Convey HC-1 Hybrid Core Computer – The Potential of FPGAs in Numerical Simulation

Werner Augustin
Vincent Heuveline
Jan-Philipp Weiss

No. 2010-07

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)

$\frac{\partial u}{\partial t} - \nu\Delta u + (u \cdot \nabla)u + \frac{1}{\rho}\nabla p$
$\frac{\partial u}{\partial t}$

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)
ISSN 2191–0693
No. 2010-07

Convey HC-1 Hybrid Core Computer – The Potential of FPGAs in Numerical Simulation

Werner Augustin, Jan-Philipp Weiss
Engineering Mathematics and Computing Lab (EMCL)
SRG New Frontiers in High Performance Computing
Exploiting Multicore and Coprocessor Technology
Karlsruhe Institute of Technology, Germany
werner.augustin@kit.edu, jan-philipp.weiss@kit.edu

Vincent Heuveline
Engineering Mathematics and Computing Lab (EMCL)
Karlsruhe Institute of Technology, Germany
vincent.heuveline@kit.edu

***Abstract*—The Convey HC-1 Hybrid Core Computer brings FPGA technologies closer to numerical simulation. It combines two types of processor architectures in a single system. Highly capable FPGAs are closely connected to a host CPU and the accelerator-to-memory bandwidth has remarkable values. Reconfigurability by means of pre-defined application-specific instruction sets called personalities have the appeal of optimized hardware configuration with respect to application characteristics. Moreover, Convey's solution eases the programming effort considerably. In contrast to hardware-centric and time-consuming classical coding of FPGAs, a dual-target compiler interprets pragma-extended C/C++ or Fortran code and produces implementations running on both, host and accelerator. In addition, a global view of host and device memory is provided by means of a cache-coherent shared virtual memory space.**

In this work we analyze Convey's programming paradigm and the associated programming effort, and we present practical results on the HC-1. We consider vectorization strategies for the single and double precision vector personalities and a suite of basic numerical routines. Furthermore, we assess the viability of the Convey HC-1 Hybrid Core Computer for numerical simulation.

***Keywords*-FPGA, Convey HC-1, reconfigurable architectures, high-performance heterogeneous computing, coherent memory system, performance analysis, BLAS**

I. Introduction

Field Programmable Gate Arrays (FPGAs) have their main pillar and standing in the domain of embedded computing. Application-specific designs implemented by hardware description languages (HDL) like VHDL and Verilog [1], [2] make them a perfect fit for specific tasks. From a software-oriented programmer's point of view FPGA's capabilities are hidden behind an alien hardware design development cycle. Although there are some C-to-HDL tools like ROCCC, Impulse C or Handle-C [3] available, viability and translation efficiency for realistic code scenarios still have to be proven.

For several years, FPGAs have not been interesting for numerical simulations due to their limited capabilities and resource requirements for double precision floating point arithmetics. But following Moore's law and with increased FPGA sizes more and more area is becoming available for computing. Moreover, further rates of increase are expected to outpace those of common multicore CPUs. For a general deployment and in particular for numerical simulation FPGAs are very attractive from further points of view: run-time configurability is an interesting topic for applications with several phases of communication and computation and might be considered for adaptive numerical methods. In addition, energy efficiency is a great concern in high performance computing and FPGA technology is a possible solution approach. The main idea of FPGAs is to build one's own parallel fixed-function units according to the special needs of the underlying application.

Currently, numerical simulation adopts all kinds of emerging technologies. In this context, a trend towards heterogeneous platforms has become apparent [4]. Systems accelerated by *graphics processing units* (GPUs) offer unrivaled computing power but often suffer from slow interconnection via PCIe links. The idea to connect FPGAs via socket replacements closer to CPUs is nothing new (cf. technologies from Nallatech, DRC, XtremeData) – but the software concept offered by Convey is revolutionary [5]. A related FPGA platform is Intel's Atom reconfigurable processor – an embedded single board computer based on the Intel Atom E600C processor series which pairs with an Altera FPGA in a single package. Here, both entities communicate via PCIe-x1 links. A former hybrid CPU-FPGA machine was the Cray XD1 [6].

In this work we outline the hardware and software architecture of the Convey HC-1 Hybrid Core Computer. We analyze Convey's programming concept and assess the functionalities and capabilities of Convey's single and double precision vector personalities. Furthermore, we evaluate the viability of the Convey HC-1 Hybrid Core Computer for numerical simulation by means of selected numerical kernels that are well-known building blocks for higher-level numercial schemes, solvers, and applications. Some performance results on the HC-1 can be found in [7]. Our work puts more emphasis on floating point kernels relevant for numerical simulation. Stencil applications on the HC-1 are also considered in [8].

