“

INTERVIEW WITH GORDON BELL

A survey of the architectural landscape from a computer strategist who has seen it all.

BY ROB WARNOCK

Randal S. Becker

Besides setting context, interview introductions generally attempt to provide some reason for reading further. In some cases, though, hype clearly is unnecessary. This is one such instance. When the theme is computer architecture and the floor is held by Gordon Bell, why not let the credentials speak for themselves?

After joining Digital Equipment Corporation in 1960 as Manager of Computer Design, Bell handled the development of the PDP-4, PDP-5, and PDP-6 computers. He then took a leave of absence to serve on the faculty of Carnegie-Mellon University between 1966 and 1972, during which time he consulted on various DEC projects, including the design of the PDP-11. He returned to DEC in 1972 as Vice President of Engineering to direct the company's research, design, and development activities in hardware, software, and systems. He held the post for the next 11 years.

By mid-1983, ready for a new challenge, Bell moved to Encore Computer Corporation, where he was installed as Chief Technical Officer. Since then, he not only has helped to steer Encore's technical staff but has served as an apostle to the world at large about the multiprocessing faith.

To probe some of the issues that this computing approach raises, UNIX REVIEW asked Rob Warnock, himself an independent computer architect with nearly 20 years of experience, to ask Bell about the history of multiprocessing and the way in which it fits into the taxonomy of general-purpose computing.

REVIEW What are your current architectural "hot buttons"?

BELL Well, basically, two related things: getting good, fast microprocessors that can be used in a multiprocessor mode; and using them to build general-purpose multiprocessors as opposed to anything else.

The micros are coming along fine. If you graph the performance of CPUs built out of various component technologies against time, you'll see that TTL is getting better at a rate of about 14 percent per year, with ECL maybe five times above that, but improving at the same slope. MOS started slowly, but now it's improving at a much higher rate because of what it allows you to put on a chip. CMOS, in particular, has crossed the TTL curve and will cross ECL in a year or so. Going to some sort of RISC buys you a factor of two to three, maybe, but the slope stays the same. I think we'll see CMOS RISC processors that run at 10 to 20 times today's speeds, but by going with multiprocessors, you'll be able to multiply performance by 50 to 250 times.

REVIEW In trying to build multiprocessors, what are your main concerns with processor chips?

BELL Being able to maintain cache coherence. For example, it doesn't now appear possible to use Intel's 80386 conveniently in a multiprocessor. We can use it, but only if we pay a fairly heavy penalty—or, let's say, an uncertain software penalty.

REVIEW In terms of its caching?

BELL In terms of maintaining the translation lookaside buffer. A TLB has to be treated exactly like a cache. When you change state in the TLB, that ultimately gets reflected in the tables, so you have to be able to reflect the change in all the translation buffers or at least be able to invalidate the translation buffers. That can be done in several ways—some in software. But it's not enough that the TLBs just *work*. We're kind of greedy. We want them to be really right, and they currently aren't.

People have gone a little overboard about workstations, saying, "Oh, my God, look at how important these things are. They'll replace computing. Just think of all that power. Everybody will have to get their own." Nonsense!

REVIEW So, regardless of whatever subsystems are pulled in to the chip, you want to make sure that the right hooks are left outside so that you can put your own structure on top.

BELL Yeah, specifically so that you can gain access to the chip to deal with coherency.

REVIEW How similar in overall architecture is the Encore Multimax to, say, "Cm*" or "C.mmp"? [*The Cmm* and C.mmp are two research multiprocessors that were built at Carnegie-Mellon University.*]

BELL They're similar in the sense that they're all multiprocessor machines, which are the only machines I think we should build for general-purpose use.

REVIEW Do you mean it doesn't make sense to build uniprocessors anymore?

BELL No, I mean that for high performance we should concentrate on Multis instead of a lot of other squirrely machines.

REVIEW You mean like dataflow...?

