10/00

Carver Mead Interview Part III
(With Dick Lyon, Schlumberger AI researcher)
By: Gene Youngblood
Document 3 of 4 of Mead Interview

CARVER: When I say that one of our music chips is equal to 600 VAXes, well, "equal" is a funny word. What I mean is that if you do the same calculations on a VAX it takes about ten minutes to do one second of sound. So one of those chips does as much computation of that particular sort as 600 VAXes could do. Now VAXes are very poorly set up for doing computations of that sort, but so is every other general purpose computer. Now there are special-purpose engines that can do comparable things; so mine isn't the only way to do it.

GENE: What did you mean by "by the time you get done shuffling all the data around, you wind up with something that's a few hundred times real time instead of a few tens times real time?" What do you mean by real time?

CARVER: You ought to be able to generate the sound as fast as you need the samples. To compute the samples as fast as they're needed for you to hear the sound. You can't do that on a regular computer. If you look at how many multiplies and adds it takes per second to do these kinds of computations, it's only five or ten million per second. That sounds like five or ten times as many as a VAX can do. But it turns out that's not all the VAX is doing. It ends up shuffling all the data around to be in the right place -- you fetch it and you get it in the machine and you multiplying it and then you get it back out again. And all those how many multiplies you can do per second don't count if you're only getting the data to multiply, you see. We were very surprised. People had told me this for years, and I always thought well yeah, it's true for general programs, but I never thought that for a program like this it would be true that data shuffling took most of the time. But it turns out that there's enough pieces of data that you have to grind on and put into other places that that ends up taking more time than all the multiplies. Every time you want a piece of data you've got to compute it's address, then do a partial calculation, then figure out where to put the answer, and then put it back, and so on. All those things eat you alive. So only about one-tenth of the instructions actually end up multiplying anything. So you could do the actual multiplies in zero time and it doesn't speed things up very much. Whereas with special-purpose architecture you arrange it so the data automatically flows to the right place at the right time. In fact in our chip and others that was the key to the whole design -- to make sure that that's what happened.

GENE: Is there a rule of thumb for the amount of speedup you get from a dedicated architecture. All else being equal, if you're going to put the same algorithm in silicon as opposed to running it on a VAX. What kind of speedup in principle, just by doing that?

CARVER: For a lot of the applications we've looked at it's more than two orders of magnitude. One way to look at it is that you get one factor of ten from what was just being discussed -- from the fact that it's specialized -- and then as many more factors of ten as you like from the fact that you're putting more of the specialized hard- ware in that -- you get a factor of ten or 100 because you used 10 or 100 multipliers and another factor of ten because it's not general purpose. So that's a hundred or a thousand right there. So that's why we're at a factor of the order of 1000 -- because you've got things working in parallel and then you're not doing any data-shuffling. So that's another factor of ten.

DICK: My speech recognition machine's getting the same factor with respect to our general-purpose computer; I get somewhere between a factor of 1000 and 5000 speedup. I'm doing speech recognition by implementing a model of how the ear processes sounds. Sounds of all sorts including music.

CARVER: So my machine does this particular class of computations about 600 times faster than a VAX. But it won't run a PASCAL program and it doesn't run UNIX and it doesn't maintain a file system. The way I think of general purpose computers is they're the little environment that does all those stupid things -- hooks up to Ethernets, hooks up to telephone lines, hooks up to keyboards, hooks up to humans, compiles languages, accesses file systems -- does all those things AND issues commands to the special-purpose engines that do the really hard

grunt-work a thousand or more times faster than it could.

CARVER: Six hundred times a VAX would be about ten times as fast as a Cray. I haven't programmed this on a Cray so I wouldn't say that; in fact a Cray is set up to do this kind of calculation reasonably well also, so that for this particular task the Cray might be more than 60 times a VAX. I want to be very careful not to mislead people about this. One way of thinking about it is that you're not just taking the VAX and making a special purpose thing; you're taking the whole program that does a special thing on the VAX and you're saying that I'm going to take that very special application and instead of a general purpose programming language on a general purpose computer, I'm going to put the whole thing -- application and all -- right down into silicon.

GENE: Who invented the silicon compiler?

CARVER: It evolved gradually over a period of about eight years. But the first one that was recognizable as a compiler instead of something less general was done by Dave Johannsen in 1977. I did a very special purpose one in 1971 for finite-state machines, but it wasn't a silicon compiler in the sense that we talk about them today.

