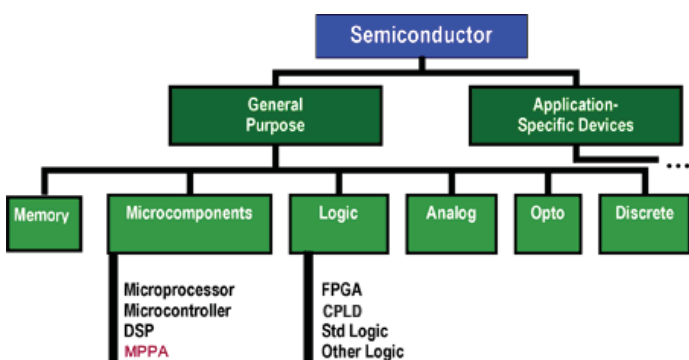




Technology Overview

Massively Parallel Processing Arrays

The Ambric architecture is a member of an emerging class of parallel chips called Massively Parallel Processor Arrays. MPPAs are distinguished from “multi-core” conventional processors, which have only a few processors and a shared-memory architecture, by having massive parallelism of at least hundreds of processing elements and distributed memories, and a rich word-wide flexible interconnect fabric.



MPPA = Massively Parallel Processor Array

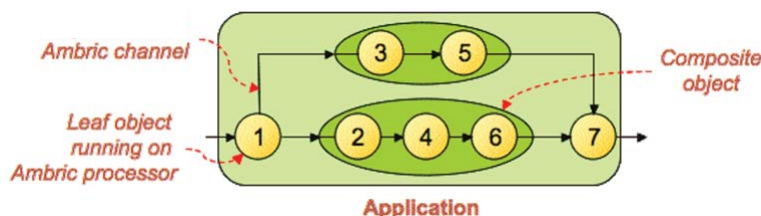
Embedded Computing Problem

Complex high-bandwidth, real-time, streaming-media, sensor, and network applications for embedded systems can be achieved by programming a massively parallel processor array (MPPA) with software rather than using the hardware design required for ASICs and FPGAs. But a fundamental problem is how to productively program and validate a complex, irregular application on hundreds of processors. With conventional multiprocessing techniques, keeping processors busy, communicating, and synchronized is difficult and prone to failure, and won't scale up to hundreds of CPUs.

ASICs and high-end FPGAs are getting harder and more expensive to develop. Global timing convergence, million-dollar plus ASIC NREs, FPGA static power issues and the RTL hardware design productivity gap between available and delivered ASIC gates are among the growing difficulties.

Structural Object Programming Model

Ambric's structural object programming model is the foundation of how it all works:



An Ambric chip is an array of hundreds of 32-bit RISC processors that are each programmed with ordinary software, and hundreds of distributed memories. These are called objects because they obey strict encapsulation rules enforced in hardware. Objects simply run independently on their own parallel hardware, not shared or multi-threaded or virtualized. Structural objects are strictly encapsulated, execute with no side effects on one other, and have no implicitly shared memory.

Ambric objects communicate through a simple parallel structure of hardware channels. Each channel is word-wide, unidirectional, point-to-point from one object to another, and acts like a FIFO-buffer. Channels carry both data and control tokens, in simple or structured messages. Ambric channels are self synchronizing. Channel hardware synchronizes its objects at each end, dynamically as needed at run time, not scheduled at compile time.

Inter-processor communication and synchronization is straightforward. Sending or receiving a word on a channel is as simple as reading or writing a processor register. Sending a word through a channel is both a communication and a synchronization event. This keeps objects in step with each other in a common-sense way. Since Ambric channels synchronize transparently and locally, the application achieves high performance without complex global synchronization.

Channels provide a common hardware-level interface for all objects. This makes it easy for objects to be assembled into higher-level composite objects. They work the same way as leaf objects, since objects are encapsulated and only interact through channels. Design re-use is practical and robust in this system.

