

Are Single-Chip Multiprocessors in Reach?

One of the advances that will be enabled by system-on-chip (SOC) technology is the single-chip multiprocessor. As VLSI technology improves to allow us to fabricate hundreds of millions of transistors on a single chip, it will become possible to put a complete multiprocessor, including both CPUs and memory, on a single chip. Single-chip multiprocessors will be useful not just in low-cost servers but also to perform video and a wide variety of consumer applications. The advent of single-chip multiprocessors will require us to rethink multiprocessor architectures to fit the advantages and constraints of VLSI implementation. This roundtable explores the design and test issues posed by single-chip multiprocessors.

IEEE Design & Test thanks roundtable participants Reinaldo Bergamaschi (IBM), Ivo Bolsens (IMEC), Rajesh Gupta (Univ. of California, Irvine), Randolph Harr (Intransa), Ahmed Jerraya (TIMA), Kurt Keutzer (Univ. of California, Berkeley), Kunle Olukotun (Stanford Univ.), and Kees Vissers (TriMedia Technologies).

D&T gratefully acknowledges the help of Wayne Wolf (Princeton Univ.), our moderator, and Kaushik Roy (Purdue Univ.), our Roundtables Editor, who organized the event.

Special thanks go to the IEEE Computer Society Design Automation Technical Council for sponsoring the roundtable.

D&T: What do we mean by single-chip multiprocessors?

Harr: Let's assume that there's at least one or more instruction-set processors on the chip. What needs to be defined next is whether there are multiple instruction-set processors on a chip, or one instruction-set processor and many custom data path processors with possibly variable control.

Gupta: With instruction-set processors, it's not just a question of what that underlying component does, but also at what level the software is integrated? Are we integrating at the compiler level or at the lowest level? In that sense, "system on chip" is pretty broad. "Systems on chip with reprogrammable DSP processors" is also very broad—"system-on-chip multiprocessors" is a much narrower term to use.

Olukotun: The classic multiprocessor chip has several instruction-set processors. We have everything from symmetric multiprocessors that share a central memory, to ones with a network between the processors, which are communi-

cating via message passing. Putting all these architectures on a single chip is what people classically mean when they talk about multiprocessor chips. The processors may not necessarily all be the same; they could be specialized.

Jerraya: The multiprocessor concept, in which "multiprocessor" means having several master processors on a chip, is central to our discussion. These processors can be programmable instruction sets or they can be specific hardware. But the key concept is that you have several masters, each with its own bus. The main issue is how to connect these through a processor network.

Harr: We can now put more than just the instruction-set processor on a chip. With floating-point operations, we still have 90% of the chip left over. With all this extra real estate available, we could look at multiple instruction-set processors or heterogeneous instruction-set processors with custom data paths intermixed—all sharing buses, memory hierarchies, and so forth.

Vissers: The application domain is shifting from desktop computers and multiprocessors

to general-purpose computers and embedded systems, especially with the new applications manipulating all kinds of sensory signals—images, audio, and so on. Look at cell phones, they already have an ARM controller and a DSP: dedicated ASICs and DMA.

Jerraya: Let's keep on the application theme for a bit, because that will help us understand the architecture space. What are single-chip multiprocessors good, or not good, for?

Bolsens: Multiprocessors will be important in multimedia data processing, because this area is first of all characterized by enormous computing power and, even more important, by a massive amount of data communication and data storage. In the future we won't be able to solve that with a monolithic memory and processor architecture. We'll need distributed architectures to deal with the performance requirements these systems have.

Olukotun: Applications are pushing for multiprocessors; we see that in the explosion of the Internet and the Web. Space is at a premium in Web servers and at the back end for transaction processing. Chip multiprocessors are a natural because a) we've got independent threads of control, which we need to take advantage of the architecture, and b) we need to fit a lot of computation into a small area.

Networking is another area that can benefit from multiprocessors. Depending on the level, we might want a different network process or architecture. For instance, if we're routing at layer 2 or layer 3, and we want to do this at wire speed, then we may want specialized processors that do very fast packet lookup and forwarding. As we move up to layer 4 or layer 5, with applications such as Web switches that balance the request load to multiple servers and we want routing based on a URL or high-level content, we'll want a general-purpose type of multiprocessor architecture.