II. Hardware Configuration of the Convey HC-1

The Convey HC-1 Hybrid Core Computer is an example of a heterogeneous computing platform. By its hybrid setup, specific application needs can either be handled by an x86-64 dual-core CPU or by the application-adapted FPGAs. All computational workloads are processed by a 2.13GHz Intel Xeon 5138 dual-core host CPU and by the *application engines* (AE), a set of four Xilinx Virtex 5 LX330 FPGAs. Two more V5LX110 FPGAs implement the host interface – the *application engine hub* (AEH) – for data transfers and control flow exchange between host and device, and eight V5LX155 FPGAs build the eight accelerator's memory controllers. Data transfers are communicated via the CPU's front-side bus (FSB), Intel's aging technology. Across the whole system a cache-coherent shared virtual memory system is provided that allows to access data in the CPU's memory and in the accelerator device memory. However, the system incorporates a ccNUMA system where proper data placement is performance-critical. In our system the host CPU is equipped with 16x667MHz FBDIMMs providing 16 GB of memory and a theoretical bandwidth of 8 GB/s. On the device 16 DDR2 memory channels feed Convey's special 16x667MHz Scatter-Gather-DIMMs with 8 GB device memory and a theoretical peak bandwidth of 80 GB/s. The fundamental advantage of this memory configuration is that non-unit strides have no drawback on the effective bandwidth. All data can be accessed with 8 byte granularity. Functional units on the FPGAs are implemented by logic synthesis by means of Convey's personalities [5]. The single precision vector personality provides a load-store vector architecture with 32 function pipes, each one containing a vector register file and four fused-multiply-add (FMA) vector units for exploiting data parallelism by means of vectorization. Furthermore, out-of-order execution provides a means for instruction-level parallelism. While the clock rate of the FPGAs is undisclosed, the peak GFlop/s rate is expected to be about 80 GFlop/s for single precision and 40 GFlop/s for double precision. The most accented difference of the FPGA accelerator memory subsystem is that there are no caches and no local memory available. All block-RAM on the FPGA is not accessible by the user (unless custom personalities and FPGA designs are created that support this feature). The whole system consumes about 650-850 Watt (depending on the actual workload). A sketch of the HC-1's hardware configuration is shown in Figure 1.

In comparison to other accelerators, the HC-1 offers lower peak performance and lower bandwidth. But in contrast, fast device memory can be configured up to 128 GB in size and is not limited to a few GB as on GPUs. Furthermore, Convey's technology and its future development will possibly allow fast cluster-like connection between several FPGA-based entities. In the latest Convey product, the HC-1ex, Xilinx Virtex 6 FPGAs and an Intel 5408 Xeon quad-core processor provide the same functionality with improved capabilities.

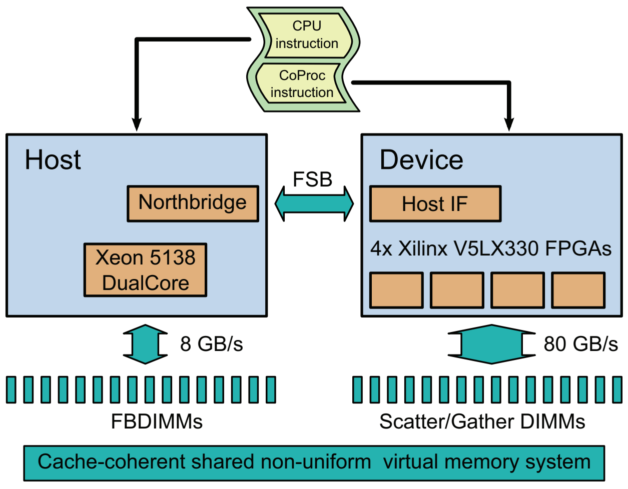

Fig. 1. Hardware Configuration of the Convey HC-1

III. Convey HC-1's Software Environment

Programming of FPGAs by means of HDL is a time-consuming and non-intuitive effort – and so FPGAs have been out of reach for many domain experts. With Convey's solution, FPGAs are now a viable alternative from a programming point of view. On the Convey HC-1, the CPU's capabilities are extended by application-specific instructions for the FPGAs [5]. However, the asymmetric computing platform is hidden by Convey's unified software interface. A single code base (C, C++ or Fortran) can be enhanced with pragmas to advise the compiler how to treat computations and data. In particular, code regions that should be executed on the FPGAs can be identified by inserting `#pragma cny begin_coproc / end_coproc`. Another possibility is to compile whole subroutines for the accelerator, or even to use compiler's capability for automatic identification of parts, that shall be offloaded to the FPGAs. Typical code sections are those suitable for vectorization, especially long loops. The compiler then produces a dual target executable, i.e. the code can be executed on both the CPU host (e.g. if the coprocessor is not available) and on the FPGA accelerator. Hence, a portable solution is created that can run on any x86 system. There are some restrictions to coprocessor code: it is not possible to do I/O, to make system calls or to call non-coprocessor functions. A compiler report gives details on the vectorization procedure. Specific pragmas give further hints for compiler-based optimizations.

