Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

INTEL'S LARRABEE REDEFINES GPUS

Fully Programmable Manycore Processor Reaches Beyond Graphics

By Tom R. Halfhill {9/29/08-01}

Intel is spreading the x86 everywhere. No longer satisfied with existing strongholds in PCs and servers, this year Intel has revived the x86 as a standalone embedded processor and has introduced the first highly integrated x86-based SoCs. And as early as next year, Intel will debut the first x86-based 3D-graphics processors.

So far, graphics is the oddest addition to the x86's growing list of target applications. A 30-year-old CISC architecture designed for general-purpose processing would seem to be seriously handicapped against special-purpose GPUs, which are highly optimized for tasks like pixel shading and texture mapping. But Intel is undeterred. At Siggraph 2008—a graphics show, not a microprocessor conference—Intel unveiled the first technical details about its future x86-based GPU, code-named Larrabee.

Among other things, Intel revealed that Larrabee is a scalable microarchitecture supporting many possible implementations, with as few as four cores per chip or as many as dozens. These cores will have coherent caches and will communicate over a ring network on chip. Architectural improvements include 16-lane SIMD extensions and four threads per core—parallel-processing features not found in any other x86 chip. Moreover, Larrabee has fully programmable graphics pipelines and much less hard-wired acceleration logic than other GPUs.

Scheduled to debut in 2009 or 2010, Larrabee is a direct challenge to the long-established GPUs from ATI (acquired by AMD in 2006) and Nvidia. Those two vendors control 98% of the discrete-GPU market, according to Jon Peddie Research. Intel's first Larrabee devices will be discrete GPUs on expansion boards for PCs, followed by integrated graphics in system chipsets and CPUs. Last year, the total market for GPUs (discrete and integrated) was 350 million units, according to Peddie.

Intel's Strategy: More Than Graphics

What's next—x86 DSPs? Actually, Intel already tried something like that in the mid-1990s with its "native signal-processing" initiative. Although Intel didn't make traditional DSPs obsolete, today it's routine for general-purpose CPUs to handle signal-processing tasks once thought suitable only for discrete DSPs. (See *MPR 5/8/95-03*, "NSP Shows Promise on Pentium, PowerPC.")

However, there's a crucial difference between Intel's latest x86-everywhere strategy and its signal-processing initiative of the 1990s. Back then, Intel wanted to move specialized processing into the host x86 CPU, not to a specialized x86 DSP. This time, with Larrabee, Intel hopes to replace existing GPUs with discrete chips of its own—or with integrated graphics that are almost as good as discrete chips. In other words, Larrabee is an aggressive frontal attack on the entrenched specialized processors, not a flanking maneuver intended to sell more CPUs.

Some observers compare Larrabee with Intel's ill-fated i740 graphics accelerator, introduced in 1998. However, the i740 had a specialized graphics architecture, not an x86 architecture, and it was quickly swamped by superior GPUs from ATI, Nvidia, and others. (See *MPR 1/25/99-03*, "Intel Stakes 3D Claim, But ATI Takes Lead.")

Intel's choice of the x86 for a comeback in the graphics market speaks volumes. Although the ancient x86 architecture isn't ideally suited for high-end graphics processing, Larrabee isn't just a graphics processor. It also aims at high-performance

computing (HPC), or what Intel variously calls "high-throughput computing" or "visual computing." HPC applications include financial modeling, pharmaceutical development, energy exploration, cryptography, weather forecasting, climate modeling, and scientific research of all kinds.

Larrabee Foreshadows Intel's Future

HPC is a fast-growing field with an insatiable appetite for processing power. It's particularly well suited for highly parallel processors like Larrabee. That's why ATI and Nvidia have been pushing their GPUs in the same direction—a trend known as "general-purpose GPU" (GPGPU). In recent years, ATI and Nvidia GPUs have become more programmable and have adopted other features desirable for HPC applications, such as double-precision FPUs. But x86 processors have always been fully programmable and have had double-precision FPUs for almost 20 years. Intel hopes to convince HPC programmers that an x86-based GPU can leverage existing x86 software-development tools and source code and will have a more familiar programming model.

Another rationale for basing Larrabee on the x86 architecture is that future PCs will be able to use their Larrabee GPUs as coprocessors for compute-intensive tasks other than graphics. Video transcoding is a good example of a highly parallel task suited for Larrabee. Although existing GPUs can do the same thing—indeed, they're doing it now, in small ways—having a GPU that shares the same architecture with the CPU could ease software development and task sharing.

In addition, Larrabee helps Intel gain experience designing manycore processors. For years, *Microprocessor Report* has covered specialized processors with dozens, hundreds, or even thousands of cores per chip. Meanwhile, Intel—the world's leading microprocessor company—is only now readying its first eight-core chip. (It's a server processor based on the Nehalem microarchitecture.)

Although *MPR* doesn't doubt that Intel has the expertise to create larger multicore designs, reducing theory to practice on a commercial scale is always revelatory. With Larrabee, Intel is leaping toward large-scale manycore designs that will have dozens of cores per chip. That experience—and the parallel software soon to be written for Larrabee—will be valuable down the road, when Intel begins designing manycore processors for PCs and servers.

Unprecedented x86 Diversity

Larrabee is only part of Intel's x86-everywhere strategy. Earlier this year, Intel's Atom microprocessor marked the return of discrete x86 processors expressly designed for embedded systems, although Atom also reaches into subnotebooks and low-end desktop PCs. (See *MPR 4/7/08-01*, "Intel's Tiny Atom.") And in July, Intel introduced the first highly integrated x86-based SoCs for networking and communications, soon followed by similar SoCs for consumer electronics. (See *MPR 8/18/08-01*, "Intel's New SoCs.")

Suddenly, there's unexpected diversity among Intel's x86 chips. Intel has simultaneously designed three new x86 microarchitectures: Atom, Nehalem (officially and confusingly dubbed the Intel Core Architecture), and the Larrabee core. Each core represents a significant design effort, conducted at disparate locations by different design teams. Other than baseline x86 compatibility, these cores have little in common. Indeed, there are now more differences among Intel's own x86 microarchitectures than there are between the PC processors from Intel and AMD.

Intel's new processor cores are so different from each other that they threaten to split the x86 architecture into market-specific fragments. This unintended consequence, if permitted to continue, could undermine the reasons for basing the designs on the x86 in the first place—software compatibility and easier software development. Intel hopes to unite its diverging microarchitectures in the future, but some features created for a graphics processor might never make sense in a server processor, or vice versa. The last thing the x86 needs is more architectural baggage to haul around. Already, it lugs outdated instructions and features that seemed modern when Intel engineers spent all of three weeks defining the x86 instruction set in 1978.

Nevertheless, diverse x86 microarchitectures may be unavoidable. The application spectrum that Intel is now pursuing looks too broad for a one-size-fits-all approach. Not long ago, CPU architects hoped that multicore processing might eliminate the need to design different cores for different markets. In theory, one basic core could serve all purposes, because designers could simply scale the number of cores per chip up or down, as the application demanded. Although that concept may still be valid within a particular domain, such as PCs, Intel is spreading the x86 so far and wide that significantly different microarchitectures may be inevitable.