Gupta: New architectures are often tiling architectures: graphics engines, packet analysis, gene analysis. These are embarrassingly parallel computations. Much of computation is mov-

Gupta:
"The problem of system-on-chip multiprocessor design is the ... complexity. We... need architectural simplification."



ing on chip, so we're talking about running literally hundreds of threads simultaneously. With these applications, we want to simplify the CPU. The problem of system-on-chip multiprocessor design is the design's complexity. We definitely need architectural simplification.

Harr: Those embarrassing parallel applications are generally also systolic, though. Systolic processors represent a custom computing machine that's not very general purpose—not what most of us are after in multiprocessor design, I believe. Instead, we're looking for platform-based, multiprocessor design that can work across multiple product generations or within a broad product family.

Bergamaschi: We've got two lines of thought: One is traditional multiprocessing, with multiple programmable processors, ranging from very nimble ones—even approaching the systolic idea—to more generic ones. To put those on a chip, to some extent, requires the approaches we would take if the multiprocessors were actually on different chips. On the other hand, there's the more embedded kind of application where multiprocessing is going on, though not necessarily in multiple, programmable, separate processors. Many SOC devices are programmable, and they do things concurrently inside an SOC. The problems in this domain are still far from being solved, and haven't been researched nearly as much as the traditional multiprocessing problem.

Bolsens: Another point is the evolution to low power. Low power is the driving force for multiprocessor architectures. Silicon real estate will let us have multiprocessor architectures running at lower clock speed and with sufficient com-

**Vissers:**

"Reuse is going to be important, and tiling and regularity are ways to achieve reuse."

puting power. This is impossible with the monolithic processors requiring 10-gigahertz clocks, which basically will heat up the chip. In all applications where power is an essential issue, we will see a trend to multiprocessor architectures.

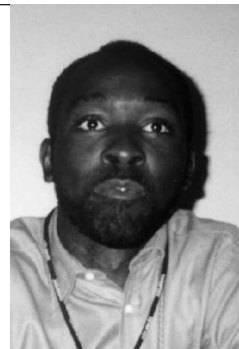
Olukotun: A further advantage of a multiprocessor architecture is that if we're not using all the processors; we can turn some off, which saves power.

Vissers: There's a correlation between the multiprocessor architecture domain and the application domain. For example, we might have multiple, independent requests—whether it's transaction processing or Web serving—and we can exploit that characteristic in our architecture to make it work well in that particular domain. For Web servers, the question is, how many Web hits can we serve per cubic foot? For cellular phones with potentially directional sensitivity and multiple antenna arrays, can we run systolic-style algorithms at the lowest possible power?

Gupta: Systolic processors have been researched and many have been built. It's one thing to have a regular architecture; it's another to have an SIMD-suitable application. Applications are also becoming very control-intensive. The new graphics applications are increasingly content-dependent, as are JPEG 2000 and MPEG. Bandwidth requirements are increasing at a rate that requires us to think about distributing the CPU/memory resources. When we talk about systolic-chip multiprocessor architecture, we must be sure that regularity doesn't just mean application regularity. We'll still have complicated applications. If we try to design on-chip, multiprocessor systems

Olukotun:

"...we're considering what the architecture should be in 2005. First, it's got to be a multiprocessor on a chip."



with very customized architectures, with little or no regularity, it's going to be hard.

Vissers: An extremely important issue that has unfortunately not been getting enough attention is whether we can build these big systems without designing every gate personally. Reuse is going to be important, and tiling and regularity are ways to achieve reuse. The multiprocessor architectures will give away some inefficiency at the lower levels in gaining the potential for reusing relatively large tiles, cores, or IP blocks.

Olukotun: At Stanford we're considering what the architecture should be in 2005. First, it's got to be a multiprocessor on a chip. Once we come up with an architecture, then we must decide how to program it and how to map applications onto this processor array. It comes down to what the memory system looks like, because that's what distinguishes architectures. What does the memory hierarchy look like; what implications does the memory have for programming? Is it shared memory where everything is equidistant? Is it a cache where some things are closer than others? Is it no sharing at all, like local memory: To talk to the other memory, do we need to make a function call? The programming model determines how the programmer breaks up a single application and distributes it among the processors.

Harr: We cannot simply shrink the board and card cage designs of traditional large-scale processors down to a chip because the trade-offs are dramatically different at these two levels of the packaging space for transistors. We must look at the architecture that makes sense to the single-chip circuit capability of a multiprocessor.

Jerraya: There will probably be several kinds of multiprocessor systems on chip, ranging from the general-purpose system processor on chip to the application-specific multiprocessor systems on chip with specific processors and specific interconnects.

