Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge

A Survey of Multicore Processors

A review of their common attributes

© PHOTO F/X2

General-purpose multicore processors are being accepted in all segments of the industry, including signal processing and embedded space, as the need for more performance and general-purpose programmability has grown. Parallel processing increases performance by adding more parallel resources while maintaining manageable power characteristics. The implementations of multicore processors are numerous and diverse. Designs range from conventional multiprocessor machines to designs that consist of a "sea" of programmable arithmetic logic units (ALUs). In this article, we cover some of the attributes common to all multicore processor implementations and illustrate these attributes with current and future commercial multicore designs. The characteristics we focus on are application domain, power/performance, processing elements, memory system, and accelerators/integrated peripherals.

INTRODUCTION

Parallel processors have had a long history going back at least to the Solomon computer of the mid-1960s. The difficulty of programming them meant they have been primarily employed by scientists and engineers who understood the application domain and had the resources and skill to program them. Along the way, a surprising number of companies created parallel machines. They were largely unsuccessful since their difficulty of use limited their customer base, although, there were exceptions: the Cray vector machines are perhaps the best example. However, these Cray machines also had a very fast scalar processor that could be easily programmed in a conventional manner, and the vector programming paradigm was not as daunting as creating general parallel programs. Recently the evolution of parallel machines has changed dramatically. For the first time, major chip manufacturers—companies whose primary business is fabricating and selling microprocessors—have turned to offering parallel machines, or single chip multicore microprocessors as they have been styled.

There are a number of reasons behind this, but the leading one is to continue the raw performance growth that customers have come to expect from Moore's law scaling without being overwhelmed by the growth in power consumption. As single core

Digital Object Identifier 10.1109/MSP.2009.934110

1053-5888/09/$26.00©2009IEEE

IN THE PAST DECADE, POWER HAS JOINED PERFORMANCE AS A FIRST CLASS DESIGN CONSTRAINT.

designs were pushed to ever higher clock speeds, the power required grew at a faster rate than the frequency. This power problem was exacerbated by designs that attempted to dynamically extract extra performance from the instruction stream, as we will note later. This led to designs that were complex, unmanageable, and power hungry. The trend was unsustainable. But ever higher performance is still desired as evident by predictions from the ITRS Roadmap [1] predicting a need for 300x more performance by 2022 as shown in Figure 1. To meet these demands, chip designers have turned to multicore processors and parallel programming to continue the push for more performance, and in turn the ITRS Roadmap has projected that by 2022, there will be chips with upwards of 100x more cores than on current multicore processors. The main advantage to multicore systems is that raw performance increase can come from increasing the number of cores rather than frequency, which translates into a slower growth in power consumption. However, this approach represents a significant gamble because parallel programming science has not advanced nearly as fast as our ability to build parallel hardware.

General-purpose multicores are becoming necessary even in the realm of digital signal processing (DSP) where, in the past, one general-purpose control core marshaled many special purpose application-specific integrated circuits (ASICs) as part of a "system on chip." This is primarily due to the variety of applications and performance required from these chips. This has driven the need for more general-purpose processors. Recent examples would include software-defined radio (SDR) base stations, or cell phone processors that are required to support numerous codecs and applications all with different characteristics, requiring a general programmable multicore.

ARCHITECTURE CLASSIFICATIONS

Multicore architectures can be classified in a number of ways. In this section we discuss five of the most distinguishing attributes: the application class, power/ performance, processing elements, memory system, and accelerators/ integrated peripherals.

APPLICATION CLASS

If a machine is targeted to a specific application domain, the architecture can be made to reflect this. The result is a design that is efficient for the domain in question but often ill-suited to other areas. The extreme example is an ASIC. Tuning to an application domain can have several positive consequences. Perhaps the most valuable is the potential for significant power savings. Conventional DSPs are a good example.

There are two broad classes of processing into which an application can fall: data processing dominated and control dominated.

DATA PROCESSING DOMINATED

Data processing-dominated applications contain many familiar types of applications including graphics rasterization, image processing, audio processing, and wireless baseband processing. Many of the classic signal processing algorithms are part of this group. The computation of these types of applications is typically a sequence of operations on a stream of data with little or no data reuse. The operations can frequently be performed in parallel and often require high throughput and performance to handle the large amounts of data. These kind of applications favor designs that have as many processing elements as practical in regards to desired power/performance ratio.

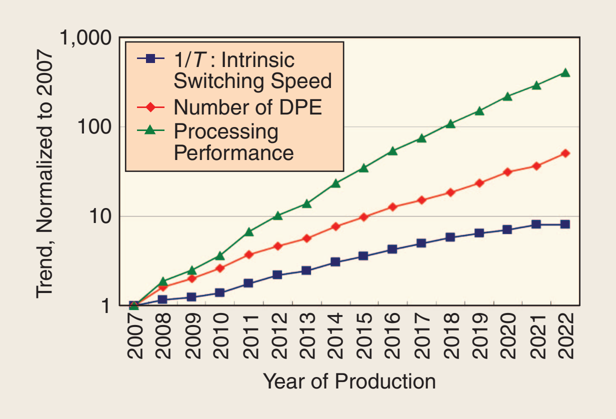
CONTROL PROCESSING DOMINATED

Control-dominated applications include file compression/ decompression, network processing, and transactional query processing. The code for these types of applications tend to be dominated by conditional branches, complicating parallelism. The programs themselves often need to keep track of large amounts of state and often have a high amount of data reuse. These types of applications favor a more modest number of general-purpose processing elements to handle the unstructured nature of control dominated code.

In almost all cases, no application can fit into these neat divisions, but execution phases of an application may. For instance the H.264/AVC [4] video codec is data dominant when performing the block filter, but control dominated when compressing or decompressing video using context-adaptive binary arithmetic coding (CABAC) compression. It is valuable to think of applications as falling into these divisions to understand how different multicores design aspects can affect performance. An unbalanced architecture may do very well on the data dominated portion of the H.264/AVC, but be very inefficient for CABAC encoding/decoding, leading to less than desired performance.

POWER/PERFORMANCE

Many applications and devices have strict performance and power requirements. For instance, a mobile phone that wants to



[FIG1] ITRS Roadmap [1] for frequency, number of data processing elements (DPE) and overall performance.

support video playback has a strict power budget, but it also has to meet certain performance characteristics.

THE MAIN ADVANTAGE TO MULTICORE SYSTEMS IS THAT RAW PERFORMANCE INCREASE CAN COME FROM INCREASING THE NUMBER OF CORES RATHER THAN FREQUENCY.

Performance has been the traditional goal. In the past decade, power has joined performance as a first-class design constraint. This is, in large part, due to the rise of mobile phones and other forms of mobile computing where battery life and size are critical. More recently, power consumption has also become a concern for computers that are not mobile. The driving force behind this is the growth in data centers to support "cloud" computing. The typical general-purpose multicore processor is ideally suited to these centers, but these centers are now consuming more energy than heavy manufacturing in the United States. They are so large—Google now refers to them as warehouse-scale computers—that the power consumption of the basic multicore component is critical to the cost and operation.

PROCESSING ELEMENTS

In this section, we cover the architecture and microarchitecture of a processing element. The architecture, or more fully the instruction set architecture (ISA), defines the hardware-software interface. The microarchitecture is the implementation of the ISA.

ARCHITECTURE

In conventional multicore processors, the ISA of each core is typically a legacy ISA from the corresponding uniprocessor with minor modifications to support parallelism such as the addition of atomic instructions for synchronization. The advantages to legacy ISAs is the existence of implementations and the availability of programming tools. An ISA may also be custom defined.

