Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

NUMBER CRUNCHING WITH GPUS

PeakStream's Math API Exploits Parallelism in Graphics Processors

By Tom R. Halfhill {10/2/06-01}

There are dozens, if not hundreds, of microprocessor architectures in the world. And *Microprocessor Report* covers new ones every year. They span the gamut from CISC to RISC to VLIW, from tiny four-bit microcontrollers to mighty 64-bit supercomputer processors with multiple cores. With such abundance, it might seem daffy to use highly specialized 3D-graphics coprocessors for general-purpose number crunching.

But the computational allure of GPUs is proving irresistible to the scientific community, chemical engineers, defense contractors, Wall Street financiers, and other heavy-duty math junkies. For their data-intensive applications, they never seem to get enough processing power, even from the most muscular general-purpose processors. They are gazing hungrily at the graphics processors on PC graphics cards and in the latest videogame consoles. Today's best GPUs can execute an astonishing 360 billion floating-point operations per second (gigaflops)—more than seven times the performance of the fastest x86 dual-core processor. Yet their typical application is to repeatedly draw and obliterate imaginary enemies for the amusement of people who can find nothing better to do than play a videogame.

What a waste. Not of synapses—of silicon! Surely we can find something more productive for those GPUs to do.

So says PeakStream, a Silicon Valley startup that was founded in February 2005 and emerged from stealth mode on September 18, 2006. PeakStream's founders hail from Network Appliance, Nvidia, Stanford University, Sun Microsystems, and VMware. They were inspired by Stanford's Brook Project on stream programming, and they have raised $17 million from Kleiner Perkins Caufield & Byers, Sequoia Capital, and Foundation Capital. PeakStream's mission is to provide low-level software and development tools that make GPUs relatively easy to program for general-purpose number crunching.

It's not a simple mission, because GPUs are notoriously difficult to program. They are highly parallel multipipelined processors with limited instruction sets. Obviously, they are optimized for the relatively narrow demands of 3D graphics. Game programmers are accustomed to bizarre architectures and have the advantage of using high-level application programming interfaces (API) like OpenGL and Microsoft's DirectX. But those APIs are unsuitable for programmers writing number-crunching software for scientific and technical applications. The only alternative may be assembly language.

To preserve the sanity of mere mortals, PeakStream has developed a software package called the PeakStream Platform. Its main feature is an API with 80 function calls for common mathematical operations. Programmers can call these functions from high-level C++ code written with industry-standard development tools. Special array structures automatically configure large data sets for parallel processing on the GPU.

For now, PeakStream supports only the ATI Radeon R580 graphics processor. It's the same GPU found on ATI's Radeon X1900-series graphics cards for the PC market, although PeakStream is using a specialized card. PeakStream hopes to port the API to other GPUs and to IBM's Cell Broadband Engine in the near future. The company says its programmers need only about three months to port the platform to a new processor.

Virtual Machine Enables Code Portability

The advantages of an off-the-shelf API will be apparent to anyone even vaguely familiar with the complex architectures of modern GPUs. Instead of learning a strange architecture and laboriously scribing their code in assembly language, programmers can write in a familiar high-level language augmented with useful function calls. Not only will their code be easier to write, but it will also be more portable. The architectures and programming models of GPUs can change significantly from one generation to another, and GPUs from different vendors have almost nothing in common. By using the PeakStream Platform, programmers can run their binary code on a new GPU, or even a completely different GPU, without recompiling.

PeakStream hasn't resorted to any kernel-level hacks to enable this portability. As Figure 1 shows, the PeakStream Platform includes a virtual machine that runs as a user-level task, cleanly isolating the application code from the gory details of the processor. Even the PeakStream math libraries behind the API function calls run on the virtual machine, thanks to a just-in-time (JIT) compiler. In many respects, the PeakStream Platform is like a Java run-time environment. The key differences are that programmers write source code in C++ instead of Java, and the source code compiles to native x86 binary code, not to Java bytecode or any other intermediate code.

