2010

High-Performance Heterogeneous Computing with the Convey HC-1

Jason D. Bakos
University of South Carolina - Columbia, jbakos@cse.sc.edu

Publication Info

Published in *Computing in Science and Engineering*, ed. Volodymyr Kindratenko and Pedro Trancoso, Volume 12, Issue 6, 2010, pages 80-87.
http://ieeexplore.ieee.org/servlet/opac?punumber=5992

Editors: Volodymyr Kindratenko, kindr@ncsa.uiuc.edu
Pedro Trancoso, pedro@cs.ucy.ac.cy

High-Performance Heterogeneous Computing with the Convey HC-1

By Jason D. Bakos

Unlike other socket-based reconfigurable coprocessors, the Convey HC-1 contains nearly 40 field-programmable gate arrays, scatter-gather memory modules, a high-capacity crossbar switch, and a fully coherent memory system.

At Supercomputing 2009, Convey Computer unveiled the HC-1, an all-in-one compute server containing a socket-based reconfigurable coprocessor board. The HC-1 is unique in several ways. Unlike in-socket coprocessors from Nallatech (www.nallatech.com/Intel-Xeon-FSB-Socket-Fillers/fsb-development-systems.html), DRC (www.drccomputer.com/drc/modules.html), and XtremeData (www.xtremedata.com/products/accelerators/in-socket-accelerator/xd2000i)—all of which are confined to a socket-sized footprint—Convey uses a mezzanine connector to bring the front side bus (FSB) interface to a large coprocessor board roughly the size of an ATX motherboard. This coprocessor board is housed in a one-unit (1U) chassis that's fused to the top of another 1U chassis containing the host motherboard.

In addition to the machine, Convey designed a selection of accelerator designs to use with it. Some of these implement soft-core floating point vector processors for which Convey has also developed a C and FORTRAN compiler. Others, such as their Smith-Waterman sequence alignment accelerator design, include an easy-to-use interface library. This makes the HC-1's FPGAs accessible to programmers who lack the expertise or patience to design their own FPGA-based coprocessors in hardware description language. However, realizing that the HC-1 appeals to customers who would like to do this, Convey offers support and tools accordingly.

Here, I examine the HC-1, emphasizing its system architecture, performance, ease of programming, and flexibility.

System Overview

The HC-1's host consists of a dual-socket Intel server motherboard, an Intel 5400 memory-controller hub chipset, 24 Gbytes of RAM, 1,066 MHz FSB, and a 2.13 GHz Intel Xeon 5138—a dual-core, low-voltage processor (the 65-nanometer Intel Core architecture released in 2006). Newer Intel Xeons based on the Nehalem or later architectures can't be used in an HC-1-like system until Convey completes the Quick Path Interconnect interface for their coprocessor board. The HC-1 host runs a 64-bit 2.6.18 Linux kernel with a modified virtual memory system to accommodate memory coherency for the coprocessor board.

Top-Level Design

Figure 1 shows the coprocessor board's design. There are four user-programmable Virtex-5 LX 330s, which Convey calls the *application engines* (AEs). Convey refers to a particular configuration of these field-programmable gate arrays (FPGAs) as a "personality." The four AEs each connect to eight memory controllers through a full crossbar. Each memory controller is implemented on its own FPGA and is connected to two Convey-designed scatter-gather dual inline memory modules (SG-DIMMs) containing 64 banks each and an integrated Stratix-2 FPGA. The AEs themselves are interconnected in a ring configuration with 668 Mbytes/s, full duplex links for AE-to-AE communication. These links can be useful for multi-FPGA applications.

Memory Interleave Modes

Each AE has a 2.5 Gbyte/s link to each memory controller, and each SG-DIMM has a 5 Gbyte/s link to its corresponding memory controller. As such, the effective memory bandwidth of the AEs is dependent on their memory access pattern to the eight memory controllers and their two SG-DIMMs. Each AE can achieve a theoretical peak bandwidth of 20 Gbyte/s when striding across eight different memory controllers, but this bandwidth would drop if two other AEs attempt to read from the same set of SG-DIMMs because this would saturate the 5 Gbytes/s DIMM memory controller links.