ISAs can be classified as reduced instruction set computer (RISC) or complex instruction set computer (CISC). Although this was a controversial distinction years ago, in today's designs, the microarchitectural distinctions have been blurred: most CISC machines look very much like their RISC counterparts once decoding has been done. On the code front, the differences are still distinct. CISC has the edge in code size due to the greater selection of instructions and richer semantics available. RISC, on the other hand, has larger code sizes due to the need to emulate more complex instructions with the smaller set of RISC instructions. The advantage of RISC is it provides an easier target for compilers and allows for easier microarchitectural design.

Beyond the base definition of the ISA, vendors have been continually adding ISA extensions to improve performance for common operations. Intel has added MMX, MMX2, and SSE1-4 [5] to improve multimedia performance. ARM has added similar instructions for multimedia with its NEON [6] instruction set. These instructions allow for a better performance/power consumption ratio as specialized hardware can do operations like a vector transpose in one instruction. The soft-core provider Tensilica has made this the main selling point for their Xtensa CPUs [7], offering customizable special purpose instructions for specific designs.

MICROARCHITECTURE

The processing element microarchitecture governs, in many respects, the performance and power consumption that can be expected from the multicore. The microarchitecture of each processing element is often tailored to the application domain that is targeted by the multicore machine. Although the commercial offerings of the major chip manufacturers like Intel employ numbers of identical cores into a homogeneous architecture, it is often advantageous to combine different types of processing elements into a heterogeneous architecture. The idea is again to obtain a power advantage without loss of performance. A typical organization has a control processor marshaling the activities of an ensemble of simpler "data plane" cores. In data-dominated applications, such architectures can often provide high performance at low power. The drawback is that the programming model for heterogeneous architectures is much more complicated.

The simplest type of processing element is the in-order processing element. This type of processing element decodes and executes instructions in program order and dynamically accounts for data forwarding and control hazards. There are two main performance parameters that can be modified to get the desired performance. First, multiple pipelines can be added to fetch and issue more than one instruction in parallel, creating a superscalar processing element to increase performance. However, increasing issue width requires extra logic to provide more complex data forwarding paths and hazard detection to assure correct code execution in the pipelines. The complexity of the logic grows greater than quadratically with the number of pipelines, and a point of diminishing returns is quickly reached. Experiments with general-purpose applications suggest that point is about three to four pipelines, but of course this is highly dependent on the applications. Second, performance can also be improved by increasing the number of pipeline stages, thus reducing the logic per stage. This enables a faster clock at the expense of greater penalty if the instruction sequence is broken by branches. In-order elements have small die area, low power, and are easily combined in large numbers if an application has abundant thread level parallelism (TLP) and few performance sensitive serial sections. For example, NVIDIA's G200 [8] gangs together 240 in-order cores because graphics processing is highly parallel with few serial sections.

Taking the superscalar core further to gain as much single thread performance as possible is the out-of-order architecture. It attempts to dynamically find and schedule multiple instructions "out of order" to keep the pipelines full. The

IT IS NOT UNCOMMON FOR MULTICORE DESIGNS TO OMIT CACHE COHERENCE TO REDUCE DESIGN AND VERIFICATION COMPLEXITY.

dynamic scheduling requires very complex and power hungry circuitry to keep track of all in-flight instructions. Out-of-order designed cores are most suitable for applications that have a wide range of behaviors and high performance is needed. However, logic complexity means that this type of processing element is not power efficient and requires substantial die area. Most out-of-order processors are multi-issue, as single-issue out-of-order processors do not have much advantage over a simpler in-order core. Because the out-of-order core is large and power hungry, very few can be combined in practice. However, they are preferable if the applications to be run are control dominated and have large critical serial portions and moderate TLP. For example, the ARM Cortex A9 [6] is targeted for netbook computers, and requires single thread performance over TLP, so it utilizes a handful of out-of-order cores.

To increase performance over superscalar architectures, but eliminate the complexity of the extra logic needed to properly execute the instruction stream, single-instruction, multiple-data (SIMD) or very long instruction word (VLIW) architectures can be used. The SIMD architecture makes use of very wide registers split into lanes to process multiple data points with one instruction. A simple example is the addition of two vectors element-wise. Each pair of elements is processed in its own lane. This style of architecture is well suited for data intensive applications that are data parallel. An example is the IBM Cell [9] that uses many SIMD cores targeted towards data dominated applications. A SIMD architecture is highly inefficient for general-purpose processing.

To avoid being limited to one instruction processing multiple data points, a VLIW can be used. VLIW uses multiple pipelines but does not typically have the forwarding, scheduling, and hazard detection logic of a superscalar core. Instead, the compiler is relied upon to group instructions into packets that can be executed in parallel and guarantee no data or control hazards—the complexity has been moved to the compiler. VLIW execution allows for very wide machines that can process multiple data points with multiple instructions at the same time, giving it a distinct advantage over SIMD. But VLIW can suffer severe under utilization problems if the compiler cannot find sufficient parallelism. VLIW and SIMD are both high-performance and power-efficient designs but are usually well suited for only very specific types of application codes with large numbers of independent operations that can found by compilers or the programmer. Architectural and microarchitectural design parameters are summarized in Table 1.

MEMORY SYSTEM

In uniprocessor designs, the memory system was a rather simple component, consisting of a few levels of cache to feed the single processor with data and instructions. With multicores, the caches are just one part of the memory system, the other components include the consistency model, cache coherence support, and the intrachip interconnect. These determine how cores communicate impacting programmability, parallel application performance, and the number of cores that the system can adequately support.

CONSISTENCY MODEL

A consistency model defines how the memory operations may be reordered when code is executing. The consistency model determines how much effort is required by the programmer to write proper code. Weaker models require the programmer to explicitly define how code needs to be scheduled in the processor core and have complex synchronization protocols. Stronger models require less effort and have simpler synchronization protocols. On the other hand, the consistency models have an effect on performance. Strong consistency models place strict ordering constraints on how the memory system is allowed to propagate reads and writes to other processing elements. For example, sequential consistency requires all processors in a

[TABLE 1] SUMMARY OF PROS AND CONS OF VARIOUS CORE DESIGN PARAMETERS.

ISA	**PRO**	**CON**
LEGACY	COMPILER AND SOFTWARE SUPPORT	MAY BE INEFFICIENT FOR TARGETED APPS REQUIRING HIGH PERFORMANCE
CUSTOM	CAN BE HIGHLY OPTIMIZED FOR TARGET APPS	COMPILER AND SOFTWARE SUPPORT CAN BE NONEXISTENT
RISC	EASIER MICROARCH DESIGN, EASIER COMPILER DESIGN	CODE SIZE CAN BE LARGE, INEFFICIENT FOR CERTAIN APPS
CISC	MORE INSTS THAT MAY ALLOW FOR BETTER OPTIMIZATION, SMALLER CODE SIZE	COMPLEX MICROARCH DESIGN TO SUPPORT ALL INSTS, COMPILER DESIGN COMPLICATED
SPECIAL INSTS	ALLOWS HIGHLY OPTIMIZED CODE FOR TARGETED FUNCTIONS	COMPLEX TO DESIGN, MAY REQUIRE HAND CODING DUE TO LIMITED/NO COMPILER SUPPORT
MICROARCH	**PRO**	**CON**
IN-ORDER	LOW TO MEDIUM COMPLEXITY, LOW POWER, LOW AREA SO MANY CAN BE PLACED ON DIE	LOW TO MEDIUM SINGLE THREAD PERFORMANCE IN GENERAL
OUT-OF-ORDER	VERY FAST SINGLE THREAD PERFORMANCE FROM DYNAMIC SCHEDULING OF INSTS	HIGH DESIGN COMPLEXITY, LARGE AREA, HIGH POWER
SIMD	VERY EFFICIENT FOR HIGHLY DATA-PARALLEL/VECTOR CODE	CAN BE UNDER-UTILIZED IF CODE CAN NOT BE VECTORIZED, NOT APPLICABLE TO CONTROL-DOMINATED APPLICATIONS
VLIW	CAN ISSUE MANY MORE INSTRUCTIONS THAN OUT-OF-ORDER DUE TO REDUCED COMPLEXITY	REQUIRES ADVANCED COMPILER SUPPORT, MAY HAVE WORSE PERFORMANCE THAN NARROWER OUT-OF-ORDER CORE IF COMPILER CAN NOT STATICALLY FIND ILP