Both memories on the host CPU and on the FPGA accelerator are combined into a common address space and data types are common across both entities. In order to prevent NUMA effects, placement of data can be controlled by pragmas. Data allocated in the CPU memory can be transferred to the accelerator with `#pragma cny migrate_coproc`. Dynamic and static memory on the FPGA device can be allocated directly via `#pragma cny_cp_malloc` and `#pragma cny coproc_mem` respectively. In case of memory migration whole memory pages are transferred. In order to avoid multiple transfers, several data objects should be grouped into larger structs.

The actual configuration of the FPGAs is represented by means of application-specific instruction sets called personalities. These personalities augment the host's x86-64 instruction set. This features allows adaptation and optimization of the hardware with respect to the specific needs of the underlying algorithms. The user only has to treat an integrated instruction set controlled by pragmas and compiler settings. Convey offers a set of pre-defined personalities for single and double precision floating point arithmetics that turn the FPGAs into a soft-core vector processor. Furthermore, personalities for financial analytics and for proteomics are available. Currently, a finite difference personality for stencil computations is under development. The choice for a requested personality is specified at compile time by setting compiler flags. With Convey's personality development kit custom personalities can be developed by following the typical FPGA hardware design tool chain (requiring considerable effort and additional knowledge). Convey's *Software Performance Analysis Tool* (SPAT) gives insight into the system's actual runtime behavior and feedback on possible optimizations. The Convey Math Library (CML) provides tuned basic mathematical kernels. For our experiments we used the Convey64 Compiler Suite, Version 2.0.0.

IV. The Potential of FPGAs

FPGAs have been considered to be non-optimal for floating point number crunching. But FPGAs show particular benefits for specific workloads like processing complex mathematical expressions (logs, exponentials, transcendentals), performing bit operations (shifts, manipulations), and performing sort operations (string comparison, pattern recognition). Further benefits can be achieved for variable bit length of data types with reduced or increased precision, or for treating non-standard number formats (e.g. decimal representation). The latter points are exploited within Convey's personalities for financial applications and proteomics. Recently, Convey reported a remarkable speedup of 172 for the Smith-Waterman algorithm [9].

Pure floating point-based algorithms in numerical simulation are often limited by bandwidth constraints and low arithmetic intensity (ratio of flop per byte). The theoretical peak bandwidth of 80 GB/s on the Convey FPGA device goes along with a specific appeal in this context. However, memory accesses on the device are not cached. Hence, particular benefits are expected for kernels with limited data reuse like vector updates (SAXPY/DAXPY), scalar products (SDOT/DDOT) and sparse matrix-vector multiplications (SpMV). Convey's special Scatter-Gather DIMMs are well adapted to applications with irregular data access patterns where CPUs and GPUs typically show tremendous performance breakdowns.

V. Performance Evaluation

In order to assess the performance potential of Convey's FPGA platform for floating point-based computations in numerical simulation we analyze some basic numerical kernels and their performance behavior. In particular, we consider library-based kernels provided by the CML and hand-written, optimized kernels. By comparing both results we draw some conclusion on the capability of Convey's compiler. In all cases, vectorization of the code and NUMA-aware placement of data is crucial for performance. Without vectorization there is a dramatic performance loss since scalar code for the accelerator is executed on the slow *application engine hub* (AEH) that builds the interface between host and accelerator device. If data is not located in the accelerator memory but is accessed in the host memory over the FSB, bandwidth and hence performance also drop considerably.

For our numerical experiments we consider some basic building blocks for high-level solvers, namely vector updates $z = ax + y$ (SAXPY/DAXPY in single and double precision), vector product $\alpha = x \cdot y$ (SDOT/DDOT), dense matrix-vector multiplication $y = Ax$ (SGEMV/DGEMV), dense matrix-matrix multiplication $C = AB$ (DGEMM/SGEMM), sparse matrix vector multiplication (SpMV), and stencil operations.