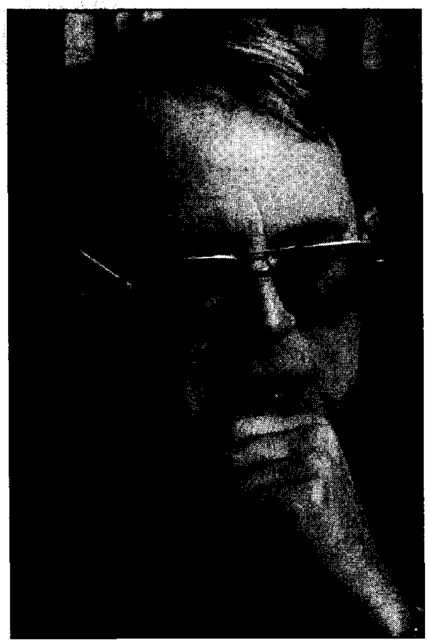
BELL The data-flow machines are strictly for research. Nothing has come out of that yet. Look, let me give you a quick taxonomy of high-performance computing. One main trunk contains all the uniprocessors. You've got things like the VAX, which is a microprogrammed machine. The RISC stuff is related to that. Then, if you get more complex with *more* microprogramming, you get the Symbolics 3600 or Prolog machines.

Then, in another branch is a whole class of scientific machines, starting with the CDC 6600. These still are uniprocessors, but branching off from them are the array processors. Put vectors on that and you get the Cray 1.

There's still another class in this trunk containing the machines I call the "locksteppers". The original machine in this class was the Illiac IV, which was pretty useless. Today, there are two fairly interesting machines, though—The Connection Machine, and IBM's GF11, both of which are big, single instruction-stream machines. The GF11 has 512 processing elements operating at 11 Gigaflops, and TCM has 64,000 processing elements [*the Illiac IV had only 64*]. Further along the branch, you've got MPP [*Massively Parallel Processor*] and a bunch of other things. These all are research machines—like data-flow. But data-flow is somewhere out in space. . . . We've also got the systolic processors to account for, but as far as I'm concerned, they're nothing more than programmable pipelines. They allow you to do certain data operations fast when you're working with a fixed program. But they're not computers *per se*, so I don't think they're particularly interesting for general-purpose use.

I'm much more concerned with how you do general-purpose multiprocessing. Now, if you look at the other main trunk, which contains machines with more than one main processor, you'll find that there are three big families—one of which already is fairly visible. This is the distributed processing family: Apollo, Xerox Star, the Sun Workstation, and the rest of them. But people have gone a little overboard and said, "Oh, my God, look at how important these things are. They'll replace computing. Just think of all that power. Everybody will have to get their own workstation." Nonsense! A workstation puts more power on the desk, but it throws you back to where we started from in timesharing—only now we've got a better file system. Distributed computing offers enormous power for one job, but only if people do their homework. Apollo made a movie using several hundred workstations operating in parallel fashion, for instance.

Besides distributed computing, there are two other paths: multiprocessors and multicomputers. Multicomputers *seem* easier to build because they have non-shared memory systems. Things like the Cal Tech Hypercube and the Flexible machine essentially



lash a bunch of computers together, but people always will take a lot of computers and stick them together in some way.

Oh, and then you've also got some real computers in this bunch—Tandem and the VAX clusters. Their approach emphasizes reliability and performance.

And, let's see, there also are components like the Transputer, which essentially is a very nice computer that allows communication with other things. Also include BBN's Butterfly in this class. But machines of this sort have many copies of the operating system, and though they may work well enough for a single fixed program, I don't think they're very interesting because they're not general. What we really need to do is work on parallel processing so we can get at the general problem.

REVIEW Is the multiprocessing branch of your taxonomy restricted to Von Neumann machines with multiple CPUs, a single shared memory, and a single copy of the operating system?

BELL Yes, basically—Von Neumann with vectors appended. The Alliant falls in there.

REVIEW That's why I asked about Cm* and C.mmp, actually.

BELL Right, those were all the same thing; they just surfaced at different times. The topology of Cm* was that of a multiprocessor because it was an entire hall full of gear that shared a single address space; it was just that we had built it out of a hierarchy of buses. In fact, the Illinois Cedar is like Cm*. The Illinois guys don't like it when I point that out.

Then there's the vector multiprocessors—the XMP, the Cray 2, the Cray 3 (which supposedly goes up to 16 processors), and the Alliant. All these are vector-plus-multiprocessors, and they make me feel a lot better about what we're doing at Encore because at one time we were doing it alone. I used to think, "Boy, we're in trouble, because there's not going to be anybody to help. We're too far out on a limb." But the fact that *all* of these other supercomputer manufacturers now are doing it is what's going to make it happen.

The machines that are using the same general approach but aren't doing the vectorizing make up a new machine class that I call "the Multis".

REVIEW Which, in truth, consists of commercial versions of C.mmp?

BELL Yeah, but these machines use a multiple microprocessor structure. We have Encore, Sequent, Synapse (before its demise; it was just too early), and a very neat fault-tolerant machine—the Stratus Multi.