Because each memory address maps only to one SG-DIMM (and

1521-9615/10/$26.00 © 2010 IEEE

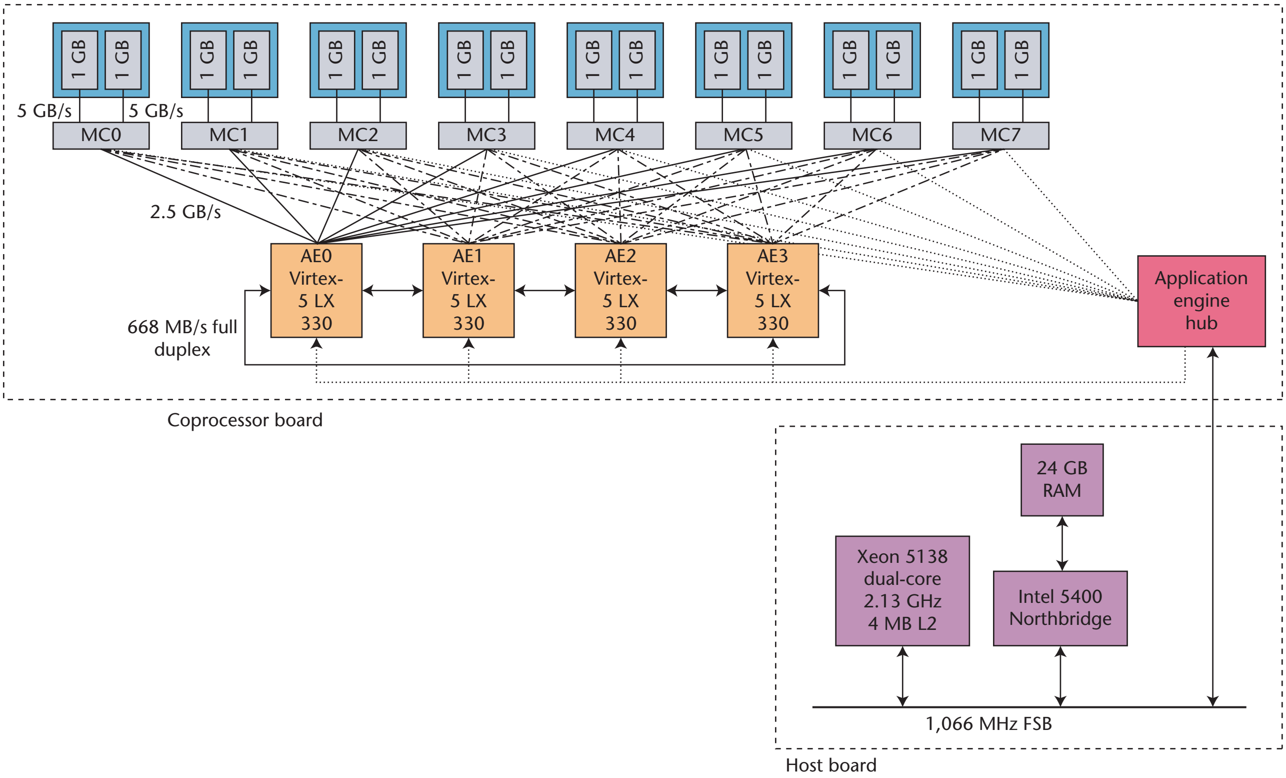


Figure 1. The HC-1 coprocessor board. Four application engines connect to eight memory controllers through a full crossbar. Each memory controller is implemented on its own field-programmable gate array.

its corresponding memory controller), Convey's goal when designing its memory system was to maximize the likelihood that an arbitrary set of unique memory references would be uniformly distributed across all 16 SG-DIMMs and eight memory controllers. Convey provides two user-selectable memory mapping modes to partition the coprocessor's virtual address space among the SG-DIMMs:

- *Binary interleave*, which maps bit-fields of the memory address to a particular controller, DIMM, and bank, and
- *31-31 interleave*, a modulo 31 mapping optimized for constant memory strides (strides lengths that are a power-of-two are guaranteed to hit all 16 SG-DIMMs for any sequence of 16 consecutive references).

The memory banks are divided into 32 groups of 32 banks each. In 31-31 interleave, one group isn't used, and one bank within each of the remaining groups isn't used. Because the number of groups and banks per group is a prime number, this reduces the likelihood of strides aliasing to the same SG-DIMM. Selecting the 31-31 interleave comes at a cost of approximately 1 Gbyte of addressable memory space (6 percent) and a 6 percent reduction in peak memory bandwidth.