GENE: The appearance of silicon compilers is a significant historical event; does what makes them possible come out of your structured approach to VLSI design?

CARVER: It had been possible for a long time. It isn't a matter of making it possible; it's that people had been thinking of hand-crafting everything because in the early days you never had enough silicon to waste any at all. And any structured approach to design is in some sense slightly inefficient at the bottom level. You can always find little places that aren't all filled up with transistors. Exactly the same history happened in code. Back when machines only had a 4K memory, people wrote every machine instruction because you couldn't afford any waste at all. But when you have a megabyte of memory you can let a compiler allocate variables because if you waste ten percent it doesn't matter any more. The same thing is true of silicon compilation: people did not foresee the fact that we would have so much real estate available; that the real problem was the design cost, not the cost of the real estate. When I tried to tell people that in the early days they laughed at me. That was within five years of the time they were standing up and saying it's the most important problem for the semiconductor industry. They were laughing when you would say that the design cost was going to be larger than the production cost for complicated parts. Now that it's true it's them that invented it. But at the time I remember trying to get the Intel people interested -- this was in 1973 -- and Gordon Moore said it was an academic problem, and by the late 1970s he was publishing his curve of the design costs going up to the moon. The compiler viewpoint comes from saying we're going to have enough real estate and we're not going to have enough engineering man-hours, so what you ought to be working on is the design problem, not how dense is the silicon.

DICK: But it's still fair to say that this approach of doing things in a more structured way is what made silicon compilers possible, by setting up that way of thinking. To go from hand-crafting to structural building blocks sets the stage for the next thing, which is compilation. So it's fair to say that the Mead-Conway approach made silicon compilers possible.

CARVER: And Dave Johannsen was the one who first really saw how to pull the whole thing together in a much more general way than people had done before.

GENE: So it's a kind of synergy between VLSI, which makes possible a more simplified structured approach, plus that making possible the silicon compiler -- all of that together results in the ability to put algorithms in silicon with a fast turnaround time. You can't separate the compiler from the structured approach and you can't separate the structured approach from VLSI.

CARVER: There are a lot of things that can't be separated. Having people in business who are willing to fab chips that you design. In fact we had to start a whole new set of companies to do that because the traditional semiconductor people didn't want to do it. They still don't want to do it. They'll do it for their large customers who want to make a million parts. So this thing couldn't happen until we had access to fabrication. And there was a major amount of energy went into getting people to understand that there was a good business in that; and right now that's the only part of the semiconductor business that's making money. It's happening in a big way now. DARPA provides fab services at a very low cost to research projects in a few hundred universities. But there are

also a number of commercial places that will do that kind of thing now for low-volume engineering prototypes.

GENE: How do you talk about what is actually on a dedicated chip? What makes the adders and multipliers in your chip dedicated?

CARVER: Well, on one of our chips we're apt to have fifty multipliers all working at the same time; on a microprocessor you have to have one -- or zero, more likely, because they use adders instead and make up the multiply out of additions. Almost all microprocessors including the MC68000 have zero multipliers. Both in Dick's speech chip and the chip we use for music, the key is there's a way to set it up so that where the data comes from and where it goes to is something that can be set up and just run and not take a bunch of instructions to figure out everything.

GENE: Because the instructions are the chip?

CARVER: The chip has a specialized way of setting up where the data should come from and where it should go, so that the facilities for moving data efficiently from every place it might want to come from to every place it might want to go to are a big part of the chip. And it's not hardwired to a particular algorithm; it's got enough flexibility to be able to run a reasonable number of algorithms within a class. Like different music instrument models can be put on our chip. Different kinds of digital filtering structures can be put on Dick's speech chip.

GENE: So it's not accurate to say that this chip is a cello?

CARVER: It can be configured to be any kind of instrument, but it's dedicated in the sense that it performs that limited class of calculations which are relevant to those physics formulas. The big challenge in this specialized hardware game is to make a piece of hardware that's not so specialized that it's only good for one thing. You want to leave a certain amount of programmability of some sort. This results in a good chunk of your silicon area -- maybe as much as half -- being dedicated to the job of getting data from where it's coming from to where it's going. In a general purpose computer that takes ninety percent of your resources; in our chips it's probably less than 20 percent, in Dick's speech chip it's fifty percent. You get different amounts of generality for the different amounts of overhead.