Leveraging the x86 Architecture

According to Larrabee's architects, upper management didn't absolutely mandate x86 compatibility for the new processor. In fact, one experimental design crammed 80 single-precision FPUs on a single chip, discarding the rest of the x86 CPU. But that prototype, code-named Polaris, or the Tera-FLOPS Research Processor, has little in common with the Larrabee design now emerging from the lab. Even the on-chip network—a critical part of any large-scale multicore design—was different. (See *MPR 4/9/07-01*, "Low-Key Intel 80-Core Intro: The Tip of the Iceberg.")

In retrospect, it's hard to imagine choosing any architecture but the x86 for Larrabee, given Intel's broader ambitions for the processor and the painful history of the i740. *MPR* questioned the Larrabee architects closely on this point. They defended their choice of the x86 on firm engineering grounds, not just because it aligns with corporate strategy emanating from Santa Clara.

From the start, the Larrabee design team wanted to make a GPU with a fully programmable graphics pipeline.

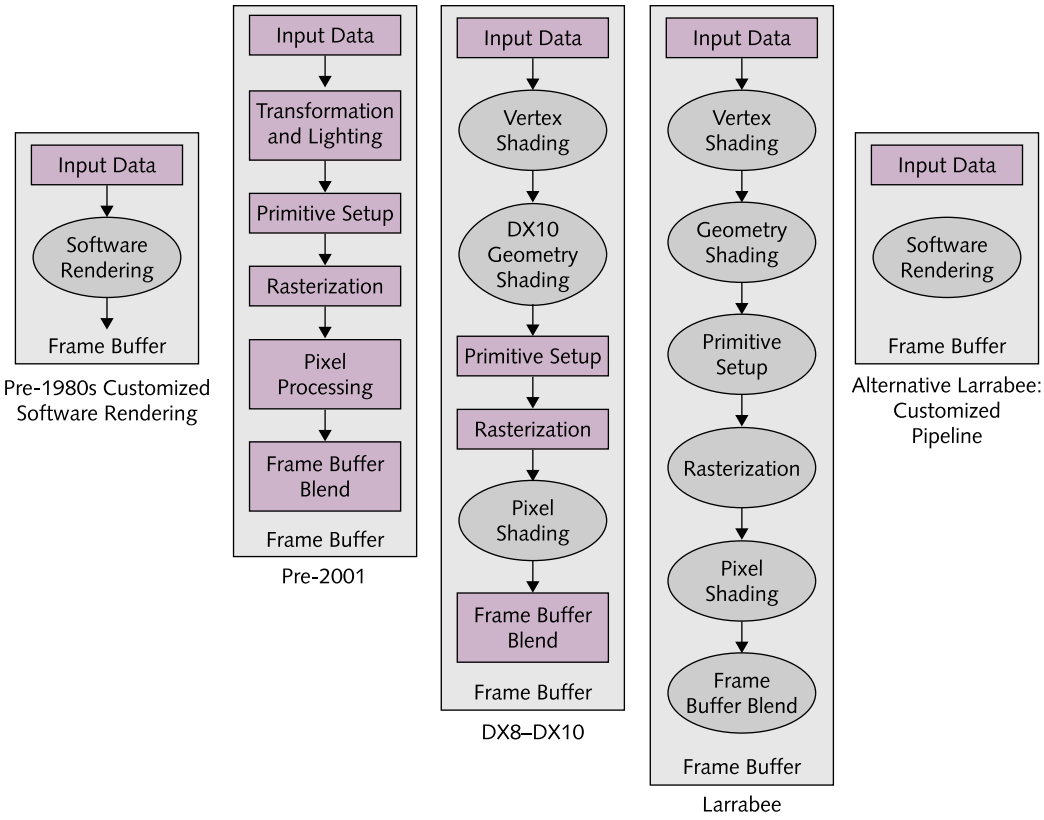

Figure 1. Larrabee has a fully programmable graphics pipeline, augmented with only a little specialized logic. Recent GPUs from ATI and Nvidia have programmable pipelines, too, but they rely more heavily on fixed-function logic to deliver high performance. Programmability has become more important to ATI and Nvidia as they pursue markets other than graphics. Interestingly, Larrabee also supports an alternative software-rendering model similar to the one that prevailed before graphics accelerators arrived in the 1980s—except that Larrabee has vastly faster 21st-century logic.

In addition, the designers wanted to leverage their expertise with the x86 and their existing software and development tools. By adapting the x86 for Larrabee, Intel doesn't have to rewrite its assemblers, compilers, profilers, debuggers, simulators, and drivers completely from scratch. Larrabee supports the popular DirectX and OpenGL graphics APIs, and it can run other existing middleware, after some modifications.

Programmability is a vital aspect of Larrabee. Since their inception in the 1980s as hard-wired graphics accelerators, conventional GPUs have gradually become more programmable, but they still have vital differences from general-purpose processors. For instance, their memory models are very different, lacking such concepts as virtual memory, protected memory, and coherent caches. GPUs are heavily threaded processors, but they can't do context switching in the same sense that CPUs do. Their ability to call procedures is more limited. And only the latest GPUs from ATI and Nvidia can natively manipulate double-precision floating-point numbers, a critical requirement for some HPC applications. Figure 1 shows Intel's view of Larrabee's place in the evolution of GPUs.

Larrabee brings the full flexibility of a general-purpose processor to 3D graphics. Of course, flexibility is also a handicap, because specialized logic usually outperforms general-purpose logic. Frankly, *MPR* will be surprised if Larrabee doesn't trail the best GPUs from ATI and Nvidia when independent benchmark testers get hold of it. However, we doubt that superior graphics performance will be critical to its long-term success. The market for discrete GPUs is relatively flat and dominated by avid gamers. We expect Larrabee to be more important for the HPC market and integrated graphics. Scaled-down versions of Larrabee, integrated in the north-bridge chip or CPU, will likely satisfy the majority of PC users.

Dr. Frankenstein Robs a Grave

Having settled on the x86 architecture, the Larrabee architects did something surprising. They didn't design an entirely new x86 core or adapt the low-power Atom core (which wasn't finished yet). Instead, pressed for time and worried about power consumption, they unearthed the brain of their new processor from Intel's graveyard. They derived the Larrabee core microarchitecture from the RTL of the original Pentium, introduced in 1993.

It's not even the Pentium Pro, Pentium II, or Pentium III—just the plain old Pentium. And remember, the Pentium

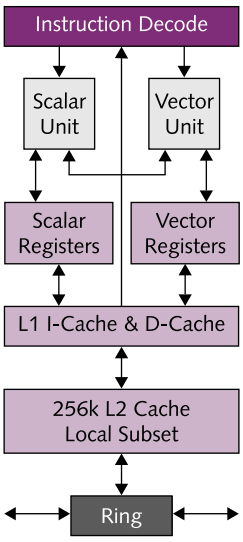

Figure 2. Block diagram of Larrabee's scalar and vector instruction paths. The vector unit has its own register files, separate from the standard set of x86 integer and floating-point registers. (Some earlier MMX and SSE instructions share the floating-point registers.) Both datapaths share the same L1 instruction and data caches, plus a local subset of the L2 cache.

was a close cousin of the even older 486. Both processors had five-stage pipelines and in-order instruction execution. Essentially, the Pentium merely doubled the 486 pipeline to create Intel's first superscalar x86.