Coprocessor Memory Coherency

The coprocessor memory is cache coherent with the host memory and is implemented using the snoopy coherence mechanism built into the Intel FSB protocol. This essentially creates a common virtual address space that both the host and coprocessor share.

In the coherence protocol, both the host and the coprocessor possess copies of the global memory space. Each block of memory addresses in both the host memory and coprocessor memory are marked as *exclusive*, *shared*, or *invalid*. A write by the host to an address block will change its status to *exclusive* and invalidate the block on the coprocessor (indicating that it's out-of-date). If one of the application engines on the coprocessor reads from this block, an updated copy of the block's memory contents is sent to the coprocessor memory, and the memory block changes to *shared* in both the host and coprocessor memory. The coherence mechanism is transparent to the user and removes the need for explicit direct memory access (DMA) transactions, which coprocessors based on peripheral component interconnect (PCI) require.

Host Interface

The coprocessor board contains two non-user programmable FPGAs that together form the application engine hub (AEH). One FPGA serves as the physical interface between the coprocessor board and the FSB, and its logic monitors the FSB to maintain the snoopy memory coherence protocol and manages the coprocessor memory's page table. This FPGA is actually mounted to the mezzanine connector.

The second AEH FPGA contains the *scalar processor*, a soft-core processor that implements the base Convey instruction set. The scalar processor is a substantial architecture, including a cache and features such as multiple issue out-of-order execution, branch predication, register renaming, and sliding register windows.

Convey develops and licenses its own set of personalities but also allows users to develop their own using the personality development kit (PDK).

The scalar processor is the mechanism by which the host invokes computations on the AEs. In Convey's programming model, the AEs act as coprocessors to the scalar processor, while the scalar processor acts as a coprocessor for the host CPU. To facilitate this, the binary executable file on the host (Intel processor) contains integrated scalar processor code (using a "fat binary" linker format), which is transferred to and executed on the scalar processor when the host code calls a scalar processor routine through one of Convey's runtime library calls (a similar mechanism is employed on Nvidia GPUs). The scalar processor code can contain instructions that are dispatched and executed (that is, offloaded) onto the AEs.

Code for the scalar processor can be generated by one of Convey's compilers or handwritten in assembly language. After compilation and assembly, the scalar processor code is linked into the executable in the `ctext` linker section. Upon execution, the host code can invoke scalar processor routines using the synchronous and asynchronous `copcall` API functions. The host CPU can also use this mechanism to send parameters to and receive status information from the AEs.

The scalar processor is connected to each AE via a point-to-point link, and uses this link to dispatch instructions to the AEs that aren't entirely implemented on the scalar processor. Instruction examples include

- move instructions for exchanging data between the scalar processor and AEs; and
- custom AE instructions, which consist of 32 unimplemented instructions that can be used to invoke user-defined AE behaviors.

Through the AE's dispatch interface, AE logic can also trigger exceptions and implement memory synchronization behaviors.

Personalities

Convey develops and licenses its own set of personalities but also allows users to develop their own using the personality development kit (PDK). Convey has established a global numeric identifier system for personalities and maintains a publicly accessible registration database for these identifiers, evidentially in the hope of fostering a marketplace for custom personalities.

Convey's "stock" personalities are individually licensed and are each designed for specific application types. Currently, the set includes a single-precision vector personality, double-precision vector personality, financial analytics personality, and Smith-Waterman personality.

The two vector personalities act as vector coprocessors for the scalar processor and are targets for Convey's vectorizing compiler. When using these personalities, each AE implements eight floating point multiply-adder pipelines and eight load/store units (for a total of 32 logically combined across four AEs).

The financial analytics personality is a double-precision personality that adds additional vector instructions, transcendental functions, probability distribution functions, and various random number generators designed for high-performance Monte Carlo simulation. In addition to the compiler, the vector and financial personalities also have robust debuggers, simulators, and performance analyzers. The single-precision vector personality also has the Convey math library (CML), a corresponding, hand-optimized basic linear algebra subroutines (BLAS) implementation. The Smith-Waterman personality is a parameterized, scalable processing element and is built around the Convey Sequence Library, a customized API.

As mentioned earlier, users who wish to develop their own personalities with HDL-based design must license the PDK, which includes design flows and robust system models that support hardware/software co-simulation.