VI. Vectorization and Optimization of Code

In order to exploit the full capabilities of the FPGA accelerator specific measures are necessary for code creation, for organizing data accesses, and to support the compiler for vectorizing code. Due to its nature as a low frequency, highly parallel vector architecture, performance on the Convey HC-1 heavily depends on the ability of the compiler to vectorize the code. One of the examples where this did not work out-of-the-box is dense matrix-vector multiplication SGEMV. The code snippet in Figure 2 shows a straightforward implementation. Here, the pragma cny no_loop_dep gives a hint to the compiler for vectorization that there are no data dependencies in the corresponding arrays.

```
void gemv(int length, float A[], float x[],
          float y[]){
  for( int i = 0; i < length; i++) {
    float sum = 0;
#pragma cny no_loop_dep(A, x, y)
    for( int j = 0; j < length; j++)
      sum += A[i*length+j] * x[j];
    y[i] = sum;
  }
}
```

Fig. 2. Straightforward implementation of dense matrix-vector multiplication (SGEMV)

Although the compiler claims to vectorize the inner loop, performance is only approx. 2 GFlop/s and by a factor of 7 below the performance of the CML math library version. The coprocessor instruction set supports vector reduction operations, but these seem to have a pretty high startup latency. The outer loop is not unrolled. Attempts to do that manually improved the performance somewhat, but introduced new performance degradations for certain vector lengths.

The solution lies in exploiting Convey's scatter-gather memory which allows for fast strided memory reads and therefore allows to change the loop ordering (see Figure 3). This gives considerably better results; performance improvements by loop reordering are detailed in Figure 4. For the reordered loops we consider three different memory allocation scenarios: dynamic memory allocated on the host and migrated with

Convey's pragma `cny migrate_coproc`, dynamic memory allocated on the device, and static memory allocated on the host and migrated to the device with the pragma mentioned above. Performance increases with vector length but has some oscillations. These results even outperform the CML CBLAS library implementation from Convey (cf. Figure 14).

```
void optimized_gemv(int length, float A[],
                    float x[], float y[]){
  for( int i = 0; i < length; i++ )
    y[i] = 0.0;
  for( int j = 0; j < length; j++ )
#pragma cny no_loop_dep(A, x, y)
    for( int i = 0; i < length; i++ )
      y[i] += A[i*length+j] * x[j];
}
```

Fig. 3. Dense matrix-vector multiplication (SGEMV) optimized by loop reordering

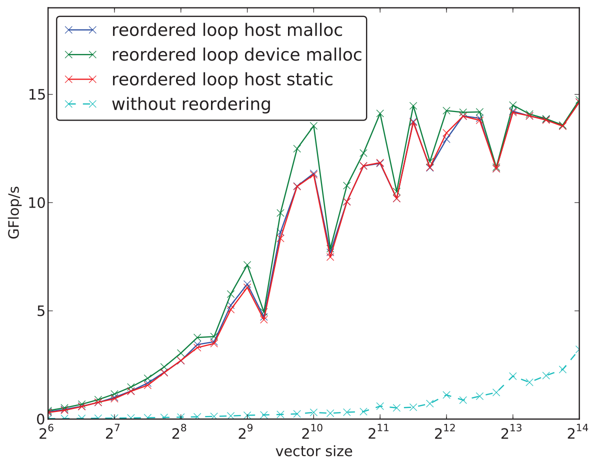

Fig. 4. Performance results and optimization for single precision dense matrix-vector multiplication (SGEMV) with and without loop reordering and for different memory allocation schemes; cf. Fig. 2 and Fig. 3

VII. Performance Results and Analysis

A. Device Memory Bandwidth for Different Access Patterns

Performance of numerical kernels is often influenced by the corresponding memory bandwidth for loading and storing data. For our memory bandwidth measurements we use the following memory access patterns that are characteristic for diverse kernels:

Sequential Load (SeLo):	$d[i] = s[i]$
Sequential Load Indexed (SeLoI):	$d[i] = s[seq[i]]$
Scattered Load Indexed (ScaLoI):	$d[i] = s[rnd[i]]$
Sequential Write (SeWr):	$d[i] = s[i]$
Sequential Write Indexed (SeWrI):	$d[seq[i]] = s[i]$
Scattered Write Indexed (ScaWrI):	$d[rnd[i]] = s[i]$

Here, $seq[i] = i$, $i = 1, \ldots, N$, is a sequential but indirect addressing and $rnd[i]$ is an indirect addressing by an arbitrary permutation of $[1, \ldots, N]$. Results are presented in Figure 5 and should be seen in comparison to results obtained on a 2-way Intel Nehalem processor with 2.53 GHz and a total of 8 cores and 8 threads running (corresponding results are shown in Figure 6).