REVIEW You say Synapse was a bit early. What about the other companies now in the Multi business?

BELL I think we're alright. We're at the beginning. There are other large companies trying to build Multis now. DEC can do a good one because it's got a good micro.

REVIEW The MicroVAX-II?

BELL Yeah, DEC eventually will put a lot of those CPUs on a board and get a Multi. This is the way that all computer structures seem to evolve. An outside organization has got to do it, and then the big guys will follow.

REVIEW And you guys are "outside" doing tightly-coupled multiprocessors?

BELL That's right. We're the ones trying to see that you have one copy of the operating system and all the work readily available so that you don't have an allocation problem. What bothers me about the multicomputers—the Hypercube and all that—is that they don't multiprogram. You can't load them; you can't get the results out; and you can't allocate work across them.

REVIEW Would you say that for now, doing *multiprocessors* right is so easy that we shouldn't be worrying about this other stuff?

BELL Right—for the time being. Large multiprocessors are the second most interesting thing to come along in computing since virtual memory.

REVIEW You're not including cheap processors?

BELL I'm talking about *architectural* things. First, we had the Von Neumann machine and then we learned how to microprogram it, but now people are saying we shouldn't use microprogramming. Then we learned about virtual memories, and then people put vectors on them.

REVIEW Even if Cray doesn't admit that virtual memory belongs in there, the Convex C-1 is virtual and it hasn't suffered from it.

BELL Right. And you can gain a lot—a factor of three or four—when you use virtual memory. Plus, you gain a lot of freedom in terms of not having to allocate where your programs go. I mean, virtual memory works. It worked in 1960, when it was invented. And it works today. It was a great idea.

REVIEW I've heard it said that virtual memory works even if you never page. As I recall, the DEC KI-10 ran for several years, still swapping, but using virtual memory to avoid shuffling.

BELL Yeah. Absolutely. Often, you can forget about the paging.

REVIEW There's been talk about the diminishing importance of architecture since it's increasingly defined by off-the-shelf components: chips, boards, subsystems, and the like. What's your opinion?

BELL I don't think instruction-set architecture is very important anymore because we're learning how to gain independence from that. We're not yet as independent as I'd like, though; I would like to see a much more concerted effort.

I wish we hadn't had the byte-sex problem with the 68000 and all that. That's been a big waste. And all the micros still have one major thing to assimilate—vectors. That'll take them another five years to figure out.

REVIEW Motorola tried to skirt the issue temporarily by putting its little instruction cache inside.

BELL Yeah, but you still get a big performance gain by vectorizing.

REVIEW On the next level up, is bus-level architecture still important?

BELL Sure. The structures are important because that's what enables us to build all these other things.

REVIEW Okay, let's look at the Encore machine—you're using a micro. Would you say it's a memory-centered design?

BELL Oh, absolutely! It's beautiful, almost trivial. We've used our "Nanobus", which is 100 MB per second. That's 64 bits of data and 32 bits of address...

I don't think the instruction set architecture is very important anymore because we're learning how to gain independence from that. We're not yet as independent as I'd like, though; I would like to see a more concerted effort.

REVIEW Excuse me for interrupting, but one of the quotes attributed to you is: "The only error that's almost impossible to recover from is having too little address space." With only 32 bits, are you beginning to worry about that?

BELL Yes, but we can't afford to solve the problem right now. You need another bit of address every two years, so, yes, we're going to have trouble. But we'll be okay for another seven years.

REVIEW Back to the Encore machine...

BELL Okay, plugged into the Nanobus is a processor module with two processors on it. The Multimax can accommodate up to 10 of these modules. We have a shared cache for the processors on each module, but decisions about whether to share it or not, or how many processors should share it, are just details.

REVIEW Do those decisions affect the extent to which the machine is suitable for very fast context switching?

BELL No, because with multiprocessors, you shouldn't have to do so much context switching because a program should run longer. As a result, multiprocessors actually can give you speedups greater than one [that is, greater than linear with the number of processors], because there is less overhead.

REVIEW One of the advantages of RISC—or perhaps I mean simple machines, not the multiregister-bank RISC—is that the small amount of machine state lets you context switch faster. The 68000, for example, has too many registers. Are you saying that your approach allows you to keep more local state without impacting performance?

BELL In principle, but we haven't worried much about that problem because if you put the machine together right, you end up with so damn much processing power that it ceases to be an important issue. For the first time, processors are so cheap that processing power no longer is the key issue. All we're trying to do is make a machine that exploits that.