GENE: How many transistors is on your chip?

CARVER: It has 64 multipliers, each giving a 32 by 32 multiply with a 64-bit product. There are 32 stages of the multiply. It has about 200,000 transistors. That's a lot less than the Hewlett-Packard chip that has 450,000. But for this particular application our chip is about 1000 times faster. The time-honored way of getting a high transistor count is to put a lot of memory on the chip. Our chip has a lot of memory which is mixed in with the processors, and the very thing that makes it efficient is that you mix the processing in with the memory so you don't have to move things from the memory to a processing element and back. You mix them together so processing is going on all the time. Let me say while we're at it that the 600 VAXes estimate is extremely conservative because the clock for our chip is slowed down so that it matches the sample rate which is the standard that all D to A converters use. There'd be nothing to prevent us from running the chip five times faster if you wanted to do graphics instead of sound. But in a sense that's not fair because we're computing these samples; and the fact that you could compute them faster than real time doesn't help. On the other hand, because a VAX computes them slower than real time we ought to be able to at least count that somehow. But essentially once you're at real time at a given sampling resolution you don't need to go any faster. Your ear couldn't hear it anyway. Now if we were working on bats instead of humans, well bats can hear at 200 KHz instead of 20 -- they're 10 times faster than people -- so then we'd have to run the clock 10 times as fast. We could do that, too, in which case it'd be the equivalent of 6000 VAXes. Now the other thing is that you may have the need to generate 10 musical instruments at once, and with our chip if you run it 10 times faster it still can't quite do that -- it can generate one very fast instrument but not ten at once. Dick's speech chip has an extra degree of flexibility that lets you be ten things at once instead of one thing very fast. In fact his chip is not very good at doing one thing very fast, so he couldn't make very good music for bats but he could maybe analyze sounds from several microphones at once. That costs him a lot; he has half his chip dedicated to getting data from where it's coming from to where it's going and by the way storing it temporarily in between. That's the extra cost of that extra generality. And that's the tradeoff you make depending on what you want to do. The discussion among people who do these specialized architectures is another level up from all this,

namely, now that you've got things that run 1000 times as fast as a general purpose computer would on the same problem, can you make it 10,000 times or 100,000 times faster -- can you get an even better match to the problem which uses the silicon in even a better way? It's just amazing when people start playing this game. It's not just more concurrency, it's algorithms that are more thought-through, tighter. In fact both Dick and I have traded off for generality the ultimate in performance. There would have been ways for each of us if we'd wanted to be really specialized to make the things another order of magnitude faster. We chose to have some generality instead. For example we both use a lot more bits of accuracy than are needed for most of the problems we're attacking, because occasionally you suddenly need more bits, so you spend the bits always and pay a big cost, but this means you don't often run into a problem where you don't have enough bits. That's one of the dimensions of variability, how many bits you choose to put in. We use 64-bit accumulators. Most people who do music synthesis would say that's outrageous, we're throwing away resources.

GENE: Today to get an updated version of a software package I buy a floppy disk. How would I update my computer if everything's in hardware? Buy a new board?

CARVER: There's a little company in the bay area called Silicon Solutions, Inc. You should talk to them because they make an algorithm in silicon. They founded this company on the premise that algorithms in silicon are going to be the future. Their first product checks design rules on integrated circuits. So it's an integrated circuit that checks for errors in the next integrated circuit. The next thing they're building is a chip simulator. What they did is make an add-on for a standard workstation. So you can plug the card into your workstation's multibus slot. And there's a little software package that goes with your standard UNIX operating system -- workstations are all UNIX based -- and now you can be up and running with their design-rule-checker in your workstation. So instead of a disk that you plug into your disk slot there's a board you plug into the board slot. There's a chip that implemented the Ethernet protocol. It got built on a little board that plugs into the PC. So that's an example of an already existing approach. When you buy software on a disk you pay five dollars for the floppy and three-hundred dollars for the ideas in it. A board costs about the same, maybe ten or twenty dollars for a cheap board instead of five dollars for a floppy disk. Or even if it was $100 for the board you're getting instead of something new but just as slow as ever, it does something new but 1000 times faster. So that factor of 1000 costs you a hundred bucks over what it would be if it was a software package. And you get a factor of 1000. That's a pretty good price. That's a factor of ten per dollar.

End of Carver 3