Fifteen years later, Larrabee has almost identical integer pipelines. It doesn't spend power on instruction reordering, speculative execution, or translating CISC instructions into RISC-like micro-ops. Indeed, Intel has resurrected the original Pentium so faithfully that Larrabee has none of the MMX instructions and Streaming SIMD Extensions (SSE) added to later versions of the Pentium and numerous descendants. (See *MPR 4/30/07-01*, "Intel Goes On the Offensive.")

Like Dr. Frankenstein, however, Intel has upgraded the old brain with lots of new parts. Longtime friends may not recognize the finished creation. Larrabee has the latest 64-bit x86 extensions, new 16-lane SIMD extensions, additional scalar instructions, the ability to execute four threads per core, and other improvements. In those respects, Larrabee is a superset of the Pentium and the most modern x86 design yet seen, even surpassing Atom. Yet on a fundamental level, it's a throwback to 1993 and the good old days of dollar-a-gallon gasoline.

Larrabee's 64-bit extensions are the same as those found in other recent Intel x86 processors. (See *MPR 3/29/04-01*, "AMD and Intel Harmonize on 64.") But four threads per core is a big improvement. Until now, Hyper-Threading (Intel's brand name for chip multithreading) was limited to two threads per core. Each thread has its own complete register file, program counter, and status registers for preserving contextual information. A hardware pointer chooses which context is currently active, allowing the processor to switch threads on every clock cycle, if necessary. Quadruple threading in Larrabee should reduce stalls, because the processor can instantly switch to another context that's ready to go.

With up to four threads per core, a Larrabee chip can appear to have four times as many virtual cores as real cores—and the number of real cores could reach dozens, in high-end devices. In that sense, Larrabee could break the vague manycore boundary to become Intel's first massively parallel processor. However, even virtual cores aren't free. The die cost in additional registers and wiring is particularly significant for Larrabee, because other new features add more context to preserve.

The Widest SIMD Units in Any x86

Among those new features are new SIMD instructions that can perform an operation on as many as 16 operands at once. Those 16 operands can be as wide as 32 bits, so the SIMD datapaths are 512 bits wide—two to four times wider than those in other x86 processors. Up to eight 64-bit double-precision floating-point operations can execute at once.

Ironically, Larrabee lacks the older MMX and SSE instructions in other x86 processors that can operate on as many as eight operands at once. The latest version of SSE is 4.2, which adds seven new instructions to SSE 4.1. SSE 4.1 has 47 new instructions and appears in the current PC and server processors based on the Penryn microarchitecture. SSE 4.2 will debut in the Nehalem microarchitecture later this year.

(Confusingly, Intel now refers to the Nehalem microarchitecture as the Intel Core Architecture, even though it's the third iteration in this naming scheme. The original Banias-derived microarchitecture was also called the Intel Core Architecture. For a time, the second version—Penryn—was called the Intel Core 2 Architecture. Nehalem is really Core 3.)

Intel hasn't yet concocted a confusing brand name for the newer 16-lane SIMD extensions in Larrabee. For now, they're simply called "Larrabee new instructions." Nor has Intel publicly disclosed the extensions in detail. Intel says Larrabee has 100 to 150 new instructions, mostly for vector operations, but also 10 to 15 new scalar instructions. By any measure, it's a significant expansion of the x86 instruction set. Although Intel consulted game developers and graphics programmers when developing Larrabee, relatively few of the new instructions are specific to graphics. Figure 2 illustrates the way vector instructions integrate with the existing scalar instruction pipeline.

The vector instructions are the most interesting additions to Larrabee. They can manipulate integer and floating-point data types up to 64 bits long. Some vector instructions—such as a fused multiply-add—can manipulate three source operands, instead of the two sources commonly handled by other instructions. (One source is also the destination.) And most vector instructions can fetch one of their source operands from the L1 cache instead of a register.

Fetching operands from the L1 cache works in concert with new cache-prefetch instructions. The Pentium already

had some prefetch instructions, but Larrabee improves on them. If a vector instruction references a prefetched operand, the cache effectively works like an extended register file—without the extra latches required by real registers. Not only that, but the processor can automatically expand 8- and 16-bit operands fetched from the cache into 32-bit integer or 32-bit floating-point values, with no performance penalty. Other new instructions can explicitly evict data from the cache. All together, these instructions give programmers finer control over cache operations, if they wish to exercise it.

FMA Instruction Boosts Throughput

The fused multiply-add instruction ($A \times B + C$) simultaneously performs two arithmetic operations on as many as 16 operands in the SIMD lanes. In other words, a single instruction performs a total of 32 operations per clock cycle. Therefore, at 1.0GHz, the maximum theoretical throughput of each Larrabee core is 32 gigaflops (billion floating-point operations per second). At 1.0GHz, a Larrabee chip would need 32 cores to reach one teraflops (trillion floating-point operations per second). Alternatively, fewer cores running at faster clock speeds could achieve the same performance.

Intel refers to the new SIMD engine as a vector processing unit (VPU). It has numerous instructions for parallel operations. It can shift and swizzle the contents of registers in several ways to realign data for efficient processing. (Swizzling is the ability to copy multiple operands from source registers to other registers in different arrangements.) A single instruction can fetch a data element from memory and broadcast it across multiple lanes in the vector registers, eliminating the need to execute multiple load instructions to fetch the same data. Figure 3 is a block diagram of a VPU.

Scatter-gather instructions, indexed by a vector register, can load or store data at 16 noncontiguous memory addresses. This capability is particularly valuable for graphics but is useful for other datatypes as well. Intel says the scatter-gather instructions allow the VPU to run 16 instances of a pixel-shader routine simultaneously, gathering data from many disparate memory locations.

The 16-lane VPU is no accident. Intel arrived at the design by simulating various configurations of the SIMD engines. For example, shader simulations indicated that it's more efficient for the VPU to process the red, green, and blue components of a pixel separately than simultaneously. In other words, the VPU can operate on 16 pixels at once in the 16 lanes of the SIMD engines by processing the RGB components serially, instead of using 15 lanes to process the RGB components of five pixels in parallel. Either way, it's parallel processing, but Larrabee's VPU appears to be more efficient when decomposing the data by color components instead of by pixels.

Not a Conventional GPU

By itself, Larrabee's approach to pixel shading isn't innovative. Nvidia's GeForce 8 processor handles pixel shading in a similar way, even outdoing Larrabee in one respect by quad-pumping each of its eight-lane SIMD engines to perform 32 operations in parallel. However, the Nvidia GPU manages much of this process in fixed-function hardware, not in software. In Larrabee, a control thread—running in parallel with the shader threads—manages the program loops and cache movements. The whole process is under program control.