REVIEW So if you waste 50 percent of each processor, who cares?

BELL Right! Who cares? And we *should* think of it that way, because if you have 4 MB of memory (up to 16 on our next card), and two processors per module (we're trying for eight on the next one), that gives you up to 80 processors. And look, two processors equal 4 MB resource-wise.

REVIEW Is this Amdahl's rule taken to very large granularity?

BELL Yeah, as it turns out. If you think of a processor strictly as being equal to a couple of megabytes, you're probably not going to worry about how much of your memory is utilized, are you? Nobody worries about memory. You're going to use it, but you are not going to worry about occupancy.

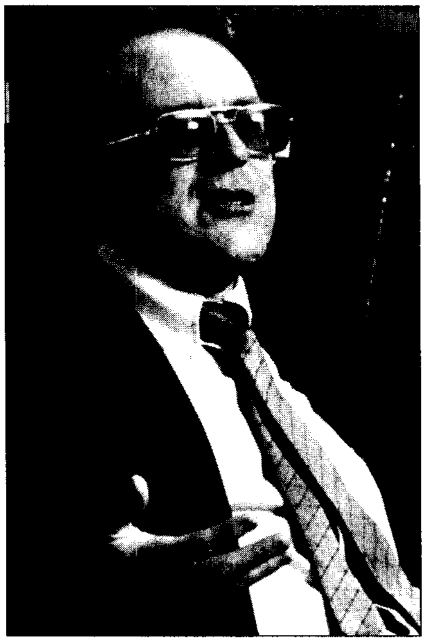
And then we have an I/O card with two channels, one that has a SCSI disk controller and another one that ties into Ethernet. We can have one to 10 of those, and the sum of processor modules plus I/O is 11. Finally, there's a system control card that handles diagnostics, initialization, and time.

REVIEW So basically you have no peripherals other than an Ethernet and SCSI?

BELL Right. I've been trying to get rid of communication I/O on computers for thousands of years. On the Ethernet, we have a concentrator called "Annex" that has a processor and 512 KB, so you can do your front-end processing or screen processing there to avoid Ethernet traffic and avoid even more context switching.

REVIEW With a 32-bit CPU and a bus of the same width, are you pipelining the fetch of several cache lines at a time? And is it interleaved?

BELL It's all pipelined. But we're not doing pre-fetch. The bus is actually 64 bits wide, and then its memory is four-way module-interleaved, so you get a total of eight-way interleaving.



REVIEW So you either have one, two, four, or eight memory modules.

BELL Right. That sort of sums up the Multimax.

REVIEW Most of the multiprocessor companies currently are using National chips. Why the National 32032? Why not the Motorola 68020? And why didn't you give in to the pressure of Intel?

BELL At the instant in time that we made the decision, we based it strictly on the fact that the 32032 offered the only truly complete chip set—that is, one with a CPU, MMU, floating point processor, and timing unit. When we made the decision, the 68020 wasn't out yet, and I was concerned that it actually wouldn't be on schedule. What's more, the 68020 didn't have its own MMU, and it still doesn't. I'm not even sure if the floating point for it is out yet or not. The 80286, meanwhile, clearly is not suitable for making a real computer, since a real computer has to handle paging. The 80386, though, finally is a real computer.

REVIEW Are you committed to the National chip?

BELL We're willing to use whatever works well. People think we're not that committed to our current byte sex, to the Little-Endians, but we probably are much more committed than we care to admit to the VAX data types, the IEEE floating point, and the VAX byte order. That pretty much limits us to either the National or Intel lines for the time being, and it would make a move to Motorola quite difficult. But MIPS is making its machine so that it can run either byte sex, and Fairchild's Clipper also can run either byte sex, so there are a number of processors coming out now that use the same byte-ordering that we do.

REVIEW Still, the network protocols are all Big-Endian.

BELL Yeah, but it almost doesn't matter. The time you spend handling network protocols is a lot greater than what you might lose by converting byte order.

REVIEW How about the impact of the RISC approach versus the impact of overall architecture?

BELL Oh, I don't think RISCs are that big a deal. I've been convinced for about four or five years that people should be developing non-microprogrammed instruction sets. I hardly would call them "reduced" instruction sets, though. I don't like the word "RISC" because it doesn't reflect the true nature of the concept. Let's say "load/store, non-microprogrammed computers with all the proper data types". The word "RISC" to me actually connotes a lot of bad stuff. The RISC-I and II were just a couple of unfinished computers. They didn't have memory management units, and they didn't do floating point. If you added these, you'd get a pretty fat computer.