Convey Instruction Set Architecture

Convey developed its own entirely new instruction set architecture from the ground up. The Convey ISA includes a scalar instruction set that's common to all personalities, including custom ones. All scalar instructions are executed on the scalar processor. The scalar instruction set includes instructions for program control (branches), context saves, scalar arithmetic, load/store, and move instructions for the set of *A* and *S* registers (which reside on the scalar processor). The instruction set also includes a large set of vector instructions that are offloaded to the vector personalities (if present).

The Convey ISA features a virtualized register set. The three register sets (scalar, address, and vector) are of arbitrary size because the hardware dynamically maps user registers to physical registers at runtime. This also applies to each vector register's length and the vector stride for the load/store units, both of which can be dynamically changed by the software at runtime if you change the vector registers' length and stride values.

Table 1. Level 3 BLAS Performance, Nehalem Xeon vs. Tesla vs. HC-1

	Single-precision general matrix–matrix multiply (Gflops/s)		
Matrix order	Dual Xeon 5520 MKL w/Intel C compiler 11.1	NVIDIA Tesla S1070 CUBLAS w/Nvidia C compiler 3.1	HC-1 coprocessor CML 1.2.2 w/Convey C Compiler 2.0.0
8,000	110	347	75
10,000	126	348	76
12,000	136	355	76
14,000	140	363	75
16,000	140	378	76
Average	**130**	**358**	**76**

Peak Floating Point Performance

The HC-1's hardware, compiler, and only one of their vector personalities cost approximately 10 times that of a state-of-the-art dual-socket Xeon-based Dell PowerEdge server, or that of a rack-mounted four-GPU Nvidia Tesla server, despite the fact that each of these systems have approximately the same physical footprint. In my lab, my research group has one of each of these systems, which allows for convenient cost-performance comparisons. We ran a series of simple tests to pit our HC-1 against our Dell PowerEdge R710 with dual Xeon 5520 processors, which use the Nehalem architecture and were Intel's state-of-the-art server processor architecture from March 2009 to March 2010. This product was recently superseded by the Xeon 5600-series (Westmere), which is a technology-scaled version of the same architecture. This PowerEdge server is attached to our Nvidia Tesla S1070, containing four Tesla GPUs. The Tesla has also recently been superceded by the Fermi.

We designed a series of tests to measure both raw performance and ease of programming. To estimate the systems' peak floating point performance, we targeted dense single-precision general matrix–matrix multiply (SGEMM) from the level-three BLAS library, because an equivalent platform-optimized implementation of this function is available in the Intel's math kernel library (MKL), Nvidia's compute unified basic linear algebra subprograms (Cublas) library (http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/CUBLAS_Library_3.0.pdf), and the Convey math library (CML). Specially, we tested the operation was **C** = **AB** where **A** and **B** are square matrices.

Table 1 shows the effective Gflops/s for each test, where we measure Gflops as:

$$\frac{time}{2 \times order^3}$$

on an unloaded system. The time includes I/O time for the Tesla and HC-1. We ran each test only once rather than averaging over a large set of runs because these results are intended to be illustrative only.

The Intel results reflect the use of all eight processor cores (two sockets each with four-core CPUs) and a SSE4.2 vector unit for each core. The Nehalem system achieved an average throughput of approximately 130 Gflops/s. This is reasonable, because each of the eight cores has an SSE unit that can perform four multiplies and four adds per cycle at 2.26 GHz, giving a theoretical peak of 145 Gflops/s without considering any effects of the memory system. The GPU-based system showed an average throughput of approximately 358 Gflops/s. The HC-1 achieved an average throughput of 76 Gflops/s.

These performance metrics don't look encouraging for the HC-1, especially given that both the Nehalem and the Tesla GT200 GPUs are already previous-generation architectures, while the HC-1 is still current generation. Convey admits that the peak throughput of the HC-1 is "nearly 80 Gflops/s" based on its coprocessor memory bandwidth, so these results indicate that the HC-1 is more capable of achieving throughput closer to its peak than the Xeon.

However, these performance results are given by heavily hand-optimized BLAS routines. In our next set of performance tests, we explored the performance given of the Intel and Convey vectorizing compilers when given non-optimized high-level code.

Power Consumption

The tested machines are powered by a power distribution unit that is capable of measuring the total current being drawn with a granularity of one amp. Although this is obviously an inaccurate method for testing power consumption, it allows us to make rough approximations.