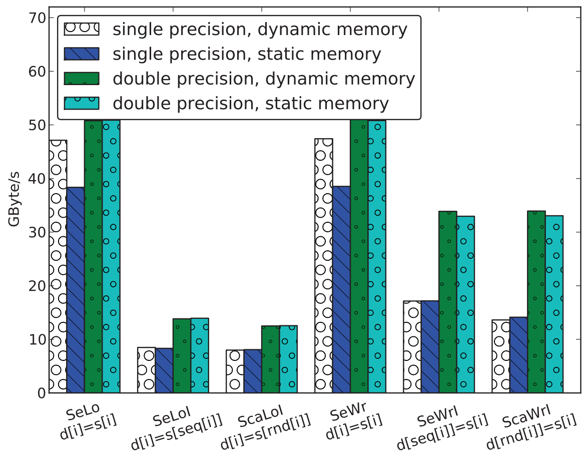

Fig. 5. Memory bandwidth of the Convey HC-1 coprocessor for different memory access patterns

For the sequential indirect access, Convey's compiler cannot detect possible improvements. For the scattered load and write, Convey's memory configuration gives better values than the Nehalem system. The Convey HC-1 not only has an about 60% percent higher peak memory bandwidth, but it really shows the potential of its scatter-gather capability when accessing random locations in memory. Here, traditional cache-based architectures typically perform poorly and GPU systems have a breakdown by an order of magnitude.

B. Data-Transfers Between Host and Device

Because of the strong asymmetric NUMA-architecture of the HC-1 there are different methods to use main memory. Three of them are used in the following examples:

- dynamically allocate (malloc) and initialize on the host; use migration pragmas
- dynamically allocate (cny_cp_malloc) and initialize on the device
- statically allocate and initialize on the host; use migration pragmas

By initialization we mean the first touch of the data in memory. Because the Convey HC-1 is based on Intel's precedent technology of using the front-side bus (FSB) to connect memory to processors a major bottleneck is the data connection between host memory and device memory. Figure 7 shows the relation between initialization and migration bandwidth in

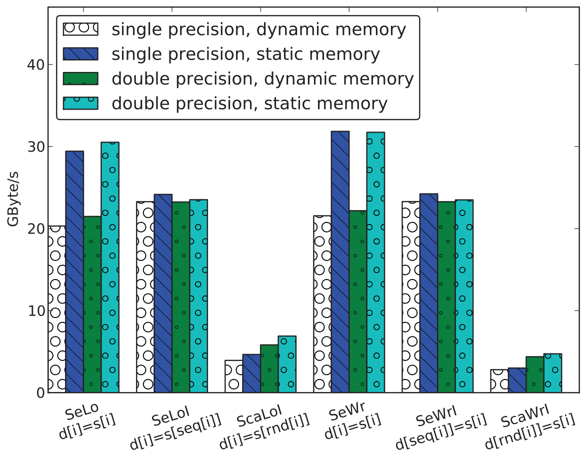

Fig. 6. Memory bandwidth of a 2-way Intel Nehalem processor with 2.53 GHz using 8 cores for different memory access patterns

terms of GB/s on the host and device and between host and device for the SAXPY vector update. Furthermore, Figure 7 depicts performance of the SAXPY in terms of GFlop/s (the unit on the y-axis has to be chosen correspondingly). For data originally allocated on the host and migrated to the device we observe some oscillations in the performance. While initialization inside the device memory reaches almost 20 GB/s, the transfer over the FSB achieves only about 700 MB/s. This impedes fast switching between parts of an algorithm which perform well on the coprocessor and its vector units and other parts relying on the flexibility of high-clocked general purpose CPU. Compared to GPUs attached via PCIe, the FSB represents an even more narrow bottleneck.

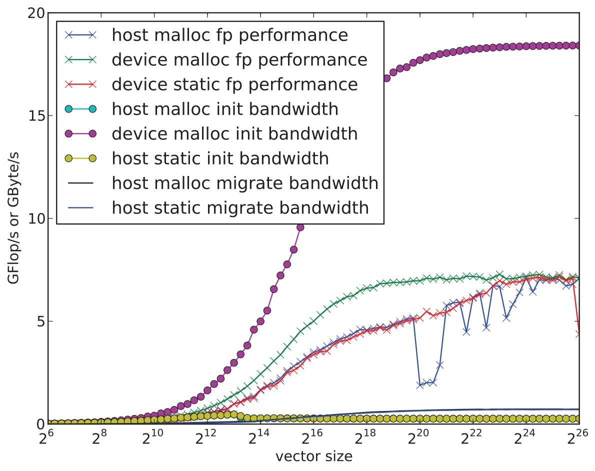

Fig. 7. Measured bandwidth for data initialization and migration for different memory allocation schemes in GB/s and performance results in GFlop/s for the SAXPY vector update