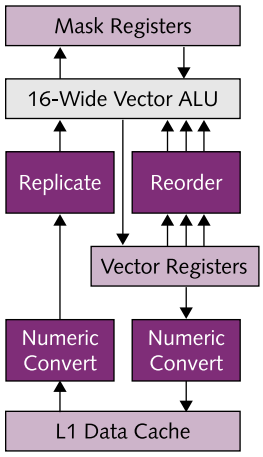

Figure 3. Block diagram of Larrabee's vector processing unit (VPU). Note the special hardware for automatically converting the datatypes of operands, for replicating loaded data across multiple lanes of the SIMD engine, and for reordering data. The mask registers have one predicate bit per vector lane, allowing the VPU to selectively store results in registers or memory. Predication eliminates the need for some conditional branches and gives the compiler more freedom when scheduling instructions in heavily branched code. More important, the mask registers support scatter-gather memory operations and the same kind of data-parallel programming used with other GPUs.

Programmability is the big difference between Larrabee's x86-based graphics pipeline and a conventional GPU pipeline. Although recent GPUs from ATI and Nvidia have programmable pipelines, they rely on hardware acceleration to a greater extent than Larrabee does. Larrabee does use fixed-function logic for texture filtering, but not for rasterization, post-shader blending, and other graphics functions that are still hard-wired in conventional GPUs.

As a less specialized processor, Larrabee has more in common with IBM's Cell Broadband Engine than it does with GPUs from ATI and Nvidia. Cell processors are found in everything from the Sony PlayStation 3 to professional workstations, servers, and supercomputers, where they perform a wide variety of tasks. Intel has similar ambitions for Larrabee.

But Larrabee differs from Cell, too, in important ways. Instead of using a single processor core to control multiple SIMD engines, as Cell does, each of Larrabee's x86 cores is capable of acting as a SIMD engine and as a control processor. Indeed, Larrabee's cores can perform both functions at the same time, in different threads. In a Cell processor, the lone Power Architecture core has a more dominant master/slave

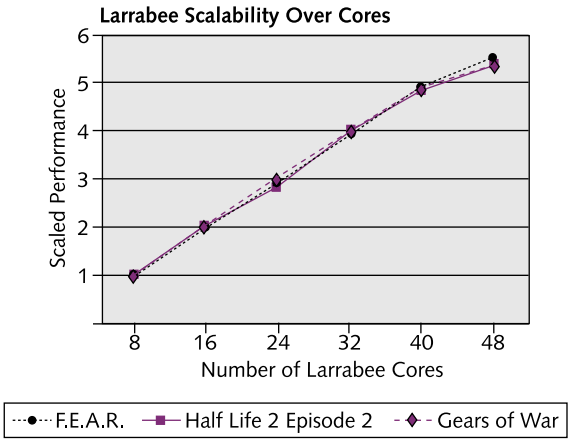

Figure 4. Graphics performance of three action games on simulated Larrabee processors with eight to 48 cores. As the number of cores rises, performance scales on a very linear slope—testimony to the efficiency of the processor and the inherent parallelism of the software. Intel provided this performance data, which is rather vague and hasn't been verified with production silicon. However, this degree of scaling is believable when running data-parallel tasks on highly parallel processors.

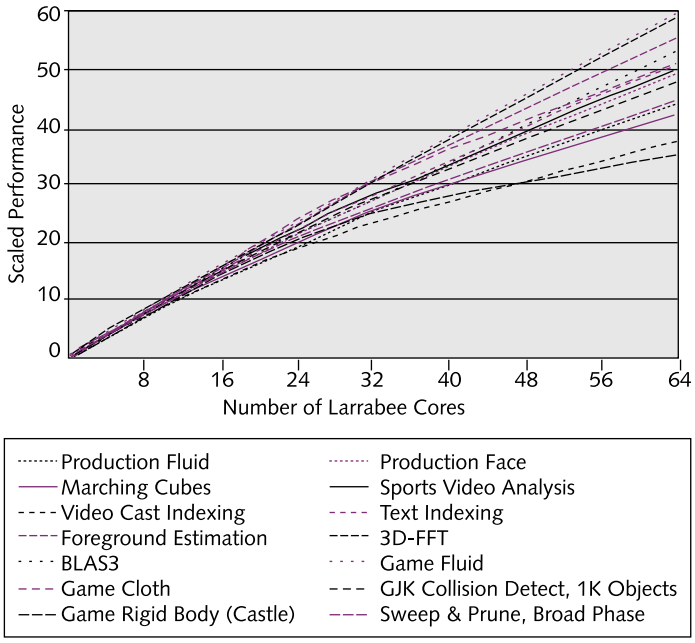

Figure 5. Preliminary benchmark testing on simulated Larrabee processors with eight to 64 cores. Most of these tests are graphics functions in action games, but some are math operations and other functions useful in a wider variety of applications. Although performance starts to fall off in some tests, it still rises at a nearly linear rate with dozens of cores. (Data source: Intel.)

relationship with the eight Synergistic Processor Elements (SPE), which are similar to Larrabee's VPUs.

Another important difference is that Cell's SPEs have a different architecture than the Power control processor. In Larrabee, the VPUs are an extension of the x86 architecture. Larrabee is a symmetrical, homogeneous multicore design, whereas Cell is an asymmetrical, heterogeneous multicore design. (See *MPR 2/14/05-01*, "Cell Moves Into the Limelight.")

Programmability Makes Trade-Offs

Full programmability with a general-purpose CPU architecture gives Larrabee an advantage in flexibility over other GPUs. Of course, the flip side of programmability is that a programmer must write the control code, which can be a disadvantage. Also, the more specialized hardware of other GPUs may score higher on 3D-graphics benchmarks while consuming less power. In other applications, Intel's less specialized hardware may have an edge. Overall, *MPR* expects competing GPUs to deliver more throughput per watt when running software that heavily exercises their fixed-function hardware, whereas Larrabee has a chance to shine when running software that's less graphics intensive but keeps the SIMD engines busy.

Intel claims that Larrabee's graphics performance will scale impressively across large numbers of cores. Figure 4 shows the performance of three action games running on simulated Larrabee processors that have as few as eight cores and as many as 48 cores. Within this range, at least, performance is remarkably consistent and scales at an impressively linear rate.

Figure 5 shows some additional preliminary benchmark results, again based on Intel's simulations. This time, Intel included a few tests outside the gaming realm and increased the maximum number of processor cores to 64. Performance doesn't scale quite as linearly as in the previous figure, but it's still impressive. Keep in mind that many programmers are still struggling to exploit dual- and quad-core microprocessors and would gladly trade a week's worth of Jolt Cola for results like this.