While running the SGEMM tests, the PowerEdge alone drew 3 amps, indicating a 360-watt consumption, and thus achieved 360 Mflops/watt. During the Tesla SGEMM test, the PowerEdge and Tesla together drew 6 amps (720 watts) and thus achieved approximately 500 Mflops/watt. During the HC-1 SGEMM test, the HC-1 alone drew 6 amps (720 watts) and thus achieved approximately 100 Mflops/watt. These results indicate that the Tesla actually wins in flops per watt and the HC-1 comes in third, which runs contrary to public popular opinion regarding the power efficiency of GPUs versus FPGAs. This indicates that there might be inefficiencies in the HC-1's system design.

Convey Compiler

Convey has developed a vectorizing C and FORTRAN compiler based on Open64 (www.open64.net) that can target the scalar processor coupled with one of its vector personalities. To use one of these personalities, users simply insert Convey pragmas—notably, `#pragma cny begin_coproc` and `#pragma cny end_coproc`—into C or FORTRAN code to denote which sections of code to execute on the coprocessor (other pragmas are also available to give programmer hints to the compiler). The Convey vectorizing compiler compiles these sections targeting the Convey ISA and executes them on the scalar processor, which offloads any vector instructions to the appropriate personality on the AEs (which are automatically configured with the appropriate personality at runtime).

To determine how well the Intel and Convey vector architectures lend themselves to automatic compiler vectorization of naïvely written, (mostly) architecture-oblivious, and (mostly) non-hand-optimized code, we wrote a simple three-loop implementation of matrix multiply, compiled this code with the maximum optimization settings with both the Intel and Convey compilers, and then compared the resulting performance on their corresponding platforms with that of the their corresponding BLAS performance.

For the Intel version, we parallelized the outermost loop with OpenMP (using the `parallel for` directive), which distributed the loop across 16 threads during runtime, fully utilizing the eight cores with two-way symmetric multithreading. Also, from prior experience we know that the Intel load/store units perform best with vector strides of one—that is, floating point values can only be loaded directly into the streaming single-instruction multiple-data extensions (SSE) extended multimedia (XMM) registers from consecutive memory locations. Because transposing one of the matrices is a minor change to the code, our Intel implementation includes this simple optimization (that is, transposing matrix **B**, making matrix **A** row-major and matrix **B** column-major). This optimization doesn't effect HC-1 performance because, as I discuss later, it's indifferent to vector stride length. As such, in our tests, the input matrices for the HC-1 implementation are both row-major.

The Convey compiler is still relatively early in its development, and—according to the compiler manual—the high-level code must be written in specific ways to ensure vectorization. The compiler also provides detailed feedback to the programmer, reporting exactly which loops are vectorized and what type and number of vector instructions are used in the generated code.

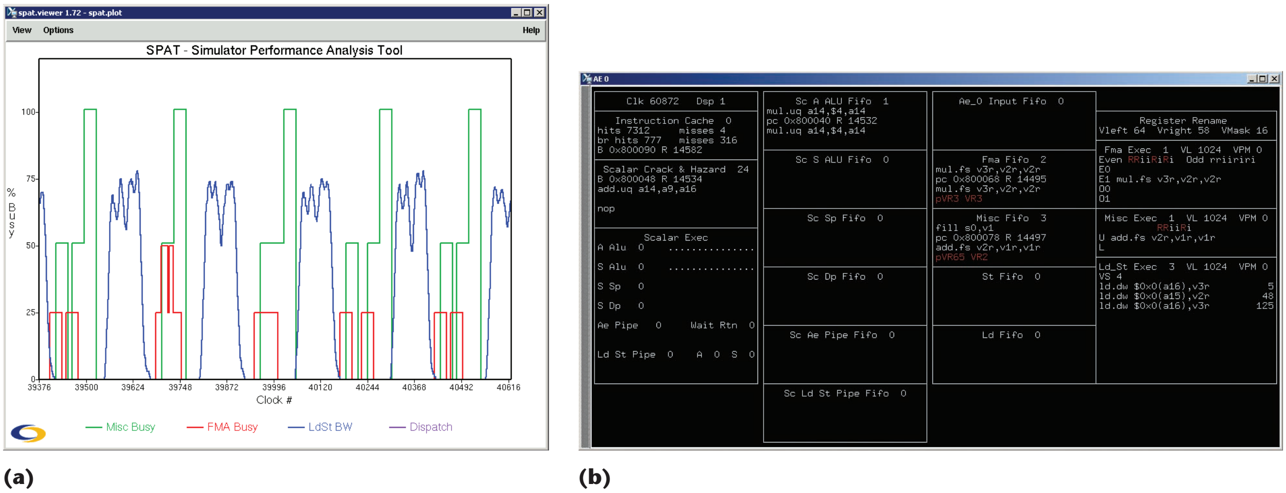
For the Convey compiler to vectorize our code, we had to apply a minor transformation, using one loop nest to initialize the result matrix to zero, followed by a second loop nest that performs the matrix multiply by computing the inner products and adding each into the entries of the result matrix. To be fair, we also tried this optimization to the Intel code but it resulted in a slight slowdown so we didn't use it for the Intel tests. In the HC-1 C code, both loops together are marked for coprocessor execution:

```
#pragma cny array(cm[size]
  [size])
#pragma cny array(am[size]
  [size])
#pragma cny array(bm[size]
  [size])

#pragma cny begin_coproc
       for (i=0;i<size;i++) {
        for (j=0;j<size;j++) {
        cm[i][j]=0.0;
        }
       }

       for (i=0;i<size;i++) {
        for (j=0;j<size;j++) {
        for (k=0;k<size;k++) {
        cm[i][j] += am[i]
           [k]*bm[k][j];
        }
        }
       }
#pragma cny end_coproc
```