system to see that all reads and writes occur in the same order globally and in program order. This can severely impact performance but makes programming simple as it is easy to reason about how parallel code will operate. Conversely, weak consistency allows reads and writes in the system to be seen in any order by all processors. Because weak consistency models allow this memory reordering, primitives known as barriers and fences are added to the instruction set. These primitives allow programmers to enforce stricter consistency on memory accesses when needed, such as an access to a synchronization variable.

Two consistency models are illustrated in Figure 2. In Figure 2, each of the processors P1–P4 are issuing write (e.g., $X = 1$) and read (e.g., $R(Z) = 0$) requests. The memory model of a sequential system states that all reads and writes to all addresses are observed to be in the same order. This means that when P2 reads Z, the value returned should be two, as a result of the earlier write by P4. This is accomplished by the processing elements and the memory system establishing a global ordering of all requests, typically enforced by the arbitration on an interconnection network. In the weak consistency case, P2 reads Z and the result returned is zero. In this case the consistency model allows different cores to see a different global ordering of events.

Weak consistency models make the memory system easier to design but place an onus on the programmer to correctly identify and place instructions in the program that enforce proper behavior. On the other hand, sequential consistency makes programming easier but makes the memory system more complicated and slower as it is unable to take advantage of performance gains that can be had by allowing memory operations to complete out-of-order.

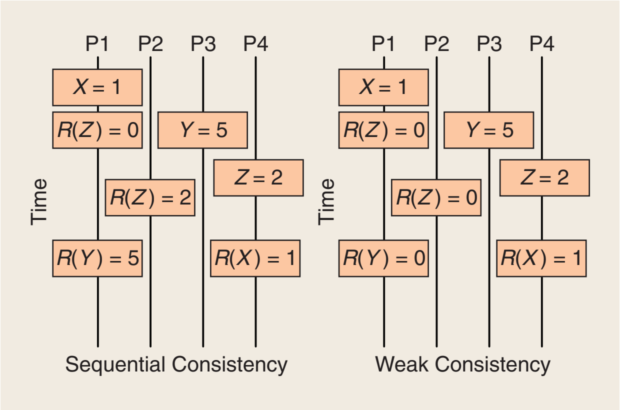
CACHE CONFIGURATION

Caches have increased importance in multicore processors. They give processing elements a fast, high bandwidth local memory to work with. This is of particular importance as an increasing number of cores are trying to access the relatively slow, low-bandwidth, off-chip memory.

Caches can be tagged and managed automatically by hardware or be explicitly managed local store memory. Automatically tagged caches are the most common form as they are transparent to the instruction stream which believes it has access to one uniform memory. The main drawbacks to automatically managed caches is they have nondeterministic performance and use die area for storing tags for each entry. Local stores conversely can provide deterministic performance because they are managed explicitly by the executing software stream and offer more storage for the same area because they do not need tags. This software management can be cumbersome and in most cases is only preferred by applications that require hard real-time performance requirements.

The amount of cache required is very application dependent. Bigger caches are better for performance but show diminishing returns as caches sizes grow. Large caches may also not be of use to applications where data is only used once, such as video decoding. In these situations, it may be desirable to have cache modes that distinguish between streaming accesses and normal accesses. The Microsoft Xenon [10] has this functionality to prevent cache pollution with data that will not be reused. Cache sizes are usually made as big as the die area and power budget will allow. This is a trend seen in control processing dominated architectures with heavy data reuse like the Intel Core i7 [2].

The number of cache levels has been increasing as processing elements both get faster and become more numerous. The driving consideration is how far away the main memory is, in cycles, from each processing element. The greater the number of cycles away, the greater the need for more cache levels. The first level of cache is usually rather small, fast, and private to each processing element. Subsequent levels can be larger, slower, and shared among processing elements. These levels are used to present the illusion that a processing element has access to a very fast memory when, in fact the main memory may be hundreds of cycles away. This is the case for server class multicores like the AMD Phenom [11] that have upwards of three levels of cache. For embedded multicores the main memory may be a few tens of cycles away and one level of cache may be sufficient conserving both die area and power. But even embedded cores are seeing frequency increases, the Texas Instruments (TI) OMAP4430 [12] will be clocked at 1 GHz, so caches will continue to gain importance to hide the widening gap in memory latency and bandwidth.



[FIG2] Illustration of consistency models.

INTRACHIP INTERCONNECT

The intrachip interconnect is responsible for general communication among processing elements and cache coherence (if present). There are many styles of interconnects for intracore communications, such as bus, crossbar, ring, and network-on-chip (NoC). Each type has advantages and disadvantages in terms of simplicity and performance. For example, the bus is the simplest to design but quickly becomes bandwidth and latency limited when trying to scale up to a large number of processing elements. The NoC, on the other hand, scales very well with the number of processing elements but is more challenging to design.

The interconnect also provides cache coherence, a very important feature because it governs the type of programming

[TABLE 2] SUMMARY OF PROS AND CONS OF MEMORY SYSTEM DESIGN DECISIONS.

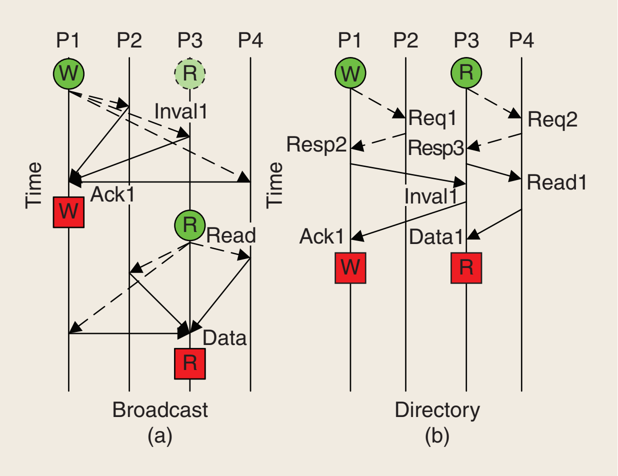
ON-DIE MEMORY	PRO	CON
CACHES	TRANSPARENTLY PROVIDE APPEARANCE OF LOW LATENCY ACCESS TO MAIN MEMORY, CAN BE CONFIGURED EASILY INTO MULTIPLE LEVELS	NO REAL-TIME PERFORMANCE GUARANTEE, NEED TO USE DIE AREA TO STORE TAGS
LOCAL STORE	CAN STORE MORE DATA PER DIE AREA AS CACHES, PROVIDE REAL-TIME GUARANTEE	MUST BE SOFTWARE CONTROLLED
COHERENCE	**PRO**	**CON**
YES	PROVIDES A SHARED MEMORY MULTIPROCESSOR, SUPPORTS ALL PROGRAMMING MODELS	HARD TO IMPLEMENT
NO	EASY TO IMPLEMENT	RESTRICTS PROGRAMMING MODELS SUPPORTED
INTERCONNECT	**PRO**	**CON**
BUS	EASY TO IMPLEMENT, ALL PROCESSORS SEE UNIFORM LATENCIES TO EACH OTHER AND ATTACHED MEMORIES	LOW BISECTION BANDWIDTH, SUPPORTS SMALL NUMBER OF CORES
RING	HIGHER BISECTION BANDWIDTH THAN BUS, SUPPORTS LARGE NUMBER OF PROCESSORS	NONUNIFORM ACCESS LATENCIES, HIGH VARIANCE IN ACCESS LATENCIES, REQUIRES ROUTING LOGIC
NoC	HIGH BISECTION BANDWIDTH, SUPPORTS LARGE NUMBER OF CORES, NONUNIFORM LATENCIES ARE LOWER VARIANCE THAN RING	REQUIRES SOPHISTICATED ROUTING AND ARBITRATION LOGIC
CROSSBAR	HIGHEST BISECTION BANDWIDTH, CAN SUPPORT LARGE NUMBER OF CORES, UNIFORM ACCESS LATENCIES	REQUIRES SOPHISTICATED ARBITRATION LOGIC, NEEDS LARGE AMOUNT OF DIE AREA