Scaling With Cores and Clock Speeds

We can't help noticing that Intel's simulations vary the number of processor cores from as few as eight per chip to as many as 64. Intel hasn't divulged the number of cores in the first Larrabee chips and cautions that test simulations aren't tantamount to product announcements. However, that range (8 to 64) is probably telegraphic. To compete in the discrete-graphics arena with the latest GPUs from ATI and Nvidia, high-end Larrabee devices will need dozens of cores. Likewise for Larrabee chips intended for HPC.

In addition to scaling the number of cores, Intel can also scale the clock frequency. Intel's Siggraph paper refers to a "Larrabee unit" as one processor core running at 1.0GHz. Hence, a 32-core device clocked at 1.0GHz equals 32 Larrabee units—as does a 16-core device

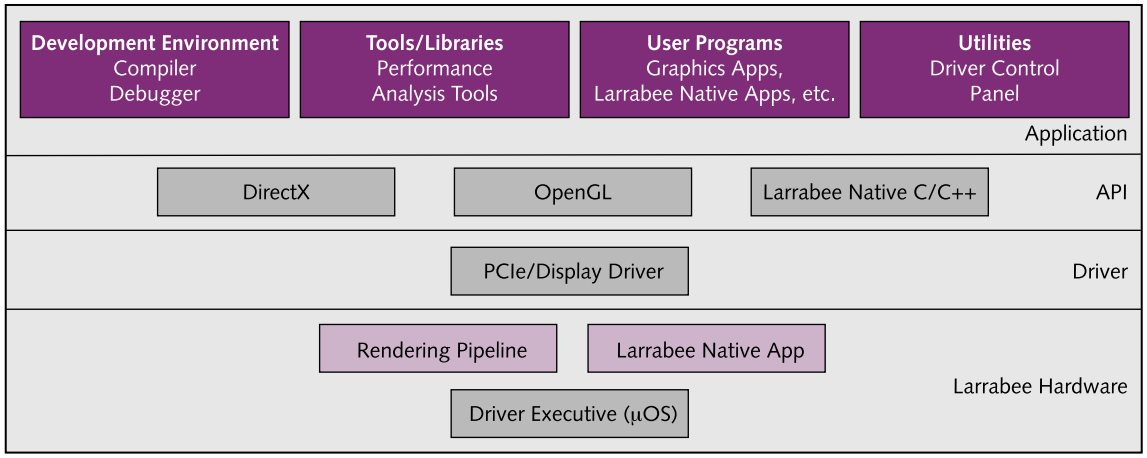

Figure 6. Larrabee's software stack. At the lowest level, a micro operating system from Intel runs on Larrabee (not on the host CPU) to support the higher-level driver software. Some programmers may tackle the challenge of writing native application software that runs directly on the micro operating system, bypassing the driver in a quest for higher performance. In most cases, however, application programs (such as games) will access Larrabee through the DirectX or OpenGL APIs. Those APIs run on the driver, which runs on the system's host CPU. In the future, host operating systems may have the ability to bypass the driver and use Larrabee's processors as if they were system-level processors, not just graphics coprocessors.

clocked at 2.0GHz. To keep power consumption within bounds, the maximum clock frequency probably won't exceed 3.0GHz, and even that speed pushes the envelope for a large implementation. At 3.0GHz, the maximum theoretical throughput of a large 64-core device would be six teraflops, but power consumption would probably be excessive. The greater use of fixed-function acceleration hardware in other GPUs helps keep them relatively power efficient, assuming the hardware is well utilized. When that hardware is idle, it becomes a liability.

In part, Larrabee's power/performance ratio depends on whether Intel introduces the chips in a 45nm or 32nm fabrication process. Around the time Larrabee debuts in 2009 or 2010, Intel will be migrating its leading-edge CPUs from today's 45nm process technology to the next-generation 32nm process. Although Intel intends to manufacture Larrabee in the same fabrication technology used for other Intel processors, it isn't clear which process will be the launchpad for Larrabee. Our guess is 45nm, because Intel probably wants to reserve the newer process for its leading-edge CPUs.

An often-overlooked aspect of Larrabee is that it's designed to scale downward as well as upward. Intel plans to integrate Larrabee into CPUs and chipsets for business PCs, notebook PCs, and low-end to midrange home PCs. It might even be suitable for a mobile Internet device (MID). For those systems, a Larrabee implementation with four or eight cores might make sense. Graphics performance will be lackluster by gamer standards but adequate for other purposes. Of course, integrated graphics save power, money, and board space.

With Larrabee, it might even be possible to continue utilizing the integrated graphics cores if a performance-hungry user installs a graphics card later. Today, installing a graphics card in a PC that has integrated graphics disables the built-in graphics processor, rendering it moot. In a Larrabee-integrated system, those integrated x86 cores might still remain part of the graphics pipeline or be available for other tasks. They are, after all, x86 cores capable of general-purpose processing.

However, to fully exploit this potential, operating systems must be able to recognize Larrabee's cores as system-level peer processors available for general-purpose workloads. Initially, Larrabee will be hidden from the operating system by its Intel driver software, just as ATI and Nvidia GPUs are hidden below their drivers today. (Figure 6 illustrates Larrabee's software stack.) *MPR* suspects that Apple will modify Mac OS X for Larrabee before Microsoft modifies Windows. Apple has more control over all aspects of hardware and software in its systems.

Multithreading at Multiple Levels

Conventional GPUs are massively threaded. They can marshal thousands of lightweight threads to perform tasks that have great inherent parallelism. In GPU terminology, a thread is a stream of operations running in one lane of a SIMD engine. Larrabee's VPUs support the same kind of multithreaded data parallelism, though not to the same degree. At a higher level, Larrabee also supports a threading model of task parallelism that has more in common with general-purpose CPUs. Larrabee can blend these data-parallel and task-parallel threading models together, a concept Intel calls "braided parallelism."

At the highest level of threading are the hardware-managed threads Intel calls Hyper-Threading on other x86 processors. These are really process threads, because they can execute a process as heavy as an application program or an operating system. They can also execute smaller tasks, such as