Vissers: With this single processor, more or less shared-memory view, we've been classically thinking that programmers can basically focus on computation variables, and we can completely hide the interconnect and memory chip subsystem. In the new generation of multiprocessor systems, the interconnect and memory subsystem are going to be so expensive that we'll need research in parallel programming languages or computation models to establish the communication cost.

Gupta: From a general-purpose viewpoint, the programming model might well be one in which we can access anything anywhere. But to assure performance, we must worry about locality and optimize the application's locality to make it match the cache hierarchy. But the picture changes once we move to, and communicate, on chip; that is, the amount of bandwidth between processes increases by an order of magnitude or more. The latencies improve, too, and that's going to change how we can parallelize applications. If the application is naturally parallel, then fine. But even applications that typically didn't work on chips, on multiprocessors that were spread among multiple chips, now work once we bring them onto the same chip.

Bergamaschi: That's definitely true, but if we start relying on on-chip communication being done over 256-bit buses routed all over the chip, we're going to have a problem actually building that chip, due to timing and other low-level issues.

D&T: What can we say about an off-chip interconnect? For example, if we have a multimedia video application that requires a lot of bits, will we be able to keep the on-chip units busy?

Bolsens: One problem in multimedia applications is that we cannot keep our processors busy.

D&T [Wolf]:

"...if we have a multimedia video application that requires a lot of bits, will we be able to keep the on-chip units busy?"



Getting the data out of the memories, because of the latency problem, degrades the system performance—not the computing performance but the overall system performance. We must realize that this situation is only getting worse. For instance, let's say an average profile on an MPEG-4 application is characterized by, say, 5-Gbyte/sec memory access (only counting relevant data bytes). That's a typical performance metric that multimedia applications are going to need, and we don't get that with current architectures.

Gupta: We're looking at a scenario where applications will have to provide more parallelism than in the past. There's also quite a bit of progress now in multithreading at the hardware level and with low overhead. Combining these two gives us architectures that are regular, highly tiled, and with good compiler-level integration for migrating applications to them. Some applications will have parallelism inherent in them; some won't. But the architecture will still outperform sequential processors because of the bandwidth capacities.

Jerraya: A key difference between classic multiprocessor architecture and this future multiprocessor system on chip is in the way we describe the applications. The main difficulty with classic multiprocessor architecture was mapping applications—people would start from a single sequential program and try to extract parallelism to map these on multiprocessor architectures. But we're lucky because we're describing our system specification using parallel models. In fact, hardware description languages have always been multithreaded and multimodule, supporting concurrency. Mapping concurrent programs on multiprocessors is



Bergamaschi:
“A generic programming model won’t be an efficient solution; we need a more bottom-up model...”

Bolsens:
“The programming model will be essential if we’re to implement multiprocessor architectures in system on a chip.”



much easier than parallelizing sequential programs for parallel architectures.

Vissers: The original question stands: How can we program these multiprocessors if we want them to do one or more tasks? I don’t believe that the VHDL-style languages give us any hope of going in the right direction. Research in parallel languages and computation models is promising.

Bergamaschi: The fallacy of this approach is saying we need a different programming model, which is more parallel and able to handle different computation models. That’s valid but will fall in the same category of having generic hardware architectures for specific applications. It’d be more efficient to model our application more directly to the hardware on which we’re going to run that application. A generic programming model won’t be an efficient solution; we need a more bottom-up model identifying the hardware, on specific architectures, that lets us derive programming components based on them.

Olukotun: We can be very specialized and get lots of performance improvement, lots of performance per area and low power, but what do we do to the compiler and programmer’s model? The challenge is to get both high performance and low power, and also a reasonable programming model.

Bergamaschi: The backdoor solution is a programming model that understands what the underlying hardware is. The opposite of that would be a top-down approach using a generic programming model, which is then compiled onto the hardware. For embedded systems where the hardware may be designed for a spe-

cific application, this generic approach will not lead to efficient solutions.

Vissers: I’m not advocating that we should do it all in classical programmable systems. In the future, we definitely will continuously have to answer one question: Given a particular functionality, can we still map it toward more dedicated blocks? That’s conventionally seen as high-level synthesis, or flavors of hardware design, or reconfigurable system design.

Harr: HDLs are better programming models than they get credit for, because they explicitly represent the cost of communication, and we’re going to face this issue increasingly with multiprocessor architectures.