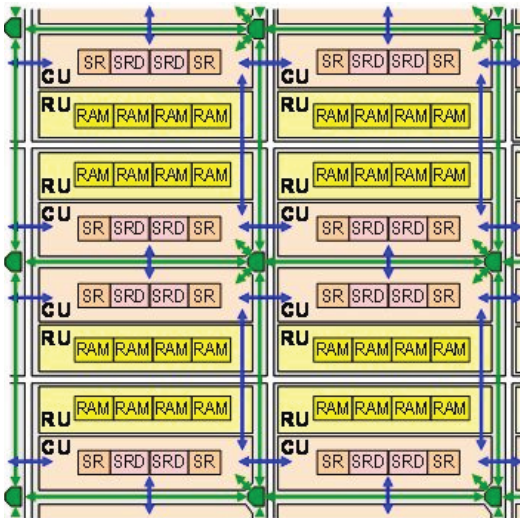
Chip Architecture: Compute Unit, RAM Unit

In Ambric's chip architecture, a cluster of four processors is a Compute Unit (or CU). It has two types of processors:

The SRD processor is a 32-bit streaming RISC processor with DSP extensions for math-intensive processing. It has local memory for its 32-bit instructions, and can directly execute more code from the RAM Unit next door.

The SR processor is a simpler 32-bit streaming RISC CPU, used mainly for managing channel traffic, generating complex address streams, and other utility tasks which enable sustained high throughput for the SRDs.

The RAM Units (or RUs) are the main on-chip memories that can stream addresses and data over channels.



Chip Architecture: Brics and Interconnect

CUs and RUs are combined into the top-level physical building-block called a bric. The core of the chip is assembled just by stacking up brics. The number of brics serves as a measure of compute-capacity.

Each bric has two CU-RU pairs, totaling 8 CPUs and 21 KBytes of SRAM.

Brics connect by abutment through channels that cross bric-to-bric. The CUs and RUs are arranged so that, in the array of brics, there are contiguous CUs and contiguous RUs.

To optimize the cost and performance of each channel, the chip's configurable interconnect is hierarchical with several levels of hierarchy. These channels are word-wide and run at up to 10 Gigabits per second.

Chip Implementation

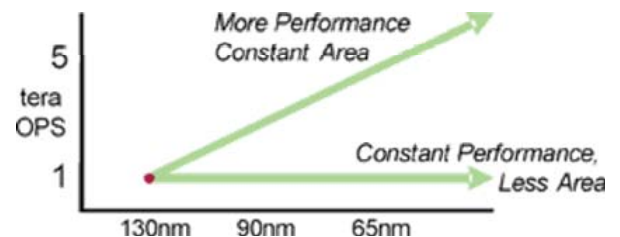
The core of Ambric's has 45 brics in an array, containing a total of 336 32-bit SR and SRD processors running at 350 MHz and 7.1 Mbits of distributed SRAM. At full speed, all processors together are capable of 1.2 trillion operations per second, over one teraOPS. This performance is supported by the interconnect's 792 gigabit per second bisection bandwidth, 26 Gbps of off-chip DDR2 memory, PCI Express at an effective 8 Gbps each way, and up to 13 Gbps of parallel general-purpose I/O.

Scalability

Ambric set out to overcome fundamental barriers to scalability in development cost and difficulty, timing, and power.

Our programming model's object-based modularity makes much greater design reuse possible. Its simple combination of objects written in normal software code, combined in a hierarchy of block-diagram structures, makes high-performance design development much easier and cheaper.

Our system of processors, memories and self-synchronizing channels, with local synchronous clocking and no long wires, is very scalable into future process generations.



Conclusion

Ambric has found practical solutions to the architectural and programming challenges that have stood in the way of achieving massively parallel embedded computing that delivers extremely high performance, which is silicon- and software-scalable long-term, and with reasonable software-only application development effort.

It's not enough to put lots of processors on a chip without thinking about how they're programmed. We started with our Structural Object Programming Model, which scales without limit, and designed our silicon to enable it. Its modularity, parallel low-power, and local timing, mean Ambric's object-based technology can continue to track Moore's Law for many process generations to come.