models the architecture supports. Cache coherence maintains a single image of memory automatically visible to all processors in the system and is essential for programming models that implicitly depend on shared memory. It is very common in general-purpose processors like the ARM Cortex A9 [6]. Two ways coherence can be implemented are broadcast based or directory based.

Broadcast-based coherence is simple; it achieves this by using the interconnect to only allow one processor at a time to perform an operation that is visible to all other processors. This is illustrated in Figure 3(a). In the broadcast protocol when a write (circled W) occurs, a single invalidate request (dashed lines) is sent to all the other processors to gain the proper permissions to perform the write. The processor holding the data returns the value to P1 (solid line) and the write is performed (square W). Because the broadcast protocol (usually on a bus) occupies the entire interconnect, the read (dashed line circled R) by P3 must be delayed until the write is completed. At that time the read (circled R) requests in a single request seen by all other processors (dashed line), and the data is returned by the current owner, completing the read (square R). Overall, for small numbers of processors the broadcast based approach is reasonable. Usually on the order of eight cores can be supported as is the case for the Intel Core i7 [2].

Directory-based coherence, on the other hand, scales to larger numbers of processors than broadcast-based coherence because it enables multiple coherence actions to occur concurrently. Directory coherence works by having nodes query a distributed directory. The directory contains information about which caches contain each memory address. Each address is assigned a home node where its portion of the directory is stored. When an access is required, the processor will query the home node of that address to obtain a list of processors currently holding that cache block. The requestor, in turn, gains access rights from all the relevant processors. Figure 3(b) shows how a directory scheme can perform multiple operations in parallel. In the directory protocol, the write performed by P1 first queries the home node (P2) of the address to determine the current owner/sharers (P3) of that cache block. P2 responds with the list and P1 then sends out an invalidate request individually to each owner/sharer. Each node will respond with an acknowledgment. Once P1 has received all the acknowledgments it can perform the write. A similar process is done for the read from P3 to the home node (P4) and owner (P4). In this case, since the network is not entirely occupied by a broadcast, both the read and write can be performed in parallel. Directory coherence is suitable for weak consistency models and large systems (tens to hundreds of cores), such as the Tilera TILE64 [13].

It is not uncommon for multicore designs to omit cache coherence to reduce design and verification complexity. A number of current multicore processors lack cache coherence, examples include the TI TMS320DM6467 [14] and the IBM Cell [9]. The lack of cache coherence means that the software must enforce the desired memory state seen by all the cores during execution. This limits the programming models to variants of message passing. For application domains that only



[FIG3] Illustration of (a) broadcast and (b) directory coherence.

share a limited amount of memory between cores, this can be a practical option. Memory system design decisions are summarized in Table 2.

ACCELERATORS/INTEGRATED PERIPHERALS

Accelerators or integrated peripherals are typically ASICs or highly specialized processors that can not be efficiently emulated by software. Some examples include graphics rasterizers, codec accelerators, memory controllers, etc. These are usually one of the smaller contributors to power consumption but can have a large impact on overall performance in many cases. An example of accelerators would be the image processing engines found on the TI OMAP4430 [12].

SOME CURRENT COMMERCIAL MULTICORES

In recent years, there has been a wide of range of multicore architectures produced for the commercial market. They have targeted every market segment from embedded to general-purpose desktop and server realms. As we have noted before, this is in large part due to the desire for increased performance with acceptable power increases.

The first four entries of Table 3 shows a selection of general-purpose multicores. The microarchitecture of their cores is traditional and based on a powerful conventional uniprocessor. They all employ a modest number of identical copies of these cores with large caches. These chips are intended for applications found in the desktop and server markets in which power is not an overriding concern. The remaining entries in Table 3 are also multicores but ones for general-purpose mobile and embedded applications. They too have identical general-purpose cores that are well suited to control dominated applications. Power is an overriding concern for these chips because many are intended to run from batteries and in all but one case they consume 1 W or less.

The next set of architectures, shown in Table 4, are more specialized and are targeted to high-performance computing. These architectures target high performance in their application domain and, for the most part, employ significant numbers of cores—for the AMD R700 and NVIDIA G200, this number is in the hundreds. The IBM Cell implements a heterogeneous architecture with a modest number of very specialized data processing engines. These designs are generally very high power ranging from 100 W to 180 W.

Table 5 presents multicore architectures that are specialized for specific application domains. They exhibit the most variety. Most of them target data dominated application domains such as wireless baseband, and audio/visual codecs where simple parallelism can often be exploited. Accordingly, they support high computation rates. Many feature interconnection networks that are tuned to the needs of their intended application domain. These architectures achieve these high computation rates without excessive power requirements.

In the remainder of this section we will discuss several multicores in more depth. They are selected from several distinct categories: server, mobile, graphics, and DSP.

[TABLE 3] TABLE OF GENERAL-PURPOSE SERVER AND MOBILE/EMBEDDED MULTICORES.

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD PHENOM [11], [15]	X86	THREE-WAY OUT-OF-ORDER SUPERSCALAR, 128-B SIMD	FOUR	64 KB IL1 AND DL1/CORE, 256 KB L2/CORE, 2-6 MB L3	DIRECTORY	POINT TO POINT	PROCESSOR	140 W	2.5 GHZ–3.0 GHZ	12–48 OPS/CLOCK
INTEL CORE I7 [2], [5]	X86	FOUR-WAY OUT-OF-ORDER, TWO-WAY SMT, 128-B SIMD	TWO TO EIGHT	32 KB IL1 AND DL1/CORE, 256 KB L2/CORE, 8 MB L3	BROADCAST	POINT TO POINT	PROCESSOR	130 W	2.66 GHZ–3.33 GHZ	8–128 OPS/CLOCK
SUN NIAGARA T2 [16], [17]	SPARC	TWO-WAY IN-ORDER, EIGHT-WAY SMT	EIGHT	16 KB IL1 AND 8 KB DL1/CORE, 4 MB L2	DIRECTORY	CROSSBAR	TOTAL STORE ORDERING	60–123 W	900 MHZ–1.4 GHZ	16 OPS/CLOCK
INTEL ATOM [18], [5]	X86	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	ONE TO TWO	32 KB IL1 AND DL1/CORE, 512 KB L2/CORE	BROADCAST	BUS	PROCESSOR	2–8 W	800 MHZ–1.6 GHZ	2–16 OPS/CLOCK
ARM CORTEX-A9‡ [6]	ARM	THREE-WAY OUT-OF-ORDER	ONE TO FOUR	(16,32,64) KB IL1 AND DL1/CORE, UP TO 2 MB L2	BROADCAST	BUS	WEAKLY ORDERED	1 W (NO CACHE)	N/A	3–12 OPS/CLOCK
XMOS XS1-G4 [19]	XCORE	ONE-WAY IN-ORDER, EIGHT-WAY SMT	FOUR	64 KB LCL STORE/CORE	NONE	CROSSBAR	NONE	0.2 W	400 MHZ	4 OPS/CLOCK

‡Numbers are estimates because design is offered only as a customizable soft core.