We have to go back to the people developing the algorithms and, more importantly, the standards. These people start with a theoretically great, mathematically sound technique like the Discrete Cosine Transform, which is being used in the initial JPEG and MPEG standards. But then they start adding control complexity to reduce the bit rates to some target that current technology can support. We then have to live with these design decisions for 20 years or more, long beyond the original problem the standard was meant to solve.

Bolsens: The programming model will be essential if we’re to implement multiprocessor architectures in system on a chip. We will have to come up with a good strategy for software architecture design—as we’ve done in hardware design. If not, we’ll never efficiently map applications onto multiprocessor architectures.

Olukotun: We’ve looked at hardware-software

codesign, and basically decided it's an issue of analyzing and optimizing concurrency, especially if our underlying architecture is a multiprocessor. We decided that Java was a good language for two reasons: It can express explicit concurrency, and it can be analyzed easily to extract implicit concurrency.

Bolsens: However, there's also an issue of data management and data transfer. To design an efficient system on chip, designers want to be able to manage that issue, and Java doesn't allow that.

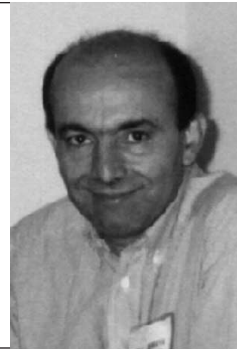
Keutzer: The silicon-capability chip multiprocessor is a manifest destiny. It's not a question of whether it's a good, bad, neutral, or above-average idea. Chip multiprocessors are going to exist. The question is how will they exist: what form, what communication network, and most importantly, what programmer's model? We have two challenges. One is the communication network: Will it go from point-to-point connections to a telecommunications switching network? Will we put it on a chip? The companies I've talked to haven't really sorted that out. The other challenge, of course, is the programming model. We already see tens, perhaps hundreds, of millions of dollars pumped into on-chip multiprocessor projects that lack any significant software support. There's a tremendous opportunity for us there.

Olukotun: What particular interconnection network we pick and how well it works will depend on the application. If we're trying to use the chip multiprocessor as a general-purpose server for transaction processing, then we want something like shared memory. If we're trying to use the multiprocessor as a network processor that's going to do two-layer free routing, then it might look like a network between the processes and have no shared address space whatsoever.

D&T: What do single-chip multiprocessors mean for design and test, in terms of tools we already have and tools we're going to need in the future?

Jerraya: A major problem is how to program these architectures and how to map our appli-

Jerraya:
"...we need ...
something that must
handle concurrency,
communication, and
the specific memory
architectures."



cations on them. Here we need more than a compiler because it's something that must handle concurrency, communication, and the specific memory architectures. We don't have these kinds of tools now.

Vissers: Obviously we'd like high-level tools to help determine whether our designs make sense. That includes performance analysis and power analysis tools. But there's an enormous gap at present; companies like IBM, Phillips, and Motorola are building their own system-level tools and simulators because the current tool suite from CAD vendors to do that isn't available yet.

Bergamaschi: We still have the problem of lower level tools and computation models. One difficulty is how to integrate the architecture with the programming models so that while we're developing hardware, the software designer can also be writing the software, and things can come together at the right time.

As an example of the programming model that we need to evolve to, look at the PlayStation2 chip. The architecture is so specialized that general compilers could not generate code for these units efficiently. How can we ever write a generic compiler that we could somehow parameterize with architecture information and have the compiler generate code for that architecture? The issue is how to work bottom up and identify what kind of knowledge we should put in the compiler so that it understands how a particular architecture works and generate the assembly language for it automatically.

Keutzer: We're talking about families of architectures, of communication networks and of



Harr:
 “The third class [of tools] ... involves moving from the algorithm space to the application implementation space.”

processing elements. We need tools that let us explore which family member to choose. For the processing elements, we need a way to automatically generate an instance of that family, and then create software environments that compile the application. We need performance analysis tools. For communication networks, ideally we'd have a synthesis tool to analyze the application and its data flow. What's less clear is whether it's possible to automate that synthesis or whether we should focus more on performance analysis tools to help our human designer build the communication network.

Bolsens: We must determine how we're going to design the software architecture and what the different abstraction levels will be—what will be the generic components in the software architectural refinement. Good APIs are probably one of many things we must consider in software architecture.