CUDA requires that programmers explicitly parallelize code into threads and blocks, making it impossible to



```
   Clk 60872   Dsp 1
   Instruction Cache  0
hits 7312     misses 4
br hits 777   misses 316
B 0x800090 R 14582

 Scalar Crack & Hazard  24
B 0x800048 R 14534
add.uq a14,a9,a16

nop

        Scalar Exec
A Alu  0    ..............
S Alu  0    ..............
S Sp   0
S Dp   0
Ae Pipe   0     Wait Rtn  0
Ld St Pipe  0    A  0  S  0

     Sc A ALU Fifo  1
mul.uq a14,$4,a14
pc 0x800040 R 14532
mul.uq a14,$4,a14

     Sc S ALU Fifo  0

        Sc Sp Fifo  0

        Sc Dp Fifo  0

     Sc Ae Pipe Fifo  0

     Sc Ld St Pipe Fifo  0

     Ae_0 Input Fifo  0

          Fma Fifo  2
mul.fs v3r,v2r,v2r
pc 0x800068 R 14495
mul.fs v3r,v2r,v2r
pVR3 VR3

         Misc Fifo  3
fill s0,v1
pc 0x800078 R 14497
add.fs v2r,v1r,v1r
pVR65 VR2

          St Fifo  0

          Ld Fifo  0

      Register Rename
Vleft 64  Vright 58  VMask 16

 Fma Exec  1  VL 1024  VPM 0
Even RRiiRiRi  Odd rriiriri
E0
E1 mul.fs v3r,v2r,v2r
O0
O1

 Misc Exec  1  VL 1024  VPM 0
          RRiiRi
U add.fs v2r,v1r,v1r
L

Ld_St Exec  3  VL 1024  VPM 0
VS 4
ld.dw $0x0(a16),v3r          5
ld.dw $0x0(a15),v2r         48
ld.dw $0x0(a16),v3r        125
```

Figure 2. Screen examples from Spat, Convey's toolset for assisting programmers in tuning their code. (a) A plot depicting the utilization of the processor subsystems versus clock cycle during a loop execution. (b) An interactive trace of the instruction stream, showing the processor's internal state during a specific clock cycle.

write architecture-oblivious code. However, Nvidia's CUDA software development kit (SDK) includes a relatively simple matrix multiply that parallelizes the matrix multiply using a simple blocking technique. We measured this implementation's performance (not allowing a "kernel warmup" and including the host-GPU I/O time, which the code doesn't incorporate in its own instrumentation) and included these results for discussion.

Table 2 shows the test results. The Intel implementation achieves 8 to 10 percent of its MKL performance using the naïvely written code, while the HC-1 outperforms the Intel implementation and achieves 20 to 24 percent of its CML performance. These results indicate the HC-1 has more potential for extracting performance and automatically parallelizing floating point linear algebra kernels that aren't mapped directly into BLAS routines. The CUDA SDK code achieves 48 to 54 percent of its peak performance but (as noted earlier) this code is explicitly parallelized by Nvidia, unlike the Intel and Convey code, so it's not a fair comparison.

Table 2. Compiler effectiveness for optimizing naïve code.