[TABLE 4] TABLE OF HIGH-PERFORMANCE MULTICORES.

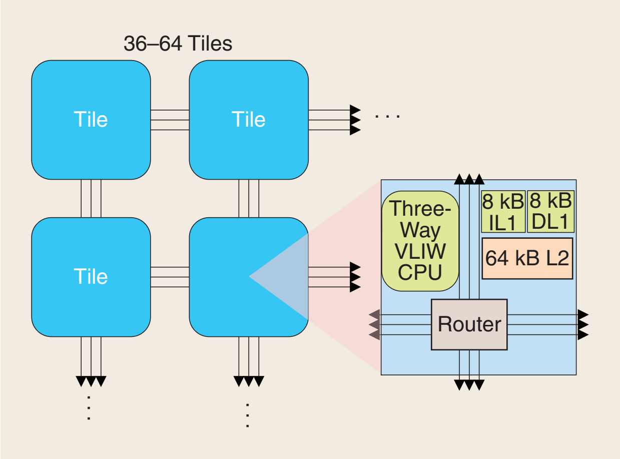
	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD RADEON R700 [20]	N/A	FIVE-WAY VLIW	160 CORES, 16 CORES PER SIMD BLOCK, TEN BLOCKS	16 KB LCL STORE/SIMD BLOCK	NONE	N/A	NONE	150 W	750 MHZ	800-1,600 OPS/CLOCK
NVIDIA G200 [8], [21]	N/A	ONE-WAY IN-ORDER	240, EIGHT CORES PER SIMD UNIT, 30 SIMD UNITS	16 KB LCL STORE/EIGHT CORES	NONE	N/A	NONE	183 W	1.2 GHZ	240-720 OPS/CLOCK
INTEL LARRABEE† [22]	X86	TWO-WAY IN-ORDER, 4-WAY SMT, 512-B SIMD	UP TO 48†	32 KB IL1 AND 32 KB DL1/CORE, 4 MB L2	BROADCAST	BIDIRECTIONAL RING	PROCESSOR	N/A	N/A	96-1,536 OPS/CLOCK
IBM CELL [9], [23]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT PPU, 2-WAY IN-ORDER 128-B SIMD SPU	1 PPU, EIGHT SPUs	PPU: 32 KB IL1 AND 32 KB DL1, 512 KB L2; SPU: 256 KB LCL STORE	NONE	BIDIRECTIONAL RING	WEAK (PPU), NONE (SPU)	100 W	3.2 GHZ	72 OPS/CLOCK
MICROSOFT XENON [10]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	THREE	32 KB IL1 AND 32 KB DL1/CORE, 1 MB L2	BROADCAST	CROSSBAR	WEAKLY ORDERED	60 W	3.2 GHZ	6-24 OPS/CLOCK

†All values are estimates as processor is not yet in production.

[TABLE 5] TABLE OF DSP AND EXOTIC MULTICORES.

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMBRIC AM2045 [24], [25]	N/A	ONE-WAY IN-ORDER SR, THREE-WAY IN-ORDER SRD	168 SR, 168 SRD	21 KB LCL STORE/EIGHT CORES	NONE	NoC	NONE	6–16 W	350 MHZ	672 OPS/CLOCK
ELEMENT CXI ECA–64 [26], [3]	N/A	ONE-WAY IN-ORDER, DATAFLOW CONNECTIONS TO 15 RECONFIGURABLE ALUs	FOUR CLUSTERS OF ONE CORE+ALUs	32 KB OF LCL STORE/CLUSTER	NONE	HIERARCHIAL NoC	NONE	1 W	200 MHZ	64 OPS/CLOCK (16-B)
TI TMS320-DM6467 [14]	ARM, C64X	ONE ARM9 ONE-WAY IN-ORDER, ONE C64X EIGHT-WAY VLIW	TWO	ARM9: 16 KB IL1, 8 KB DL1; C64X: 32 KB IL1 AND DL1, 128 KB L2	NONE	BUS	WEAKLY ORDERED	3–5 W	ARM: 297–364 MHZ, C64X: 594–729 MHZ	1–9 OPS/CLOCK
TI OMAP 4430 [12]	ARM, C64X	TWO ARM THREE-WAY OUT-OF-ORDER, ONE C64X EIGHT-WAY VLIW	THREE	N/A	BROADCAST AMONG ARM CORES	BUS	WEAKLY ORDERED	1 W	1GHZ	6–14O PS/CLOCK
TILERA TILE64 [27], [28]	N/A	THREE-WAY VLIW	36–64	8 KB IL1 AND DL1/CORE, 64 KB L2/CORE	DIRECTORY	NoC	N/A	15–22 W	500–866 MHZ	108–192 OPS/CLOCK
HIVEFLEX CSP2X00‡ [29]	N/A	TWO-WAY VLIW CONTROL CORE, FIVE-WAY VLIW COMPLEX CORE	TWO TO FIVE	2X CONFIGURABLE L1 FOR BASE CORE, LCL STORE FOR COMPLEX CORE	NONE	BUS	NONE	0.25 W	200 MHZ	2–22 OPS/CLOCK

‡Numbers are estimates because design is offered only as a customizable soft core.



[FIG4] Tilera TILE64 block diagram.

TILERA TILE64–DIGITAL SIGNAL PROCESSING

The Tilera TILE64 [13] is a DSP-focused processor that takes the concept of a multicore to a logical extreme. It uses up to 64 simple, three-way VLIW cores connected by an NoC interconnect that is fully coherent. Because the individual cores are very small and low powered, the chip needs to needs massive parallelism in the application to achieve reasonable performance. Many DSP programs can take advantage of the many threads this processor exposes to the programmer. A block diagram of the architecture is provided in Figure 4.

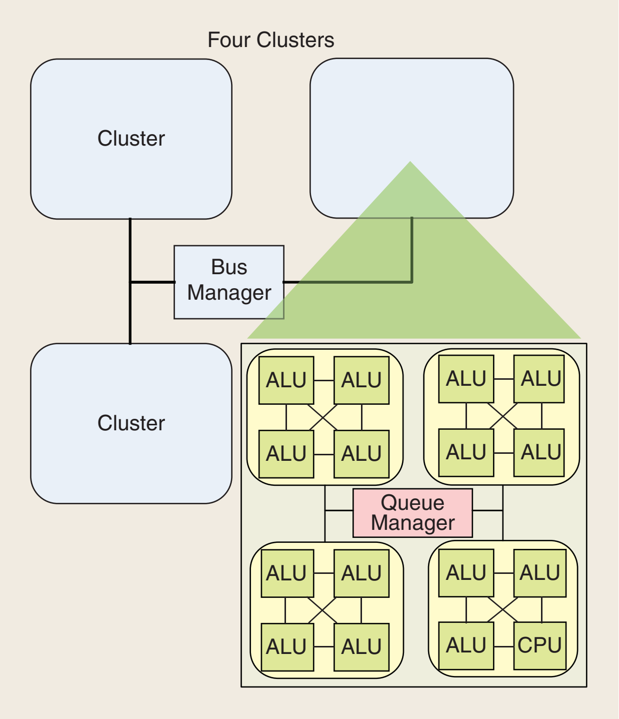
The fully coherent interconnect is not typical for a processor targeted towards DSP applications. However, it allows the processor to run more general-purpose shared memory programs. The interconnect is a large NoC, and because of the high number of cores it has a directory-based coherence policy to achieve scalable performance. Because of the extra general-purpose additions, such as a coherent interconnect and wider individual processing elements, its power consumption is, at 18 W, higher than most in its class.