Harr: Traditionally, multiprocessor design has been a custom process involving a few key architects; now it seems we want to turn it into a formalized, standard process involving many designers. We'll have three classes of tools that must work together. The first class involves taking an application specification to an appropriate architecture, where we're basically analyzing whether an application will work efficiently in a given form on an existing architecture. The second class of tools, which has generally been worked on for the past 10 or 20 years in the EDA community, concerns the architecture down to the IC design space. These need some improvements, but the tools are pretty good. The third class, and one needing a lot of work, involves moving from the algorithm

space to the application implementation space.

Currently, the architectural analysis tools include things like instruction set simulators, in-circuit emulators, assembly code development systems, and high-level language compilers. We've got to build a whole new class of tools to aid the more efficient algorithm translation into an efficient implementation on the architecture.

Vissers: So far we haven't mentioned that these embedded systems have specific time requirements. Real-time requirements are the additional multiprocessor requirement that's completely underestimated in the classical programmer's compiler world so far. We need a mechanism to guarantee that the system is doing what it should with the time response. That's why everybody still thinks that truth is in gate-level simulation; otherwise we don't know what we're really getting.

Gupta: The second class of tools is going from architecture to implementation. There's one big difference in talking about on-chip multiprocessors in terms of these ISS tools. Currently, most of our tools are singular design tools that build and load efficiently, such as those for discrete-event simulators. But remember, the tool will have to consider node performance in the context of other nodes. For example, a protocol compiler that works with a specification for node implementation isn't enough when it comes to multiprocessors. We need an NS (network simulator) view in which we model the multinodal system first, even while we're designing a single node. That's a key concept tool design, apart from the integration itself.

Vissers: We're going to need a lot of research in the application-to-architecture tool space because it comes down to how our programming model can change the way our application maps to a particular architecture. The programming model will dramatically affect the sort of parallelism and communication that we can extract from the application—that's the issue.

Keutzer: For these multiprocessor systems, will the primary medium be, say, a conventional

programming language with some additional concurrency or p-threads, or will we go to more-computational models?

Gupta: There's no way to avoid going to an application-level exposition of parallelism. We'll have to move when the technology makes it possible.

Bolsens: Perhaps we can learn from the telecommunications world? Complex multiprocessor architectures can inherit from the solutions provided in the telecommunications backbone architectures, including switches and routers, for example. Perhaps at some time we will get a kind of ATM backbone on a chip. Further, why can't some of the techniques used to build complex distributed systems be applied to program massive multiprocessors on a chip? We of course have to add a performance criterion, which is less of an issue in the distributed-computing systems as they're now being supported by CORBA. Perhaps that's a good starting point.

Keutzer: I don't see these computation models and application-specific environments progressing fast enough such that the technology we need will be available when the multiprocessors are available. Realistically, we're going to be programming these things in languages we know today, like C++ or Java, perhaps with an additional layer of communication structure.

Harr: A lot of high-level computation models are good for certain situations; we have vector processing and vector languages that are good for certain applications and still used heavily in, for example, nuclear simulations and the like. To have physically heterogeneous processors, we need heterogeneous system description capabilities and a way of tying them all together so that we have a "best of class" for each type of application that needs to be described.

D&T: Are there any wrap-up comments?

Jerraya: We'll see specialization when it comes to system-on-chip multiprocessor designs. It's happening already in several domains like game processors or network processors.

Keutzer:
"...we're going to be programming... in...C++ or Java, perhaps with [a]...layer of communication structure."



Bergamaschi: We need to tie the hardware implementation architecture to the software architecture. One way to do that is to use a component-based programming model. The software developer doesn't have to know what's inside the component. The hardware architecture developer could generate that component specifically for any given part, but the component itself wouldn't change as far as the API. That's the model that might allow the software developer to make the best use of the architecture.

D&T: Thank you all very much.

About the participants

Reinaldo Bergamaschi is a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, N.Y.

Ivo Bolsens is vice president of design technology for the Integrated Information and Communication Systems Division at IMEC, Leuven, Belgium.

Rajesh Gupta is an associate professor of information and computer science at the University of California, Irvine.

Randolph Harr is vice president of engineering at Intransa Inc., Santa Clara, Calif.

Ahmed Jerraya is group leader, System Level Synthesis Dept., at TIMA, Grenoble, France.

Kurt Keutzer is a professor of electrical engineering and computer sciences at the University of California, Berkeley.

Kunle Olukotun is an associate professor of electrical engineering and computer science, Stanford University, Stanford, Calif.

Kees Vissers is principal architect at TriMedia Technologies Inc., Milpitas, Calif.

Wayne Wolf, our moderator, is a professor of electrical engineering at Princeton University, Princeton, N.J.