Application programmers can use industry-standard C/C++ development tools, such as Microsoft's Visual Studio, the Intel Compiler, or GNU C (GCC). PeakStream supplements those tools with a plug-in debugger and a performance profiler. Programmers write their C++ code using PeakStream's function calls and special array structures (more on this later). The compiled binary is native x86 code that runs on any x86-based computer. (The first version of the PeakStream Platform is for Linux servers, but PeakStream plans to release future versions for Microsoft Windows.)

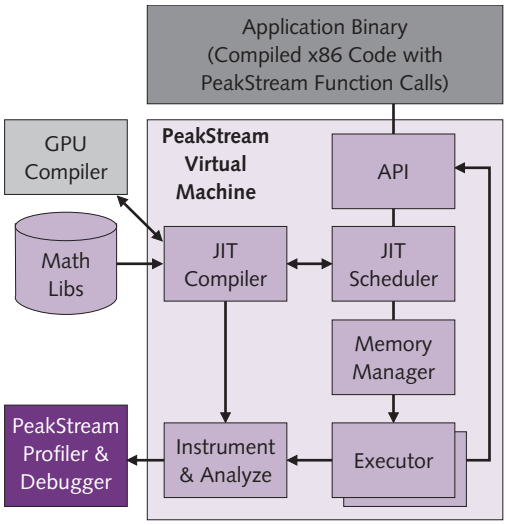

Figure 1. The PeakStream Platform includes a special virtual machine and a just-in-time (JIT) compiler. Programmers write application code in C++ and compile to a standard x86 binary, which embeds the function calls to PeakStream's math libraries. At run time, the JIT compiler converts the function calls into the target processor's machine language. This arrangement preserves code portability across different GPU architectures, allows the virtual machine to exploit the GPU's inherent parallelism, and relieves the application programmer of memory-management chores.

In effect, the x86 binary code is merely control code. At run time, PeakStream's virtual machine intercepts the special function calls embedded in the x86 binary. The JIT compiler converts those routines into the GPU's native machine language, and it caches frequently used code in main memory for later use. The virtual machine also determines how to extract data parallelism from the application code, depending on the type of GPU in the host system. Because those decisions happen at run time, the application code isn't bound to any particular GPU implementation or architecture. It's inherently portable.

PeakStream says it doesn't necessarily require technical assistance from GPU vendors to implement any of this low-level software, because publicly available documentation contains all the pertinent information. However, PeakStream has established a development and joint-marketing relationship with ATI, which, for now, makes the only GPU that is supported. Nvidia says it will provide technical support when PeakStream ports the platform to an Nvidia GPU.

PeakStream Platform Has a Few Shortcomings

The ATI Radeon R580 that PeakStream supports will execute 360 gigaflops in single precision at 650MHz. That's nearly the highest performance available today in a GPU, which is the reason PeakStream chose the R580 as its first target. The maximum theoretical throughput of the fastest x86 dual-core processor (Intel's 3.0GHz Core 2 Duo) is 48 single-precision gigaflops—less than 14% the performance of the Radeon R580. No wonder the most avid number crunchers find GPUs an attractive alternative. However, the PeakStream Platform does have shortcomings.

PeakStream's virtual-machine approach is wonderful for code portability, but one drawback is that a virtual machine and a JIT compiler usually don't deliver the same high performance as statically compiled code running natively on the target processor. Although C++ code written for the PeakStream Platform compiles to a native x86 binary, the virtual machine and JIT compiler don't translate the PeakStream library functions into native code for the GPU until run time.

PeakStream doesn't view this limitation as significant. Thanks largely to the popularity of Java, virtual-machine technology and JIT compilers have made tremendous progress in the past ten years. In some cases, programs compiled at run time can exceed the performance of statically compiled programs, especially if they use run-time metrics to optimize the compilation. PeakStream's JIT compiler currently doesn't use run-time profiling for that purpose, but it's a feature on the company's to-do list. What's more important

is that JIT compilation allows the PeakStream virtual machine to optimize data parallelism at run time, according to the specific microarchitecture of the GPU.