ELEMENT CXI ECA-64–DIGITAL SIGNAL PROCESSING

The Element CXI ECA-64 [3] is a very low-power multicore targeted at DSP. It takes a very different design philosophy from any other multicore presented in this section by using a handful of control cores to manage a "sea" of ALUs. This is shown in the block diagram in Figure 5. This core is focused on data driven applications and very low power. The programming model is similar to programming a field-programmable gate array (FPGA).

The focus on low power is helped by a heterogeneous design. The processor itself is made up of four clusters of 16 processing elements. These clusters each include one RISC style processing core and 15 ALUs that are each specialized for different purposes. Also, each ALU is data driven, only performing operations when data is present at the input helping to keep power consumption low.

The memory subsystem follows design decisions made for low power. Each cluster shares 32 kB of local memory that is managed by software. The interconnect is hierarchial. It tightly couples four processing elements via a crossbar, and then four of these tightly coupled groups are connected using a point-to-point set of queues to form a cluster of 16 elements. The clusters then are able to communicate by a bus to each other. Although this is a low-power memory organization, the need for software control can make programming a challenge.

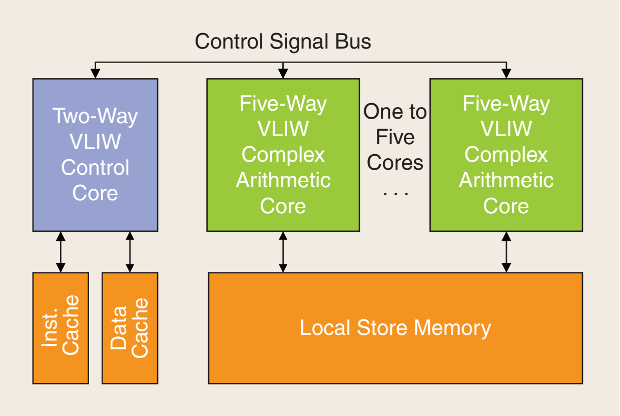


[FIG5] Element CXI ECA-64 block diagram.

SILICON HIVE HIVEFLEX CSP2X00 SERIES–DIGITAL SIGNAL PROCESSING

The HiveFlex CSP2x00 [29] series are soft cores offered by Silicon Hive. They are very low power, operating at around one quarter of a watt. A block diagram of the architecture is provided in Figure 6.

To achieve this low power, the series employs a heterogeneous collection of cores to attain the desired performance target. It has a control chip that is a general-purpose two-way VLIW design


Control Signal Bus
Two-Way VLIW Control Core
Five-Way VLIW Complex Arithmetic Core
One to Five Cores . . .
Five-Way VLIW Complex Arithmetic Core
Inst. Cache
Data Cache
Local Store Memory

[FIG6] Silicon Hive HiveFlex CSP2x00 block diagram.

with small standard caches used to run the main program. It then off-loads data intensive work to so-called "complex cores." The "complex cores" are five-way VLIW cores with customized ALUs for accelerating mathematical operations connected to a large local store RAM. The "complex cores" do not support branching, so they must be fed straight line code from the control core. The removal of branching support simplifies the "complex cores" to allow for savings in area and, more importantly, power.

In addition, the CSP2x00 has a very simple memory hierarchy. Coherence and consistency is controlled by software running on the control core. The bus interconnect of the CSP2x00 is primarily for transferring commands from the control core to the complex cores, which then communicate amongst themselves through the local store. All these design characteristics make for a very low-power yet high-performance chip. However, creating efficient software is a challenge.

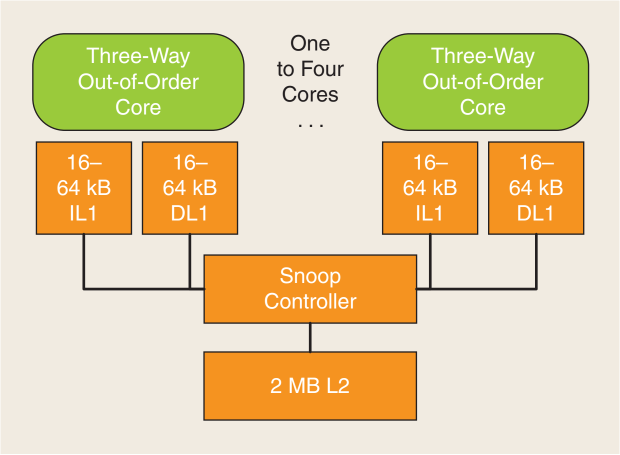
ARM CORTEX A9–GENERAL-PURPOSE MOBILE

The ARM Cortex A9 [6] is a general-purpose mobile embedded soft core that can be custom tailored before manufacturing; a block diagram is provided in Figure 7. The most common configurations are very low power—1 W or less. The design is targeted to general-purpose computing from smart phones to full featured netbooks. This entails a design that handles control dominated applications well. The individual processing cores on the A9 are three-way out-of-order that offer high general-purpose performance. The chip is targeted to run an OS and more traditional desktop style applications.

The interconnect used for the memory system is a fully coherent bus. The coherence is broadcast based since the number of cores this design uses is small. The caches are fairly large for a processor targeted at embedded applications. These are required to support the high clock speed and aggressive single thread design of the individual processing elements. More data dominated applications will not execute particularly efficient on this machine, because as noted, the chip is geared to traditional desktop style applications.

TI OMAP 4430–GENERAL-PURPOSE MOBILE SoC

The TI OMAP 4430 [12] is a general-purpose system-on-chip (SoC) targeted to future smart phones and mobile-Internet-devices (MIDs). A block diagram is provided in Figure 8. The design is also very low power, reported to be about 1 W, with significant processing capabilities and a large number of peripherals. It uses two ARM Cortex A9 processors for general-purpose applications and a C64x DSP to be used for emerging data-dominated media applications. To process most media and graphics, it has three fixed function ASICs to accelerate performance at very low power: a GPU, image processor, and audio/visual codec processor. It also has many peripherals and other accelerators like encryption on chip. This chip is predominantly a collection of ASICs that are controlled by the general-purpose processors. This is done to save as much power as possible. In cases where the ASICs can not be employed, like running a new media codec, it has ample processing capabilities from the three general-purpose cores but uses more power.



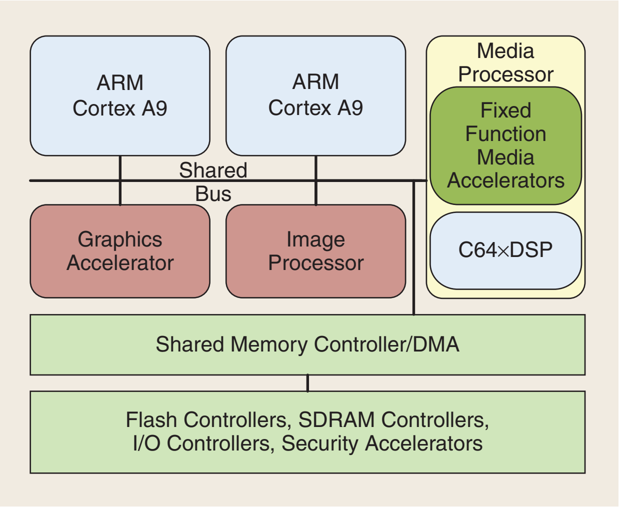
[FIG7] ARM Cortex-A9 block diagram.

