent six months at Advanced Decisions Systems testing parallel logic implementations. My conclusion from that experience is that parallelism cannot be simulated.

Clarifying (programmer)

Yeah. Simulation was the wrong word. I just meant write the application linearly, study its behavior, then break the code up appropriately.

Disagreement (another programmer)

I'm kinda surprised to hear of your conclusion that parallelism can't be simulated, with a period at the end. After all, simulation's just a form of abstraction that takes time and change into account. Something's bound to be lost, as it is in any abstraction. I hope you aren't saying that because simulated parallel processes aren't tantamount to physical parallel processes that they (the simulations) aren't worth doing. No doubt you're building to a case for lots of physical processors running in parallel, and that's great; the more the merrier. But we probably won't have those for a good while, and we'll never have enough.

My feeling is that we CAN simulate parallel processes, to very good practical effect. I certainly agree that to truly understand parallel phenomena, at a deep, mathematically rigorous level, we need to work with actual parallel processes. But we can provide a convincing ILLUSION of parallelism, and for most cyberspace applications that will be enough. The practical criterion is not how valid cyberspace is, but how valid it APPEARS TO BE, or how useful it is for certain workaday purposes. It seems to me that the challenge for us is to support the illusion (that one is in another reality) despite technical and economic limitations (lack of parallel processors). My own work at ADS, on TeamWorks, convinced me that we can go very far toward supporting the illusion on sequential machines.

On the other hand, if you want to drop off, say, 40 or 50 transputers in my office, I'll be happy to find something for them to do.

Advocating Cellular Automata (CAD developer)

To make it fun we want complex and dynamically changing multidimensional environment. I think an easy way to do it is by thinking CA, cellular automata. William makes the point that the Space is the fundamental thing. So what can we put in space that updates itself easily, rapidly, reprogrammably, alterably, reproducibly, yet unexpectedly, complexly and in a way that is enjoyably deep?

Cellular Automata of course. Now that CA Lab is reaching its final FLING IT AWAY!!! I'd like to help with some input on what to put in C-space.

Cyberspace is indeed an ideal environment for running 3D CAs and 4D, or ND, in fact.

After all my thought about higher dimensions, I finally became comfortable with the belief that the MindScape or Inner Reality which we inhabit is infinite dimensional. The world is what I live in so the world is infinite dimensional. Quantum mechanics has a great power and advantage because the theory stipulates initially that the universe is an infinite dimensional

Hilbert Space.

I've been reading up on chaos theory, and it looks to me like CAs are a richer structure. The reason, in a nutshell, is that a program which draws something like the Lorenz Mask or, for that matter, the Mandelbrot Set, draws something and then stops, at least as far as my present sizescale is concerned. The CAs influences spread out into a space like hypersurface which can be perceived as a now. The Mandelbrot set's features can only be seen as a time like hypersurface of computations. Hyperline, I guess.

Though of course you could use some AI and write a software "robot" who will search out deeper and deeper levels. Particularly if there is a way to pipeline a process like that of the FractInt program.

Save all numbers in 32 bit words, consisting of 16 binary integer bits and 16 binary pointfraction bits. Save the sign in a different field of the record. The sign, if you like, can have 16 bits as well, thus sectioning off a binarily divided hypercube of options.

SIMULATING PARALLELISM (William)

The primary difference between real parallel and simulated parallel is time-ordering of computational events. In simulated parallelism, we use a sequential process to generate one possible ordering of a bunch of asynchronous events. Most simulations are an ordered instance, created by selecting events one at a time, from what is really a set of (unordered) events.

We could do a Monte Carlo simulation, generating lots of particular ordered sequences, to get a statistical approximation of all possible sequences that might show up in a truly asynchronous process. Basically, for N asynchronous events there are N! possible orderings. For most computational processes, N is too large to exhaustively simulate, and the statistical approach is the only tractable alternative.

When I think of simulation, I think of doing it for the propose of verifying that the parallel approach works. Thus simulation only proves the viability of a parallel algorithm when it is either exhaustive (not statistical) or formal. Formal proof methods for parallel computation have not yet been established.

There are distinct kinds of parallelism. Data parallelism (used by the Connection Machine), takes homogeneous vectors (eg: a list of integers) and does the same functional process to each at the same time (say, doubling each member of the list). This is easy cause the elements of the list are independent.

Logical (or control) parallelism is hard, because lists of control processes are not independent. When you take the THEN branch of an if-then-else, you are in a different computational regime. Logical parallelism permits conjuncts and disjuncts to be treated as data parallel, *but* the returned value is not independent. The first AND that is false, for instance, terminates all the other processes. Simulation of parallel logic is impossible in my experience.

Object-oriented parallelism is the way we think about parallel CAD. Each figure can be displayed independently and in parallel. Object-oriented parallelism is not different from data or control parallelism, it's just in a more friendly language for our minds. When is comes to implementation, object-oriented processes achieve data and control parallelism via message-passing and method-calls. The main point is that object-oriented thinking does not provide easy parallelism, it merely lets us confuse data and control parallelism. Only if the object hierarchy and the message passing regimes are extremely principled (knowledge-engineered), can we separate easy parallelism from hard parallelism, and know what will work and what will hang.

So sure, if you have a world of independent objects, they are easy to treat as-if parallel. When objects are mutually dependent, however, asynchrony will eat you alive.

We agree that simulation of parallelism is worthwhile. But I expect parallel processors to be in bulk supply soon now. It's parallel algorithms that will be slow coming, and they require physically parallel processors. I think what you are identifying as the "illusion of parallelism" is data parallelism, and yes our algorithms must (minimally) accommodate that. That's a great practical goal.

Reply (programmer)

On real versus simulated parallelism:

OK. I see the sense in which you mean "parallelism can't be simulated". It's not at odds with my perspective. You're thinking of general worlds in which anything can happen in any order, and you want a mechanism that can represent the possibilities. I merely want (for now) a mechanism that will enable me to orchestrate parallel events; that is, to IMPOSE an ordering for the sake of creating an illusion. Asynchrony won't eat me alive because I won't let it. I'll simply push the illusion as far as possible and then I'll give up. All I really want to do is hop into cyberspace and forget that it isn't real. From there it can only get better.

And, again, please do sign me up for that first bulk order of parallel processors. I've got nothing against the real thing.