Again, these machines were developed by universities, so the people working on them weren't expected to build real computers. As a result, I think people have a tendency to disregard that work because it was incomplete and because the benchmarks they used basically were toy benchmarks. But mainly, both machines simply failed the primary test: you never would use one of them to design the chip itself. For that, they always used VAXes because VAXes have floating point. On the other hand, if you ignore all the RISC proselytizing and focus on what is really important in that work, you'll see that it *is* the right way to build a computer now.

REVIEW What other existing machines are non-microprogrammed, load/store, have only memory protection, and contain the right data types?

BELL The Cray stuff is exactly that way. Cray has never deviated from that architecture. So the only thing that's really happened is that people have rediscovered Cray.

The other thing to remember is that technology drives all of this anyway and that we're just on our second turn around the wheel. The same people that fanatically talk about RISC today are like the ones who were fanatics about user microprogramming 10 to 12 years ago. We went through a time at DEC where folks thought it was absolutely immoral not to provide a microprogrammed machine—that we needed to make the microprogram store writable so that people could write their own microprograms.

In fact, the real reason we put microprogramming in at all was to get the size of the floating point interpreter down. If you throw out floating point, the RISC/non-RISC argument becomes specious.

REVIEW Given the existence of UNIX, are operating systems still as important as, say, bus-level architecture?

BELL I think they're of equal importance and difficulty. We still have to go through the change of making the operating system deal with the multiprocessor problem. And all the current buses—VME, Multibus-II—support multiprocessors *in principle* but not in practice. They've totally botched it, which gets back to my first "hot button". The micro guys have got to understand Multis so that they don't continue to botch things.

REVIEW Since we already have VME, Multibus-I, Multibus-II, and Q-bus, is there room for a new open bus that's better suited to multiprocessors?

BELL I think VME and Multibus-II can—and must—evolve. But they're not moving in that direction right now. I'm concerned about how to *get* them evolved so that people start thinking in terms of multiprocessors.

REVIEW For a while I thought the MIT "Nu" bus might have a chance.

BELL That bus could have done it. But I don't see the latest IEEE bus, "Future Bus", even getting into the game because no one is championing it. See, the people who could produce the right bus—who really understand Multis—are companies like Encore and Sequent that have a proprietary interest. I'd like to see our Nanobus become a standard, because I basically believe in standards.

The Nanobus really is worth it. It's a 100 MB bus—not one of these 20-25 MB things that you can't build a Multi with. When you *try* to make a decent computer with one of these other buses, you end up having to stick another bus on the side. . .look at Multibus.

REVIEW And that puts you back to building a distributed computer.

BELL Yeah, that's what you generally end up doing to get the bandwidth because the microprocessor guys haven't quite discovered how to handle caches yet. There are really two benefits to caching: one is to get access time down, and the other is to filter requests so that a bus can handle all the data.

I don't think RISCs are that big a deal. I have believed for about four or five years that people should develop non-microprogrammed instruction sets. I hardly would call them "reduced" instruction sets, though.

This is the first time caches really work for you. In the past, caches worked against you. Suppose you had a couple of processors sharing a multi-port memory; if you put caches on, then suddenly you were a long way from being able to update that memory. I mean, you ended up needing some kind of a cache-to-cache connection.

REVIEW So when I asked you earlier if the Multimax has a memory-centered design, perhaps I was wrong. Isn't it really a cache-centered design?

BELL No, I think it's memory and bus-centered.

REVIEW But if you have more than one of these caches sitting out on the bus, don't you have exactly the same problem?

BELL No! No! No! Every transaction watches the bus. So, in the simplest case, if I do a write and invalidate a word in a cache, it goes out and writes in the memory. Since all the other caches in the system are looking at the bus, they all immediately invalidate themselves as well. That's why I love this structure. Everything works *for* you, whereas in a lot of other structures, things start working against you and life becomes a tradeoff.

REVIEW Getting back to UNIX, what do you see as its role? Do you see it as an advantage or a stumbling block?

BELL I see UNIX as a major component that lets this all happen by getting rid of the operating system argument. That is, it eliminates one major risk users generally take when accepting a new architecture. As far as I'm concerned, UNIX is helping the development of multiprocessors. Cray has adopted it. Everybody else is adopting it, so we can be pretty sure we'll have some user software to pull onto our machine. If Encore had to develop its own operating system from scratch, along with all the utilities and all that other stuff—we wouldn't even be able to get started. It's the same with all these other companies.