Simple three-loop matrix multiplication (Gflops/s)			
Xeon 5520 C code SSE4.2/ OMP w/ICC 11.1 (row major × row major)	**Xeon 5520 C code SSE4.2/ OMP ICC 11.1 (row major × column major)**	**Nvidia CUDA SDK `matrixMul_` routine**	**HC-1 C code single-precision vector personality**
1 (<1 % peak)	11 (10% peak)	189 (54% peak)	15 (21% peak)
1 (<1 % peak)	11 (9% peak)	190 (54% peak)	15 (20% peak)
1 (<1 % peak)	11 (8% peak)	189 (53% peak)	16 (21% peak)
1 (<1 % peak)	11 (8% peak)	184 (51% peak)	16 (21% peak)
1 (<1 % peak)	10 (8% peak)	180 (48% peak)	15 (24% peak)

Convey Simulator and Performance Analysis Tool

To help developers get the most performance out of their code, Convey also offers a simulator and corresponding performance analysis tool called "Spat" that graphically plots how various aspects of the code map to the architecture and can assist in code tuning.

As Figure 2a shows, the information is presented as a plot of clock cycle vs. usage of various architectural features. The tool can also graphically depict detailed state information for various units within the scalar and vector processors (see Figure 2b). This information lets users step across clock cycles and witness how the system executes various instructions. The figure's plots originate from my handwritten assembly-language implementation of the matrix-multiplier, with which I attempted to outperform the compiler-generated implementation. After approximately one day's effort, I was able to match only the compiled code's performance, which speaks well of the Convey compiler.

Memory-Intensive Applications

HC-1's real strength is its memory-centric applications, or applications that require nonconsecutive memory access strides.[1] Our experimental results are evidence of this; but to demonstrate, I offer results from a benchmark designed to stress memory systems.

The Stride3 benchmark is part of Lawrence Livermore National Lab's Sequoia benchmark suite (https://asc.llnl.gov/sequoia/benchmarks) and uses a series of sequential kernels that perform double-precision floating

Table 3. Stride3C benchmark for Xeon vs. HC-1 coprocessors.

Stride	Stride3C benchmark (Gflops/s)	
	Xeon 5520 single-thread	HC-1 w/double-precision personality
256	0.06	4.3
512	0.05	4.3
1024	0.05	4.3
961	0.04	0.1 (lowest)
992	0.06	0.3 (2nd lowest)
8	0.07	4.4 (highest)
Overall average	**0.05**	**4.1**

Table 4. Smith-Waterman performance on Xeon vs. HC-1 coprocessors searching a protein database with an 80-character query.

Database size (amino acids)	Xeon 5520 multithread	HC-1 w/AESW personality	HC-1 speedup
8 × 10^7	3,073 ms	353 ms	8.7
4 × 10^8	14,763 ms	1,773 ms	8.3
8 × 10^8	29.754 ms	3,589 ms	8.3

point operations using values from two matrices at various stride distances. In our particular test, we set the matrix sizes such that they're too large to fit in the Xeon's cache.

Table 3 shows the results: HC-1 easily outperformed the Xeon 5520 (the Stride3 benchmark is single-threaded, which might be a disadvantage for the Xeon).

Convey has also recently developed a Smith-Waterman personality for high-throughput genomic database searches.[2] The Smith-Waterman personality derives its performance from the FPGA's ability to perform comparisons on sub-byte data units (that is, 2 bits for nucleotide and 5 bits for protein), which allows it to pack more operations per memory access than is possible with fixed-architecture CPUs and GPUs. However, the current version of the Smith-Waterman personality seems to use a simplistic variant of the Smith-Waterman algorithm in that it considers match, mismatch, insert, and delete penalties rather than more aggressive implementations with more complex cost models that allow different costs for opening gaps and extending gaps.

To approximate the Smith-Waterman personality's performance relative to a well-known software implementation, we ran a series of performance tests of the personality against the University of Virginia's SSearch35 version 35.04 (http://fasta.bioch.virginia.edu/fasta_www2/fasta_list2.shtml), a highly optimized multithreaded SSE-based Smith-Waterman implementation. SSearch35 uses the slightly more complex cost model described earlier, so these implementations use a slightly different scoring model. However, both are based on the traditional dynamic programming approach to compute optimal alignment scores and both use the Blosum substitution matrix. As before, the time values include I/O time between the host and coprocessor.