C. Avoiding Bank Conflicts with 31-31 Interleave

The scatter-gather memory configuration of the Convey HC-1 can be used in two different mapping modes:

- Binary interleave: traditional approach, parts of the address bitmap are mapped round-robin to different memory banks
- 31-31 interleave: modulo 31 mapping of parts of the address bitmap

Because in the 31-31 interleave mode the memory is divided into 31 groups of 31 banks, memory strides of powers of two and many other strides hit different banks and therefore do not suffer from memory bandwidth degradation. But to integrate this prime number scheme into a power of two dominated world, one of 32 groups and every 32th bank are not used resulting in a loss of some addressable memory and approximately 6% of peak memory bandwidth. In Figure 8 performance results for the SAXPY vector update are shown for both interleave options. For the SAXPY, binary memory interleave is slightly worse. Performance results for the CML DGEMM routine in Figure 9 show larger variations with 31-31 interleave. The DGEMM routine achieves about 36 GFlop/s and the SGEMM routine yields about 72 GFlop/s on our machine. In both cases this is roughly 90% of the estimated machine peak performance.

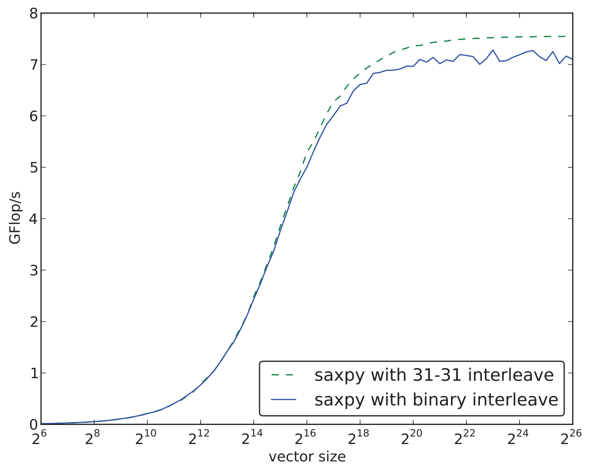

Fig. 8. Performance of SAXPY vector updates with 31-31 and binary interleave

D. BLAS Operations

Basic Linear Algebra Subprograms (BLAS) [10] are a collection and interface for basic numerical linear algebra routines. We use these routines for assessment of the HC-1 FPGA platform. We compare our own, straightforward implementations of BLAS-routines with those provided by Convey's Math Library (CML). Loop reordering techniques are applied for performance improvements. In the following examples we use the three different memory usage schemes

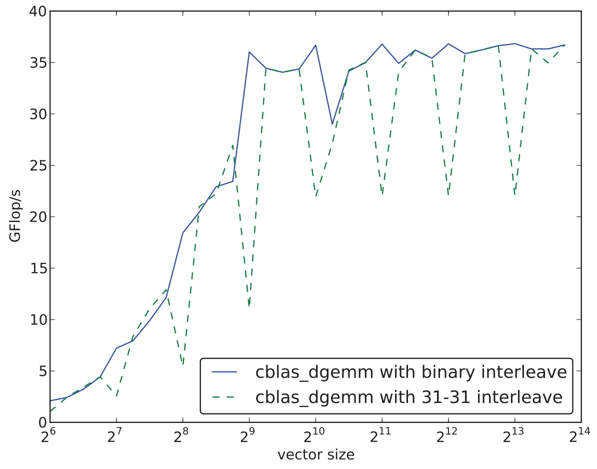

Fig. 9. Performance of the cblas_dgemm matrix-matrix multiplication provided by the CML with 31-31 and binary interleave

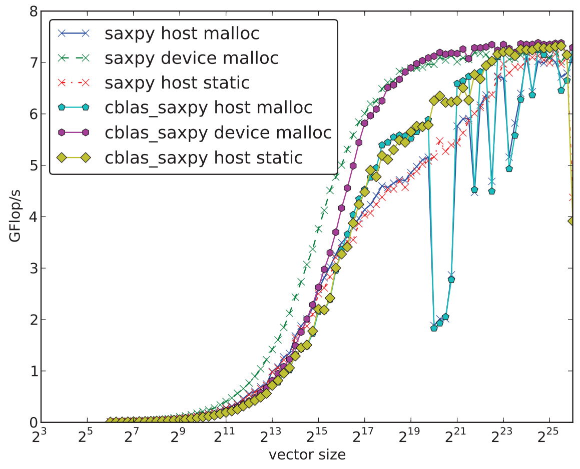

Fig. 10. SAXPY vector update using different implementations and different memory allocation strategies

detailed in Section VII-B. In all three cases initialization and migration costs are not considered in our measurements.