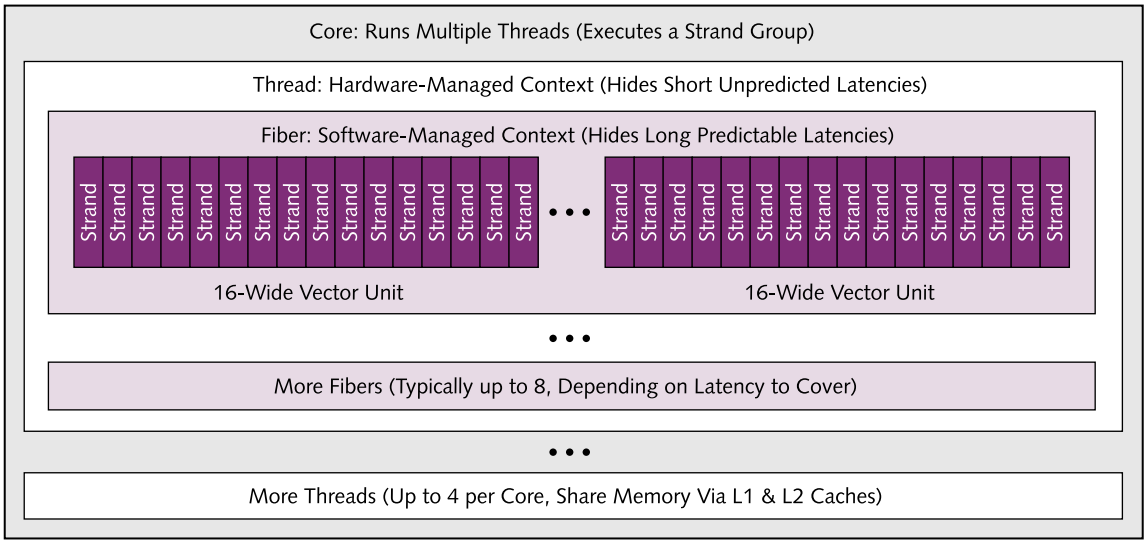

Figure 7. Larrabee's multithreading model. Heavyweight threads, managed by the processor cores, can run application programs and operating systems. Fibers are software-managed lightweight threads for SIMD operations. Strands are individual operands manipulated by SIMD instructions in the 16 lanes of each VPU. This figure illustrates data parallelism, not task-level parallelism. Intel refers to the combination of the two parallel-threading models as "braided parallelism."

the lighter-weight threads spawned by application programs and operating systems. As explained above, each x86 core in Larrabee can simultaneously manage up to four of these threads, allocating a separate register file and context to each one. These threads can share memory through the L1 and L2 caches.

On Intel's other multithreaded x86 processors, Hyper-Threading is largely transparent to application programmers. The operating system decides whether the processor will assign a separate hardware context to an application program or to a thread it spawns. Larrabee is similar. Programmers can invoke hardware-level threading through the OpenMP API or Intel's Thread Building Blocks, which build on the Posix Thread API. Multithreading at this level can be transparent to programmers. However, Larrabee gives programmers a little more control over "affinity"—the ability to assign hardware threads to specific processor cores, and hence to the same caches.

Posix pthreads map directly to hardware-level threads. Intel has extended the Posix API to let programmers assign pthreads to a particular processor core and even to a specific thread context on that core. The OpenMP API is similar; special pragmas in the Larrabee C/C++ compiler let programmers explicitly assign software threads to hardware-managed threads. At a slightly higher level of abstraction, Intel's Thread Building Blocks provide a task-scheduler API that maps software threads to hardware threads.

Untangling Threads, Fibers, and Strands

Below hardware-managed multithreading is a lower level of threading managed in software. These lower-level threads are called fibers. They execute tasks by running SIMD instructions on the VPUs. Compilers can decompose the data into chunks, which fibers can execute in parallel without stalling on interdependencies or memory-access latencies. In other words, if one chunk of data needs results from processing another chunk of data—or must wait for additional data to load—the compiler or run-time software can assign those chunks to different fibers that execute serially, not in parallel.

At the very lowest level of threading are individual operations destined for parallel execution in the 16-lane SIMD engines of the VPUs. As mentioned above, some new SIMD instructions in Larrabee can perform operations on as many as 16 operands at once. These 16 operations are subdivisions of a fiber; Intel calls them strands. Each strand corresponds to a thread on other GPUs. Figure 7 illustrates these different levels of multithreading.

In a previous example, we described the way a VPU can simultaneously operate on 16 pixels in the 16 lanes of a SIMD engine by processing the RGB components serially. That is, a SIMD instruction can operate on the red components of 16 pixels at once, followed by another SIMD instruction that operates on the green components of the same 16 pixels, followed by another SIMD instruction that operates on the blue components of those pixels. In this way, Larrabee can perform a complete operation on the RGB components of 16 pixels using only three instructions.

By intelligently assigning chunks of pixels to different fibers, compilers can use Larrabee's numerous SIMD engines to perform the same operations on many chunks of 16 pixels, in parallel with each other. The goal is to avoid

data dependencies and memory stalls among the fibers. This orchestration can be extremely efficient, though it requires the compiler to understand the execution latencies and caching characteristics of the processor. Different implementations of Larrabee—with different numbers of processor cores, different-size caches, and so on—may require programmers to recompile their software to get the best possible performance.

At the highest level, heavyweight threads composed of several fibers can take turns executing to hide delays caused by their own interdependencies and memory accesses. One thread might be a control process that's managing the whole thing. Of course, the same braided parallelism can be happening at the same time on other processor cores on the chip.

Larrabee's multilevel threading model appears mind-boggling, but APIs hide much of this complexity from application programmers. Indeed, when using the DirectX and OpenGL APIs, the shader compiler can handle the interleaving of threads, fibers, and strands. Game programmers often have in-house development tools for creating animation at even higher levels of abstraction. In addition, some third-party tool vendors intend to support Larrabee. Among them is RapidMind, which has an innovative development platform for parallel processors, including GPUs, IBM's Cell, and existing x86 chips. (See *MPR 11/26/07-01*, "Parallel Processing For the x86.")

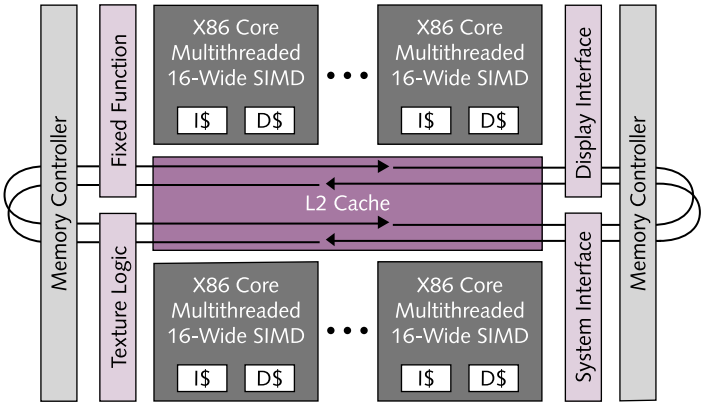

Figure 8. Block diagram of Larrabee's on-chip network. Four to 16 processor cores attach to a bidirectional ring with 512-bit datapaths in each direction. Other components attached to the ring include the shared L2 cache, memory controller, fixed-function graphics logic, display controller, and system I/O interfaces. Transferring data between each "stop" or node on the ring requires one clock cycle. Because the ring is bidirectional, traffic between the most distant cores needn't travel more than halfway around the ring. Intel has not disclosed the width or speed of the system I/O interface, which will undoubtedly vary with the implementation.

Intel Simplifies the On-Chip Network

Manycore and massively parallel processors often link their cores together in an on-chip network configured as a mesh or fabric. In this topology, each core has direct connections with all its neighbors. Local groups of cores are typically arranged in nodes or tiles, often with additional connections among their most distant members. Sometimes, long rows and columns of cores are linked by datapaths running across the chip. A mesh provides numerous local and global pathways among the cores, but it also requires lots of wiring. In some cases, the wiring is dense enough to limit the complexity of the cores or the size of the fabric.