REVIEW So, even with all of its warts, UNIX still is a major impetus to hardware innovation?

BELL Absolutely—because it takes care of the operating system question and lets you innovate like heck *under* that level in a relatively transparent way.

REVIEW Some say that UNIX is *the* system call interface, and that anything below that is invisible.

BELL That's right. All I want to do is extend UNIX, make sure that the distributed processing parts are done well, and get some sort of agreement on the network file system. I think Sun is doing fine there. AT&T should start leading, though. It should be better at taking the advanced work and assimilating it more rapidly. AT&T didn't take a leadership role in shaping the standard, you know; that all came from the /usr/group work.

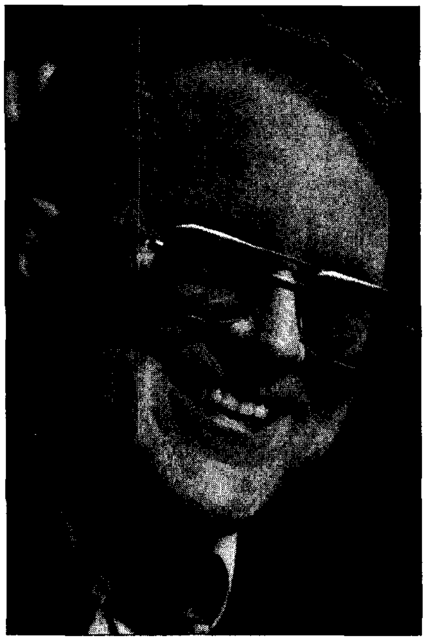
REVIEW Is there anything in particular about multiprocessing that you see as a real stumbling block for UNIX—such as rethinking the locks on the kernel data structures?

BELL You have to rewrite the system so that the system calls are multithreaded, but you can do that.

REVIEW So, for example, all I/O must be enqueued so that whatever processor is free to take the interrupt can do so?

BELL Right. In fact, my druthers would be to get rid of interrupts altogether, with the exception of the end-of-time-slice clock interrupts. With all this processing power, you're much better off without interrupts. We've got plenty of I/O computers, so why use the interrupts?

REVIEW Have you done anything yet in terms of strategic scheduling, such as putting different processes in a pipeline on different processors?



BELL Oh, yeah. Different tasks end up on different processors. You actually get transparent multiprocessing. You get speedup, and you get increased throughput.

REVIEW And this is what gives you your greater-than-one scale factor?

BELL Yes, even for a single job. What we really want to do is get to parallel processing, where we can run a single job over *n* processors with full multitasking. To achieve that, we've just made a minor extension to UNIX in the form of a **share** command, which allows all the children of a process to inherit a region of shared memory.

REVIEW What about synchronization?

BELL We provide libraries with a hierarchy of locks, mostly spin locks, to avoid rescheduling.

REVIEW Have you done anything yet with any of the languages such as, say, LISP—like making sure that the garbage collector runs in parallel?

BELL Not yet. We're just getting to the point where we can look at things like that. We've proposed a research program that would work on parallel LISP. It also would allow us to run up to 500 to 1000 processors.

REVIEW Five hundred to 1000 processors? Are you still going to be able to have a single virtual address space?

BELL The structure uses a single-address space. It's in a hierarchy, so it's done by caching. Basically, it's a scheme that uses a switch and a set of filters that can interconnect between eight and 16 machines—somewhat like Cm*.

REVIEW You said that the truly interesting case is the speedup of a single job, by internally multitasking across several processors. That raises the question of automatic decomposition of jobs. Is that possible?

BELL Oh, sure, if the semantics of the language are right. For example, the OPS-5 production system that some AI applications have been written in already has the proper semantics for parallel processing. You also could argue that APL does that. Even Fortran—post-1977 Fortran with vectors—does that. It's only too bad that we didn't start work on these new semantics a whole lot earlier.

REVIEW You've already given us part of the answer, but. . .what does the future hold?

BELL Oh, man! In the multiprocessor arena, I would hope it brings about a change in the way computers are built. My fervent hope is that we're finally going to have parallel processing and get off our slow, technology-only evolution.

REVIEW How much of that depends on changing the way people *write* programs?

BELL First off, it isn't very hard to write programs for multiprocessors. Some people even enjoy it. So far the people we are talking to are just ecstatic about having a new challenge. Maybe it even will help people stretch their minds out of a 40-year sequential rut. ●