Data allocation on the host followed by migration routines or pragmas is not really a controllable and reliable procedure. From time to time considerable drops in performance are observed. So far, we could not identify a reasonable pattern or a satisfactory explanation for these effects. Our measurements are made using separate program calls for a set of parameters. When trying to measure by looping over different vector lengths, allocating and freeing memory on the host and using migration calls in between, the results are even less reliable.

We observe that our own implementations are usually faster for short vector lengths – probably due to lower call overhead and less parameter checking. For longer vector lengths the CML library implementations usually give better results. Results for the SAXPY/DAXPY vector updates are depicted in Figure 10 and in Figure 11. Performance data for the SDOT/DDOT scalar products are shown in Figure 12 and in Figure 13, and for the SGEMV/DGEMV dense matrix-vector multiplication in Figure 14 and in Figure 15.

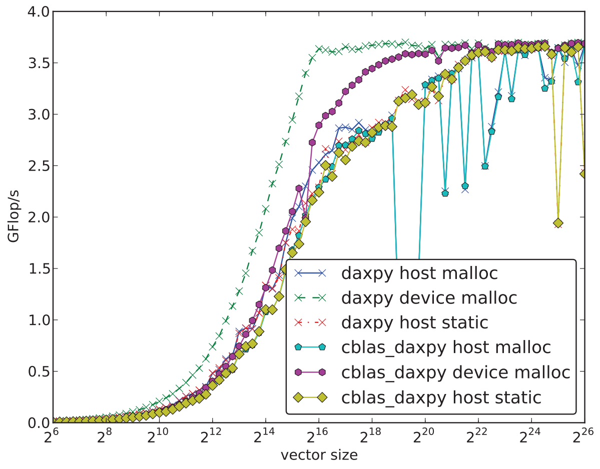

Fig. 11. DAXPY vector update using different implementations and different memory allocation strategies

E. Sparse Matrix-Vector Multiplication

Many numerical diescretization schemes for scientific problems result in sparse system matrices. Typically, iterative methods are used for solving these sparse systems. On top of scalar products and vector updates, the efficiency of sparse matrix-vector multiplications is very important for these scenarios. When using the algorithm for the compressed sparse row (CSR) storage format [11] presented in Figure 16, loops and reduction operations are vectorized by the compiler. However, the performance results are very disappointing – being in the range of a few MFlop/s. Although the memory bandwidth for indexed access as presented in Figure 5 is very good, the relatively short vector length and the overhead of the vector reduction in the inner loop seem to slow down computations (see also observations in Section VI on loop optimizations). Unfortunately, in this case loop reordering is not that easy because the length of the inner loop depends on the outer loop. A possible solution is to use other sparse matrix representations like the ELL format, as used on GPUs e.g. in [12].

F. Stencil Operations

Stencil kernels are one of the most important routines applied in the context of solving partial differential equations (PDEs) on structured grids. They originate from the discretization of differential expressions in PDEs by means of finite element, finite volume or finite difference methods. They are defined as a fixed subset of nearest neighbors where the corresponding node values are used for computing weighted

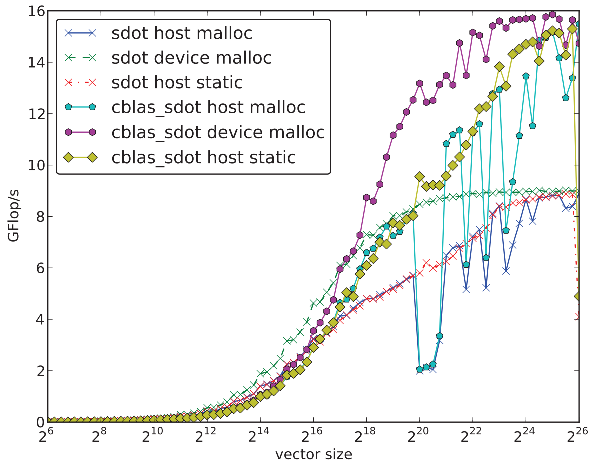

Fig. 12. SDOT scalar product using different implementations and different memory allocation strategies