The interconnect used for the memory system is a fully coherent bus between the ARM cores for general-purpose shared memory programming. The bus between the accelerators and C64x is noncoherent, requiring the ARM cores to explicitly manage data movement to and from the accelerators and DSP. The memory controller is shared, making the point of coherence for the entire system at the main memory level.

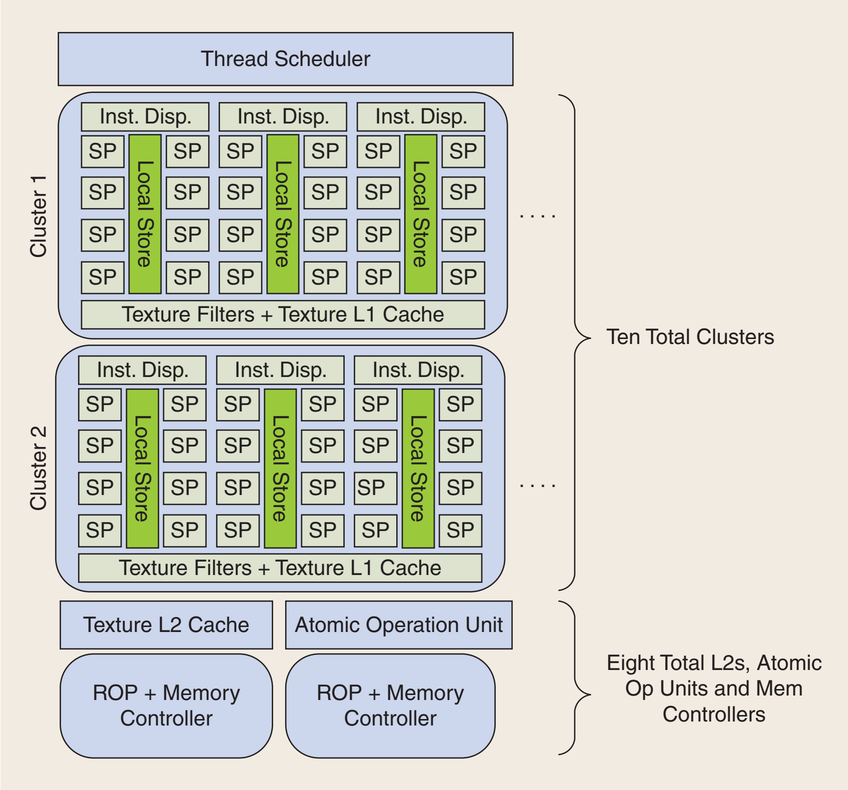
NVIDIA G200–GRAPHICS/ HIGH-PERFORMANCE COMPUTE

The NVIDIA G200 [21] is a high-performance architecture specifically aimed at data dominated applications, particularly raster graphics. However, it is also able to provide more general programmability to support nongraphics related, data dependant applications.

The architecture itself contains 240 simple one-way in-order cores. Each core is grouped together with 24 other cores in a cluster. Every group of eight cores share a 16 kB local store memory. The 24 cores are controlled in a SIMD manner: each


ARM Cortex A9
ARM Cortex A9
Media Processor
Fixed Function Media Accelerators
Shared Bus
Graphics Accelerator
Image Processor
C64×DSP
Shared Memory Controller/DMA
Flash Controllers, SDRAM Controllers, I/O Controllers, Security Accelerators

[FIG8] TI OMAP 4430 block diagram.

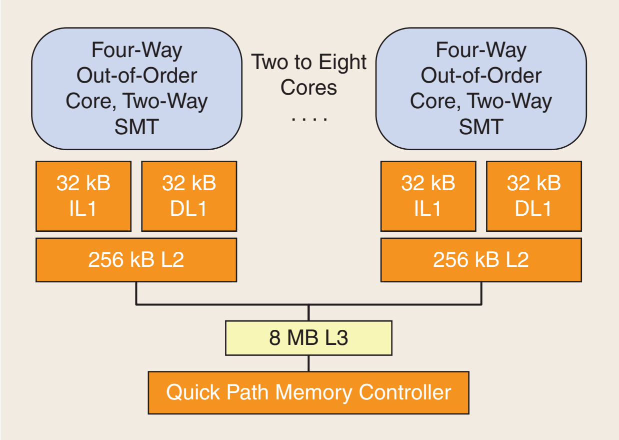


[FIG9] NVIDIA G200 architecture block diagram.

core executes the same instruction from different "threads." Unlike an SIMD processor, each core is capable of branching, but if this happens, all the other cores must execute both paths of the branch and only keep the path they would have followed. The G200 cores can also access memory in a non-SIMD like fashion, e.g., if core one accesses address x, core two can access address y rather than only $x + 1$ as would be the case in a traditional SIMD machine. Though accessing memory in this fashion imparts a performance penalty as the memory controller cannot, in general, coalesce memory accesses. This makes the G200 more general than a traditional SIMD machine and is closer to a multiple instruction multiple data (MIMD) machine. But because the architecture incurs penalties when the instruction streams and memory accesses for processors in a group diverge, it sits in between traditional MIMD machines from manufacturers like Intel and its previous fixed function GPU predecessors.

The design of the memory system is also tuned to data dominated applications. The memory system is noncoherent and uses small local stores instead of a standard cache style architecture (caches do exist, but they are used as texture caches for raster graphics and not general-purpose computing). The G200 has relatively little on die memory, instead favoring compute resources. It is able to accommodate this by using very low latency high speed RAM, since power is not a factor in this design. Even though the memories are noncoherent, the G200 does provide some facility for more general parallel programs by providing "atomic operation" units as seen in Figure 9. These are used for controlling access to shared data structures that live in the GPU's main memory.

As noted, this architecture is well suited for applications that are highly data dominated, for example, medical imaging, and financial data processing. It however, is not very well suited for control-dominated applications because branches and random memory accesses incur stiff performance penalties. The G200 is a unique architecture that is almost a general-purpose MIMD but to maximize compute density was designed with certain restrictions and specializations to accomplish its primary task, which is graphics processing.


Four-Way Out-of-Order Core, Two-Way SMT
Two to Eight Cores
. . . .
Four-Way Out-of-Order Core, Two-Way SMT
32 kB IL1
32 kB DL1
256 kB L2
32 kB IL1
32 kB DL1
256 kB L2
8 MB L3
Quick Path Memory Controller

[FIG10] Intel Core i7 block diagram.

INTEL CORE I7–GENERAL PURPOSE

The Intel Core i7 [2] is a high-performance general-purpose processor in all respects. It attempts to do everything well. This comes at the cost of a high (140 W) maximum power dissipation.

It is implemented with up to eight four-issue out-of-order, two-way symmetric multithreading (SMT) cores, as seen in Figure 10. These cores contain many complex enhancements to extract as much performance out of a single thread as possible. Each core also contains a 128-b SIMD unit to take advantage of some data parallelism. In keeping with most Intel processors, it supports the CISC x86 ISA. This design allows it to do many things well, but lower power more specialized designs can compete favorably in particular application domains.

The memory system is typical of that found in a general-purpose multicore machine with just a few cores. It uses a fully coherent memory system and has large standard caches. The coherence is broadcast based, which is sufficient because of the limited number of cores. These characteristics come together to create a chip that is good at a wide variety of applications provided power is not a constraint.