Table 4 shows the results. For the three sample database sizes, the HC-1 performs just over eight times better than the Xeon. Although these are encouraging results, it's not clear if FPGAs will continue to maintain this lead as CPUs architectures continue to scale.

Developing Custom Personalities

According to Convey, its target customers are primarily interested in using predesigned personalities. We purchased the system primarily as a platform for testing our research group's customized accelerator designs. We chose the HC-1 because it had four large Virtex-5 330 LX FPGAa and because its memory-coherent host interface eliminates the extra engineering time required for DMA-based interfacing. Because I've worked with PCI-based FPGA coprocessors, working with the HC-1's memory model is much easier than having to coordinate with the host to set up explicit DMA transfers, which greatly simplifies host interfacing.

Designing custom personalities requires the use of Convey's PDK, which contains

- a set of makefiles to support simulation and synthesis design flows,
- a set of Verilog support and interface files,
- a set of simulation models for all of the coprocessor board's nonprogrammable components (such as the memory controllers and memory modules), and
- a programming-language interface (PLI) to let the host code interface with a behavioral HDL simulator such as Modelsim.

The kit's simulation framework is easy to use and allows users switch between a simulated coprocessor and an actual coprocessor by changing only one environment variable.

Developing with the PDK involves working within a Convey-supplied wrapper that gives the user logic access to instruction dispatches from the scalar processor, access to all eight memory controllers, access to the coprocessor's management processor for debugging support, and access to the AE-to-AE links. However, the wrapper requires fairly substantial resource overheads: 184 out of the 576 18-Kbytes block random access memory (BRAMS) and approximately 10 percent of each FPGA's slices. Convey supplies a fixed 150-MHz clock to each FPGA's user logic.

Users who develop custom personalities must also develop a corresponding API. That is, although Convey's compiler, debugger, and analysis tools can be used with their vector personalities, there's no compiler support—or tool support at all—for custom personalities. For example, if I were to develop a custom personality to accelerate molecular dynamics, I'd also need to develop a corresponding software library that would let users execute the accelerated kernels on the AEs from their own software. This library would be responsible for interfacing with the scalar processor and AEs through the `copcall` and custom instruction mechanism.

The HC-1's FPGA-based coprocessor doesn't compete in peak floating point performance with Nvidia GPUs or even Intel Xeon processors, but its vector personality architecture is more flexible and allows its compiler to extract greater performance from generalized high-level code than Intel's compiler. This is partly because the HC-1's vector personalities and coprocessor memory system are capable of single-instruction loads of vectors that are stored in nonconsecutive memory locations, allowing it to achieve a higher ratio of its peak memory bandwidth relative to the Xeon and Nvidia GPUs for "strided" data. This is perhaps its greatest advantage over the Xeon and Nvidia architectures. In other words, both the Xeon and Tesla lose a substantial amount of memory system performance when loading vectors whose elements are not aligned properly and not stored in consecutive memory locations (Nvidia refers to such behavior as "non-coalesced" loads or stores). In addition, the FPGAs' reconfigurable nature lets the HC-1 perform operations on nonstandard memory units and arbitrary precision values, making it more efficient for applications such as sequence alignment. CiSE

Acknowledgments

This material is based on work supported by the US National Science Foundation under grant nos. CCF-0844951 and CCF-0915608. Thanks to Glen Edwards, Chris Parrott, Mark Kelly, John Leidel, Kirby Collins, and Tom Murphy of Convey Computer for answering my questions, for providing prerelease versions of the Convey compiler, and for providing free 30-day licenses for the double-precision and Smith-Waterman personalities.

References

1. J. Leidel, "Design Philosophies for Memory-Centric Instruction Set Architectures," presentation, Symp. Application Accelerators in High Performance Computing (SAAHPC'10), 2010; http://saahpc.ncsa.illinois.edu/presentations/day1/session4/presentation_Leidel.pdf.
2. Convey Computer, "Convey Computer Announces Record-Breaking Smith-Waterman Acceleration of 172x," press release, 24 May 2010; www.conveycomputer.com/Resources/Convey_Announces_Record_Breaking_Smith_Waterman_Acceleration.pdf.

Jason D. Bakos is an assistant professor in the Department of Computer Science and Engineering at the University of South Carolina. His research interests include computer architecture, very large-scale integration (VLSI) design, and high-performance heterogeneous computing. Bakos has a PhD in computer science from the University of Pittsburgh. He is a member of IEEE and the ACM. Contact him at jbakos@sc.edu.