A more significant limitation of the PeakStream Platform is that it's currently much more suitable for single-precision (32-bit) floating-point operations than for double-precision (64-bit) operations. PeakStream's API supports double precision as easily as single precision, but double-precision function calls currently execute on the system's host processor (the x86 CPU), not on the GPU. Result: even on a 3.0GHz Intel Core 2 Duo, maximum theoretical throughput plunges 94%, to 24 gigaflops. This limitation will disappoint scientists, mathematicians, and others who need double precision. PeakStream estimates that about half the mathematical applications in high-performance computing require double precision.

Don't blame PeakStream for this disadvantage. Today's GPUs and game-oriented CPUs are highly optimized for single-precision math, because it's sufficient for calculating the vertices of 3D-graphics polygons in videogames. For example, in single precision, IBM's smoothly pipelined Cell BE can execute 256 gigaflops at the chip's nominal clock frequency of 3.2GHz. In double precision, that rate falls to 25 gigaflops—nothing to be ashamed of, certainly, but nowhere near the breakneck single-precision pace. (See *MPR 3/13/06-01*, "The Cell, at One.")

PeakStream expects GPUs will advance to double precision by 2008 to provide better realism and more-detailed 3D graphics. Already, digital-movie houses like Pixar Animation Studios render all their graphics in double precision, distributing the heavy workload across vast render farms of networked computers. PeakStream figures it's only a matter of time before gamers and others demand the same quality in real-time graphics on PCs and

PeakStream API Function Calls	
Basic Operators and Functions	**Reduction Functions**
+, –, *, / (add, sub, mul, div)	B = all(A, [dim])
% (modulus)	B = any(A, [dim])
&&, ||, ! (logical AND, OR, NOT)	B = max_value(A, [dim])
<, <=, ==, >, >= (comparisons)	B = mean(A, [dim])
gather1_floor (create & reorder array)	B = min_value(A, [dim])
gather2_floor (create & reorder array)	B = product(A, [dim])
Array-Generation Functions	B = stddev(A, [dim])
A = identity_fNN(size)	B = sum(A, [dim])
A = index_fNN(dim, size0, [size1=1])	B = variance(A, [dim])
A = ones_fNN(size0, [size1=1])	**Miscellaneous Functions**
A = zeros_fNN(size0, [size1=1])	B = abs(A)
Attribute-Query Functions	C = fmod(A,B)
size = A.get_cols()	C = fmod_inplace(A, B)
dim = A.get_dims()	B = cdf_normal(A, MU, SIGMA)
size = A.get_rows()	B = is_nan(A)
size = A.get_size([dim])	C = max(A,B)
bool = A.is_same_shape(B)	C = min(A,B)
bool = A.is_scalar()	B = sign(A)
Elementwise Functions	**Linear Algebra**
B = ceil(A)	B = dot_product(A, B, [dim])
B = floor(A)	B = matmul(A, B)
B = round(A)	B = transpose(A)
B = trunc(A)	B = lu_condition(LU, NORMOFA)
B = exp(A)	lu_decomp(A, LU, PIVOT, SINGULARITY)
B = log(A)	C = lu_solve(LU, PIVOT, B)
B = log10(A)	**Basic Linear Algebra Subprograms**
B = log2(A)	cblas_daxpy(n,ALPHA,X,incX,Y,incY)
C = pow(A,B)	cblas_saxpy(n,ALPHA,X,incX,Y,incY)
B = pow10(A)	cblas_dcopy(n,X,incX,Y,incY)
B = pow2(A)	cblas_scopy(n,X,incX,Y,incY)
B = sqrt(A)	A = cblas_ddot(n,X,incX,Y,incY)
Trigonometric Functions	A = cblas_sdot(n,X,incX,Y,incY)
B = acos(A)	cblas_dgemm(order,transA,transB, m,n,k,ALPHA,A,lda,B,ldb,BETA,C,ldc)
B = acsc(A)	cblas_sgemm(order,transA,transB, m,n,k,ALPHA,A,lda,B,ldb,BETA,C,ldc)
B = asec(A)	cblas_dgemv(order,transA,m,n, ALPHA,A,lda,X,incX,BETA,Y,incY)
B = asin(A)	cblas_sgemv(order,transA,m,n, ALPHA,A,lda,X,incX,BETA,Y,incY)
B = atan(A)	cblas_dger(order,m,n,ALPHA,X,incX,Y,incY,A,lda)
C = atan2(A,B)	cblas_sger(order,m,n,ALPHA,X,incX,Y,incY,A,lda)
B = cos(A)	cblas_dnrm2(n,X,incX)
B = cosh(A)	cblas_snrm2(n,X,incX)
B = cot(A)	cblas_dscal(n,ALPHA,X,incX)
B = coth(A)	cblas_sscal(n,ALPHA,X,incX)
B = csc(A)	cblas_dswap(n,X,incX,Y,incY)
B = csch(A)	cblas_sswap(n,X,incX,Y,incY)
B = sec(A)	
B = sech(A)	
B = sin(A)	
B = sinh(A)	
B = tan(A)	
B = tanh(A)	