CONCLUSION

With the emergence of commercial multicore architectures in an array of application domains, it is important to understand the major design characteristics common among all multicores. In this article, we defined five major attributes common among multicore architectures and discussed the tradeoffs for each attribute in the context of actual commercial products. These areas were application domain, power/performance, processing elements, memory, and accelerators/integrated peripherals. We then covered in greater detail several commercial examples of multicore chips in a variety of application areas. We illustrated how attributes such as DSP, general-purpose mobile, and high-performance general purpose directed these example architectures to very unique designs.

With transistor budgets still increasing every few years and the desire for more performance still apparent, multicore architectures will continue to be produced. As more applications are developed that can take advantage of multicore, the designs will continue to evolve to offer the desired balance of programmability and specialization.

AUTHORS

Geoffrey Blake (blakeg@umich.edu) received his B.S.E. degree in computer engineering and his M.S.E. degree in computer science and engineering from the University of Michigan, where he is also a Ph.D. candidate. His research interests are in multicore architecture, operating systems, and transactional memories.

Ronald G. Dreslinski (rdreslin@umich.edu) is a Ph.D candidate at the University of Michigan. He received the B.S.E. degree in electrical engineering, B.S.E. degree in computer engineering, and M.S.E. degree in computer science and engineering, all from the University of Michigan. He is a Member of the IEEE and the ACM . His research focuses on architectures that enable emerging low-power circuit techniques.

Trevor Mudge (tnm@umich.edu) received the B.Sc. degree from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1973 and 1977, respectively. He has been on the faculty of the University of Michigan, Ann Arbor since 1977. He was named the first Bredt Family Professor of Electrical Engineering and Computer Science after concluding a ten-year term as the director of the Advanced Computer Architecture Laboratory. He authored numerous papers on computer architecture, programming languages, VLSI design, and computer vision. His research interests include computer architecture, computer-aided design, and compilers. He runs Idiot Savants, a chip-design consultancy. He is a Fellow of the IEEE and a member of the ACM, the IET, and the British Computer Society.

REFERENCES

[1] International Technology Roadmap for Semiconductors, "International technology roadmap for semiconductors—System drivers," 2007 [Online]. Available: http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_System-Drivers.pdf

[2] Intel Corp., "Intel core i7-940 processor," Intel Product Information, 2009 [Online]. Available: http://ark.intel.com/cpu.aspx?groupId=37148

[3] Element CXI Inc., "ECA-64 elemental computing array," Element CXI Product Brief, 2008 [Online]. Available: http://www.elementcxi.com/downloads/ECA64ProductBrief.doc

[4] ITU-T, "H.264.1: Conformance specification for h.264 advanced video coding," Tech. Rep., June 2008.

[5] "Intel 64 and IA-32 Architectures Software Developer's Manual," *Intel Developer Manuals*, vol. 3A, Nov. 2008.

[6] ARM Ltd., "The ARM Cortex-A9 Processors," ARM Ltd. White Paper, Sept. 2007 [Online]. Available: http://www.arm.com/pdfs/ARMCortexA-9Processors.pdf

[7] Tensilica Inc., "Configurable processors: What, why, how?" Tensilica Xtensa LX2 White Papers, 2009 [Online]. Available: http://www.tensilica.com/products/literature-docs/white-papers/configurable-processors.htm

[8] A. L. Shimpi and D. Wilson, "NVIDIA's 1.4 billion transistor GPU: GT200 arrives as the GeForce GTX 280 & 260," Anandtech Web site, 2008 [Online]. Available: http://www.anandtech.com/video/showdoc.aspx?i=3334

[9] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, 2006.

[10] J. Andrews and N. Baker, "Xbox 360 system architecture," *IEEE Micro*, vol. 26, no. 2, pp. 25–37, 2006.

[11] Advanced Micro Devices Inc. "Key architectural features—AMD Phenom II processors," AMD Product Information, 2008 [Online]. Available: http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15917%5E15919,00.html

[12] Texas Instruments Inc., "OMAP 4: Mobile applications platform," 2009 [Online]. Available: http://focus.ti.com/lit/ml/swpt034/swpt034.pdf

[13] Tilera Corp., "Tilepro64 processor," Tilera Product Brief, 2008 [Online]. Available: http://www.tilera.com/pdf/ProductBrief_TILEPro64_Web_v2.pdf

[14] Texas Instruments, Inc., "TMS320DM6467: Digital media system-on-chip," Texas Instruments Datasheet, 2008 [Online]. Available: http://focus.ti.com/lit/ds/symlink/tms320dm6467.pdf

[15] Advanced Micro Devices Inc., "Software optimization guide for AMD family 10h processors," AMD White Papers and Technical Documents, Nov. 2008 [Online]. Available: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/40546.pdf

[16] Sun Microsystems Inc., "UltraSPARC T2 processor," Sun Microsystems Data Sheet*s*, 2007 [Online]. Available: http://www.sun.com/processors/UltraSPARC-T2/datasheet.pdf

[17] T. Johnson and U. Nawathe, "An 8-core, 64-thread, 64-bit power efficient sparc soc (niagara2)," in *Proc. 2007 Int. Symp. Physical Design ISPD '07.* New York, NY: ACM, 2007, pp. 2–2.

[18] Intel Corp., "Intel atom processor for nettop platforms," Intel Product Brief, 2008 [Online]. Available: http://download.intel.com/products/atom/319995.pdf

[19] D. May, "XMOS XS1 architecture," XMOS Ltd., July 2008 [Online]. Available: http://www.xmos.com/files/xs1-87.pdf

[20] Advanced Micro Devices Inc., "ATI Radeon HD 4850 & ATI Radeon HD 4870—GPU specifications," AMD Product Information, 2008 [Online]. Available: http://ati.amd.com/products/radeonhd4800/specs3.html

[21] NVIDIA Corp., "NVIDIA CUDA: Compute unified device architecture," NVidia CUDA Documentation, June 2008 [Online]. Available: http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

[22] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: A many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–15, Aug. 2008.

[23] IBM Corp., "POWER ISA Version 2.05," Power.org documentation, Oct. 2007 [Online]. Available: http://www.power.org/resources/reading/PowerISA_V2.05.pdf

[24] T. R. Halfhill, "Ambric's new parallel processor: Globally asynchronous architecture eases parallel programming," *Microprocessor Rep.*, Oct. 2006 [Online]. Available: http://:/www.mdronline.com/mpr/h/2006/1010/204101.html

[25] Nethra Imaging Inc., "Massively parallel proccessing arrays technology overview," Ambric Technology Overview, 2008 [Online]. Available: http://www.ambric.com/technologies_mppa.php

[26] S. Kelem, B. Box, S. Wasson, R. Plunkett, J. Hassoun, and C. Phillips, "An elemental computing architecture for SD radio," in *Proc. Software Defined Radio Technical Conf. Product Exposition*, 2007.

[27] M. Baron, "Tilera's cores communicate better: Mesh networks and distributed memory reduce contention among cores," *Microprocessor Rep.*, Nov. 2007 [Online]. Available: http://www.mdronline.com/mpr/h/2007/1105/214501.html

[28] A. Agarwal, B. Liewei, J. Brown, B. Edwards, M. Mattina, C.-C. Miao, C. Ramey, and D. Wentzlaff, "Tile processor: Embedded multicore for networking and multimedia," in *Proc. Hotchips 19: A Symp. High Performance Chips*, 2007.

[29] "HiveFlex CSP2000 series: Programmable OFDM communication signal processor," *Silcon Hive Databrief*, 2007 [Online]. Available: http://www.siliconhive.com/Flex/Site/Page.aspx?PageID=8881 [SP]