In contrast, Larrabee has a simpler ring topology. Each x86 processor core, L2 cache, and block of fixed-function logic is attached to only one point on the ring, which extends from one end of the chip to the other. To reduce the latency of traffic circling the ring, the pathways are bidirectional—there's a 512-bit datapath in each direction (clockwise and counterclockwise). In addition, separate rings are dedicated to memory addressing and cache synchronization. Figure 8 is a block diagram of a four-processor ring.

Each "stop" or node on a ring represents one clock cycle of wire latency. Transferring data from one node to an immediate neighbor takes one cycle. Transferring data to a node that's two stops down the line requires two clock cycles, and so on. The worst-case latency for traffic circling the ring depends on the number of cores and other components attached to the network. There's no crossbar switch to provide shortcuts, so a communication between the two most distant nodes must travel halfway around the ring.

Intel settled on a ring topology to simplify the design, cut costs, and get to market faster. Although a ring isn't as versatile as a mesh, it simplifies the wire routing and allows Intel to populate the network with relatively complex 64-bit x86 processor cores. Most manycore and massively parallel chips that have mesh fabrics also have simpler processor cores. In some cases, those cores are relatively primitive 8- or 16-bit processors—although their design is also dictated by the specialized tasks they perform, not just by the complexity of their on-chip networks.

For example, most of PicoChip's massively parallel devices have several hundred 16-bit multiply-accumulate (MAC) processors and 16-bit integer processors arranged in a dense on-chip fabric. They specialize in signal processing, not graphics processing. (See *MPR 10/14/03-03*, "PicoChip Makes a Big MAC.") Interestingly, PicoChip's latest design—the PC302 chip for femtocells—significantly reduces the number of programmable processor cores in favor of greater hardware acceleration. However, the PC302 retains the PicoArray mesh fabric. Another example of a large-scale multicore processor with a mesh fabric is Tilera's 64-core Tile64 chip. In this design, the "tiles" are 64-bit VLIW processor cores. (See *MPR 11/5/07-01*, "Tilera's Cores Communicate Better.")

The traditional drawback of a ring network like Larrabee's is the longer signal delay when shuttling data between nodes that aren't immediate neighbors. In a mesh network, nearby nodes might have direct pathways to each other. According to Intel, the latency inherent in Larrabee's

ring network is less significant than the usual memory latency for loads and stores, so the ring won't slow things down. In other words, the processors will spend more time waiting for data to arrive from off-chip memory than they will spend waiting for data to circle the ring. Of course, this equation depends on the number of nodes attached to a ring and the likelihood that widely separated nodes will have to communicate with each other. Intel says the rings will be limited to four to 16 processor cores. Beyond that threshold, the ring grows so large that external memory latency won't hide the internal network latency.

For designs integrating more than 16 cores, Larrabee chips will implement multiple rings, perhaps with fewer than 16 cores each. Intel hasn't described the inter-ring connections in detail. Inevitably, communications between distant cores on different rings will suffer, but cleverly written (or cleverly compiled) software can minimize the need for these long-distance calls. By decomposing the data into chunks small enough to be crunched by the processors on a single ring, a program can distribute a large workload throughout the chip, with little or no need to share data among multiple rings.

Dividing and Sharing the L2 Cache

Each x86 processor core in Larrabee has its own L1 instruction and data caches, but they all share a global L2 cache attached to the ring. Each L1 cache is 32KB—four times the size of the L1 caches in the original Pentium. The extra room is necessary because Larrabee's quad-threaded cores would thrash a smaller cache while switching contexts among the threads. (The Pentium, of course, was a single-threaded processor.)

Although the L2 cache is a global resource, processor cores don't have unrestricted access to it. Each core is assigned a 256KB subset of the L2 cache. This arrangement has two benefits. First, and foremost, it allows all processors on a ring to access the L2 cache in parallel, each fetching data from its own portion of the cache. Parallel access greatly improves bandwidth between the L1 and L2 caches. Second, subdividing the L2 cache prevents one or a few cores from monopolizing the whole cache at the expense of their fellows.

The disadvantage of subdividing the cache in this manner is the possibility of uneven utilization. Depending on their workloads, some cores might starve for cached data while other cores make relatively little use of their portions. Uneven utilization is more likely to happen when multiple workloads are unbalanced or completely different from each other. For 3D graphics and other data-intensive tasks, the workloads should be sufficiently similar and evenly distributed to avoid this problem. This is another instance in which clever programmers or compilers can markedly improve performance.

The L2 cache is fully coherent. A separate ring in the on-chip network handles synchronization among the cache subsets dedicated to each core. When a core writes data to its portion of the L2 cache, the cache manager automatically signals the other subsets to flush their corresponding "dirty" data, if necessary. Processor cores can read data from their cache subsets in parallel with other cores, unless the ring's datapath becomes congested. In those cases, the stalled core can switch to another thread that may not be blocked.

Because Intel allots 256KB of L2 cache for each processor core, Larrabee can manipulate relatively large "tiles" of pixels when rendering graphics. Larger tiles are often preferred because they reduce the chance of a single polygon spanning multiple tiles. Intel says that as long as an entire tile fits in a 256KB subset of the cache, there's almost no penalty in rendering speed for using larger tiles. A tile measuring 128 × 128 pixels in 32-bit color plus depth requires 128KB—only half the space available in each core's subset of the L2 cache. That leaves plenty of room for caching other data.

Overall, Larrabee's memory hierarchy and load/store model more closely resemble a general-purpose CPU than a conventional GPU. Data flows through the L1 and L2 caches as expected, with little or no manual intervention, although programmers can certainly use the new prefetch instructions and other cache-control operations to optimize critical functions. Programmers will welcome Larrabee's memory model, particularly for applications other than graphics. Other GPUs have a more distributed memory model, forgoing a large shared memory like Larrabee's global L2 cache.

The Importance of x86 Compatibility

Larrabee has the potential to upset the GPU market. By reverting to a 15-year-old x86 microarchitecture (albeit with many improvements) and adopting a simple ring for the on-chip network, Intel has created a very scalable basic design. A small implementation can be integrated into a system chipset or host CPU. Larger implementations can aim for different segments of the discrete-GPU and HPC markets. These markets aren't zero-sum games, but if Larrabee succeeds, it will take sales from ATI and Nvidia.

For software developers, system vendors, and users, there are advantages to having all the system's processors based on the x86. The same basic driver software could work with integrated graphics or discrete graphics. In a system with discrete graphics that isn't currently running a heavy graphics workload, a Larrabee GPU could offload other tasks from the host CPU. In a PC with integrated Larrabee graphics, a performance-minded user could add a Larrabee graphics card, and the system could still use the integrated Larrabee cores to assist the discrete GPU or perform other tasks.