Table 1. PeakStream's API makes all these function calls available to programmers in C++ by including the header file peakstream.h in the application source code. Common arithmetic operators also work with the special array structures. Application code written in C++ using these functions will run on different generations of a GPU architecture and even on different architectures—if PeakStream supports them. Currently, PeakStream supports only the ATI Radeon R580.

game consoles. GPUs would require relatively little extra hardware to support double precision if they run 64-bit operations at half the speed of 32-bit operations.

Record-breaking double-precision performance requires significantly more hardware. For the world's fastest supercomputer—the Lawrence-Livermore BlueGene/L—IBM created a "double hummer" FPU. In essence, IBM cloned the entire 64-bit pipeline and register file of a single-pipelined 64-bit FPU. This remarkable processor crunches through double-precision math as quickly as it handles single-precision. Fully configured with 131,072 processors, BlueGene/L's peak performance is 367 trillion floating-point operations per second (teraflops). That's three orders of magnitude faster than a Radeon R580—albeit with five orders of magnitude more processors. (See *MPR 10/11/04-01*, "IBM Makes Designer Genes.")

API Supports Common Math Functions

Table 1 lists all the function calls in PeakStream's API. These calls are available to programmers after including a header file called peakstream.h in the include section of a C++ program. Note several functions for creating and manipulating special arrays. These arrays are the key structures in PeakStream's data parallelism. In addition to these functions, the API supports familiar symbolic operators for manipulating arrays, such as = for assignments, == for equality comparisons, && for logical AND, ! for unary operations, and the usual arithmetic operators. Most function calls have meaningful names and are largely self-explanatory.

Figure 2 shows two examples of C++ source code. Example A is typical sequential code for approximating the value of pi. Example B has been refactored after including the peakstream.h library. It uses PeakStream's special arrays to exploit the parallel-processing capabilities of a multi-pipelined GPU. PeakStream provides a profiling tool that helps application programmers find such opportunities for data parallelism in their code.

For now, the PeakStream Platform uses only the pixel-shader pipelines and caches of the GPU. The GPU's vertex shaders, texture-processing filters, video decoders, and other hardware lie fallow. It seems like a waste of processing resources, but PeakStream has good reasons for this decision.

First, the pixel shaders are the most computationally powerful parts of the Radeon R580 and occupy most of the chip's logic. It would take much more work for PeakStream to parcel out the number crunching to other parts of the GPU. Second, that work would probably become redundant. ATI's next-generation GPUs will have a unified shader architecture that devotes one set of resources to pixel, vertex, and geometry shading. Unified shaders are a step toward easier programmability, and a future version of Microsoft's DirectX graphics API (Direct X 10) is expected to support this feature. Of course, ATI is taking this step primarily for the benefit of graphics programmers, but it

****SAMPLE A: TRADITIONAL SERIAL CODE****

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>

int main ( int argc, const char** argv ){
  long int i;
  float x, y;
  float num_inside, distance_from_zero;
  float Pi;

  num_inside = 0.0f;
  for( i = 0; i < 1000000; i++ ) {
    x = float( rand() ) / float( RAND_MAX + 1 );
    y = float( rand() ) / float( RAND_MAX + 1 );
    distance_from_zero = sqrt( x*x + y*y);
    if ( distance_from_zero <= 1.0f )
      num_inside += 1.0f;
  }
  Pi = 4.0f * ( num_inside / NSET );