More companies are trying to leverage the processing power of GPUs for things other than graphics, even in ordinary PCs. The latest versions of Adobe Photoshop, After Effects, and Premiere Pro can outsource such tasks as image rotation, zooming, panning, antialiasing, compositing, and video effects to an Nvidia Quadro or GeForce GPU. Elemental Technologies recently announced the Badaboom Media Converter, which uses Nvidia GPUs for video transcoding—a

chore that runs painfully slow on CPUs. Nvidia is encouraging developers to create more products like these. In 2006, Nvidia introduced the Compute Unified Device Architecture (CUDA), a C/C++ platform for developing and running massively parallel software on Nvidia processors. (See *MPR 1/28/08-01*, "Parallel Processing With CUDA.")

CUDA has some impressive wins. Among other things, it's used for geographic information systems, by Manifold; for biochemistry, by Max Planck; for weather modeling, by the National Center for Atmospheric Research; for financial modeling, by SciComp; for oil and gas exploration, by SeismicCity; and for medical imaging, by TechniScan. The recent release of CUDA client software for protein folding has added more than 1.2petaflops (1,200 trillion floating-point operations per second) of distributed processing power to Stanford University's Folding@Home project on the Internet.

Although CUDA is gaining followers, it's surprising how long GPUs have been underutilized in millions of PCs. When a PC with a powerful graphics card isn't running an action game or something equally taxing, the GPU is little more than a case heater. An x86-based GPU like Larrabee might attract more software development and put those wasted resources to work.

That said, we don't wish to overstate the importance of Larrabee's x86 compatibility. In some ways, the compatibility is superficial or irrelevant. Existing programs written and compiled for single-core or even multicore x86 CPUs cannot take advantage of Larrabee's manycore resources without some recoding and recompilation. The extent of modifications depends greatly on the program. Developers must banish all fantasies of simply recompiling their serial code to run on a parallel processor—whether the parallel processor is Larrabee or something else. Existing software that uses MMX or SSE instructions will need rewriting, too, because Larrabee's Pentium-derived x86 core lacks those instructions.

In fact, Larrabee's x86 heritage will likely be invisible to most application programmers. They usually write software to high-level APIs, not to the low-level instruction-set architecture. The style and quality of the development tools will be more influential than the instruction set. Indeed, the architectures of ATI and Nvidia GPUs have always been hidden from programmers by driver-level software. Their architectures can change from one generation of GPUs to the next while maintaining compatibility with application software. Nevertheless, the attraction of x86 compatibility is a factor that could lure developers toward Larrabee.

Speculating on Intel's Strategy

At times in this article, we have expressed doubts that Larrabee will exceed or even match the power/performance benchmarks of GPUs from ATI and Nvidia. Of course, this is pure speculation on our part, because Larrabee silicon isn't yet available for independent testing. One reason for our skepticism is that an x86-based GPU with minimal hardware acceleration seems poorly matched against highly specialized GPUs from vendors with more experience in the field. Furthermore, we perceive that superior graphics performance is not Intel's top priority for Larrabee—at least, not at this time.

However, Larrabee can succeed without beating ATI and Nvidia in 3D graphics. Even if Larrabee doesn't dethrone the champions, it can hurt them by capturing market share in the lower segments of the graphics market. Larrabee can also win devotees in the growing HPC market. If ATI and Nvidia continue pursuing HPC more aggressively, they may be forced to make architectural changes that impair the progress of their own graphics performance. Over time, as the GPUs from all vendors edge closer to general-purpose processing, the playing field could level out.

Another possibility is that Intel could use Larrabee as a template for different manycore-processor designs. Only a few of the new SIMD extensions in Larrabee are specific to graphics. By swapping Larrabee's fixed-function graphics logic for other kinds of application-specific logic, Intel could produce chips optimized for networking, communications, video processing, signal processing, and other purposes. With the addition of some string-handling acceleration, a future descendent of Larrabee might even make a killer Googleplex-on-a-chip.

Intel's fab strategy is another factor to consider. Although Intel is rightly regarded as the world's largest microprocessor company, it's also the world's largest captive foundry. To sustain its huge capital investments in multibillion-dollar fabs and biennial process-technology advances, Intel must generate the massive revenues that only high-volume products can deliver. Intel can't bother with niche products or markets, so the general-purpose orientation of Larrabee could be a plus.

Larrabee could help Intel wring more profit from its fabs by creating demand for GPUs and GPU-integrated chipsets that trail the company's leading-edge process technology by a generation. For instance, Intel could manufacture the first Larrabee devices at 45nm while moving its high-end PC and server processors to 32nm. The 45nm fabs would stay busy a few years longer, working wonders with Intel's amortization tables. On the other hand, if Intel has enough fab capacity to manufacture top-of-the-line Larrabee parts in the latest-generation fabrication technology, it would exert more pressure on ATI and Nvidia to keep up.

Larrabee's Success Isn't Guaranteed

What might go wrong for Intel? If Larrabee's 3D-graphics performance isn't competitive, ATI and Nvidia can successfully defend their high-end graphics business. Intel could respond by throwing more cores at the problem, but heat and power will soon impose a limit. If Intel tries to compensate by manufacturing high-end Larrabee devices in its latest fabrication technology, the additional load on Intel's leading-edge fabs could slow production of Intel's all-important PC and server processors.

For More Information

Intel plans to introduce the first Larrabee devices in 2009 or 2010. They will be GPUs on expansion boards for PCs, primarily for games but also suitable for high-performance computing applications. Later implementations will be integrated with system logic or CPUs. Intel will announce specific products, prices, and availability near introduction. For more information, visit:

- *www.intel.com/pressroom/archive/releases/20080804fact.htm*

Intel's Siggraph paper on Larrabee is available here:

- *http://softwarecommunity.intel.com/UserFiles/en-us/File/larrabee_manycore.pdf*

Even lower-end graphics customers might avoid Larrabee if AMD does a better job of integrating graphics with PC processors than Intel does. Poor graphics performance could make HPC customers reluctant to use Larrabee, too.

Another obstacle is software development. If porting or writing code for Larrabee appears too daunting, the initial lure of x86 compatibility could lose its luster. Larrabee doesn't solve the fundamental computer-science problems of writing parallel software or debugging multithreaded software. Indeed, Larrabee's multiple threading models will leave some programmers scratching their heads and others fretting about deadlocks. (See *MPR 7/28/08-02*, "Tools for Multicore Processors," and *MPR 4/30/07-02*, "The Dread of Threads.")

Larrabee's driver software is another potential trapdoor. The driver must translate DirectX or OpenGL calls into native code for Larrabee's unconventional graphics pipeline. ATI and Nvidia have been optimizing their drivers for ten years. When Larrabee debuts, Intel's drivers will have been under development for maybe two years. Driver bugs—always a hazard with new GPUs—can break popular games and other graphics programs. Already, some PC users complain about software compatibility on Intel's integrated-graphics chipsets. Early adopters of Larrabee might wish they had made the safe choice of buying an ATI or Nvidia graphics card instead.

For AMD/ATI and Nvidia, the biggest threat from Larrabee may be psychological. They must fight any perception that Intel's onslaught is irresistible. Our conclusion is that Larrabee is an innovative design and a potential winner, but triumph is by no means certain, and even its success won't necessarily clear the field of competitors.