  printf("Value of Pi = %f \n",Pi);
}
```

****SAMPLE B: PEAKSTREAM PLATFORM CODE ****

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <peakstream.h>

using namespace SP;
using namespace SP::Generator;

int main ( int argc, const char** argv ){
  float Pi_cpu;
  init();
Arrayf32 Pi;
{
   DefaultGenerator32 G; // Random Number generator handle
   Arrayf32 X = G.make(10000000); // distributed in [0,1)
   Arrayf32 Y = G.make(10000000); // distributed in [0,1)
   Arrayf32 distance_from_zero = sqrt(X * X + Y * Y);
   Arrayf32 inside_circle = cond((distance_from_zero <= 1.0f), 1.0f,
0.0f );
     Pi = 4.0f*sum(inside_circle)/NSET;
  }
  Pi_cpu = Pi.read_scalar();

  printf("Value of Pi = %f \n",Pi_cpu);
shutdown();
}
```

Figure 2. These two examples of C++ source code are routines for approximating the value of pi. Example A is typical sequential code without using any special function libraries. Example B uses PeakStream's math library (included as peakstream.h). Note the call to PeakStream's fast random-number generator and the operations on arrays of 32-bit floats. The program defines two arrays, each filled with one million random numbers. After defining the arrays, the program multiplies and adds them together to create a third million-element array, then calls another PeakStream function to find the square root. At run time, PeakStream's virtual machine automatically determines how to execute this code for maximum data parallelism on the target processor. To convert this code to double precision, the programmer would change the array definitions from "Arrayf32" to "Arrayf64".

will also allow programmers in other fields to utilize the chip's resources to their fullest.

More Applications Need High Performance

Applications that demand very high levels of processing power fall into several categories. Life sciences are a rapidly growing field, especially for molecular modeling, drug discovery, genome mapping, and protein folding. Some life-science applications require double-precision math. Nevertheless, scientists in this field are looking eagerly at single-precision GPUs.

One example is the Folding@Home Project, which runs a distributed protein-folding program on more than a million computers connected to the Internet. Folding@Home is patterned after another massively parallel Internet collaboration, SETI@Home (Search for Extraterrestrial Intelligence). A related effort is the Cure@PS3 Project, which hopes to use thousands of Cell BE processors in Sony PlayStation 3 game consoles for medical research.

Oil and natural-gas exploration is another hot field, and single-precision math is often good enough. An example is Kirchhoff migration—a mathematical method of analyzing the echoes from test explosions to determine the composition of subsurface geological layers or the contours of a seafloor. Faster analysis allows geologists and engineers to get results from their probes in real time. PeakStream says a Kirchhoff-migration program written with its API can analyze two billion samples per second, compared with only 60 million samples per second for the same program running on a regular CPU.

Wall Street financiers are demanding more compute power, too. Faster processors allow them to perform more-complex analyses of financial data and offer new investment products. One example is a Monte Carlo simulation, so called because it generates random numbers for interest rates, currency exchange rates, and other economic variables as part of an elaborate simulation that estimates the future values of fixed-income derivatives, such as exotic swaptions. Faster processing might allow a stock brokerage to perform all the necessary calculations overnight, after the stock market closes, so brokers can sell the investments the next morning, when the market opens. PeakStream says a Monte Carlo simulation developed with its API can generate 700 million pseudorandom numbers per second, about ten times the performance of a 2.6GHz dual-core AMD Opteron.

The military has an insatiable appetite for performance, as well. PeakStream says a large government systems integrator has used the API to write a signal-processing program in only two weeks that previously took six months to hand-code in assembly language. For this application, the primary goal was to reduce the computer's weight. Running the PeakStream Platform on a GPU allowed the developers to cram more processing power into a smaller package. PeakStream cannot disclose the exact nature of the application, but the customer is frequently associated with pilotless drone aircraft.

Price & Availability

The PeakStream Platform is available now. The first version, called PeakStream Server, is for an x86-based Linux server with a single ATI Radeon R580 graphics processor. Pricing starts at $2,000 per GPU for commercial users. PeakStream offers volume discounts and academic discounts. For more information about PeakStream, visit *www.peakstreaminc.com*.

- For information about ATI's Radeon R580 graphics processor, see *www.ati.com/products/radeonx1900/specs.html*
- For information about Stanford University's Brook Project, see *http://graphics.stanford.edu/projects/brookgpu/index.html*.
- For information about the Folding@Home Project, see *http://folding.stanford.edu/*.
- For information about the Cure@PS3 Project, see *http://folding.stanford.edu/FAQ-PS3.html*
- For a list of the world's fastest supercomputers, see *www.top500.org/list/2006/06/100*.

Sharp Growth in High-Performance Computing

PeakStream is entering a relatively new but interesting business. GPU vendors annually ship tens of millions of chips on PC graphics cards and in game consoles, so the high-performance computing market may seem puny in comparison. Yet according to IDC, high-performance and technical computing generated $9.2 billion in revenue in 2005, up 24% from 2004. It was the second consecutive year of revenue growth exceeding 20%.

Meanwhile, the PC graphics market is changing in ways that create uncertainty. AMD recently spent $5.4 billion to acquire ATI, which significantly alters the competitive landscape. (See *MPR 8/28/06-03*, "AMD Writes a New Chapter for PCs.") Intel continues integrating better graphics into some of its PC chip sets, which tends to shrink the audience for discrete GPUs to performance-obsessed gamers. Both developments are forcing the largest surviving independent GPU vendor, Nvidia, to reevaluate the graphics market and its prospects for future growth.

At present, the PeakStream Platform exploits data parallelism on only a single GPU. Developers could tap much greater parallelism if PeakStream supported operations on multiple GPUs—whether those chips were clustered on a single board, on multiple boards in a single system, or distributed across multiple systems. The Folding@Home, Cure@PS3, and SETI@Home projects chart a fascinating course for massively parallel "supercomputers" linking thousands of ordinary PCs and game machines on the Internet. Supporting multiple GPUs ganged together in parallel is an ambition for PeakStream's future development.

One alternative to using GPUs for high-performance computing is a new breed of floating-point coprocessor. Discrete FPUs were common 20 years ago, when microprocessors lacked integrated FPUs. Longtime *MPR* readers may recall Weitek's popular FPUs for x86 processors. Intel's first x86 processor with an integrated FPU was the 486DX, introduced in 1989; by the early 1990s, external FPUs had become redundant. Recently, however, the concept has been revived, primarily by ClearSpeed Technology, a British company. In 2003, ClearSpeed introduced the CS301, a massively parallel floating-point coprocessor capable of delivering 25.6 gigaflops at only 200MHz. (See *MPR 1/12/04-02*, "ClearSpeed Hits Design Targets," and *MPR 11/17/03-01*, "Floating Point Buoys ClearSpeed.") The CS301 was primarily a development chip and was quickly superseded in 2004 by the CSX600. It delivers 25 gigaflops and consumes only 10W.

Eight of ClearSpeed's Advance accelerator boards (each with two CSX600 chips) clustered in four Hewlett-Packard Proliant DL380 G5 servers recently achieved 364.2 gigaflops while consuming only 200W additional power. This modest installation is nearly as fast as the world's best supercomputer in 1996. Although the HP cluster's performance is only slightly faster than the 360 gigaflops of a single Radeon R580, the ClearSpeed FPUs execute double-precision math—a make-or-break advantage for some applications. ClearSpeed is also supplying 720 coprocessors to Sun Microsystems for Japan's largest supercomputer, now under construction for the Tokyo Institute of Technology. ClearSpeed's coprocessors will assist more than 10,000 AMD Opteron processor cores. Performance is expected to reach 85 teraflops.

Despite this competition, GPUs show great promise. They evolve faster than more-specialized processors because the high-volume game market fuels their development. *MPR* expects that GPU vendors will recognize enough potential in high-performance computing to begin tweaking their future designs in small ways that encourage it. The consumer market is still the main driver, of course, but relatively minor improvements in programmability would make a big difference to developers that have other applications in mind. A few years from now, when double-precision math becomes a consumer-market necessity, GPUs will overcome their worst shortcoming and evolve into very competitive engines for high-performance, low-cost computing.