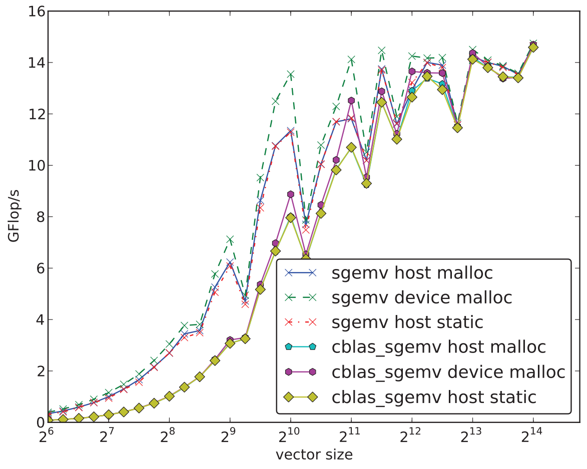

Fig. 14. SGEMV matrix-vector multiplication using different implementations and different memory allocation strategies

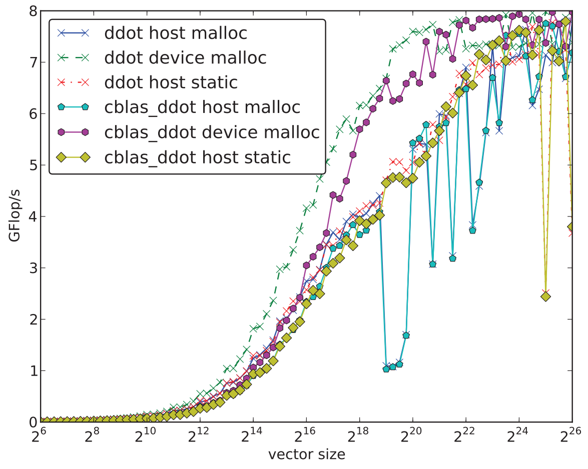

Fig. 13. DDOT scalar product using different implementations and different memory allocation strategies

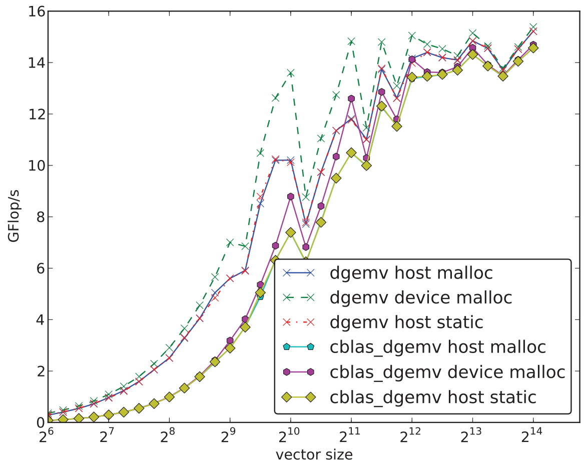

Fig. 15. DGEMV matrix-vector multiplication using different implementations and different memory allocation strategies

sums. Theassociated weights correspond to the coefficients of the PDEs where coefficients are assumed to be constant in our context. In our test we used a 3-dimensional 7-point stencil for solving the Laplace equation on grids of different sizes. The performance results are shown in Figure 17. Our stencil code is close to the example given in the Convey documentation material. The CPU implementation is the one used in [13], not using the presented in-place optimization but only conventional space-blocking and streaming optimizations.

For the conventional CPU one can see a high peak for small grid sizes when the data can be kept in the cache. For larger grid sizes a pretty constant performance with slight increases due to less loop overhead is observed. The Convey HC-1 on the other hand shows no cache effects but lower performance on smaller grids. But unfortunately because of its lack of caching, neighboring values of the stencil have to be reloaded every time they are needed – wasting a large portion of the much higher total memory bandwidth. On the Convey HC-1, the difference between single and double precision stencil performance becomes apparent only for large grid size.

VIII. Conclusion

Convey's HC-1 Hybrid Core Computer offers seamless integration of a highly capable FPGA platform with an easy coprocessor programming model, a coherent memory space shared by the host and the accelerator, and remarkable bandwidth values on the coprocessor. Moreover, Convey's scatter-gather memory configuration offers advantages for codes with

```
void spmv(int nrows, float val[],
          int coli[], int rowp[],
          float vin[], float vout[]){
#pragma cny no_loop_dep(val, vin, vout)
#pragma cny no_loop_dep(coli, rowp)
 for( int row = 0; row < nrows; row++ ) {
  int start = rowp[row];
  int end   = rowp[row+1];
  float sum = 0.0;
  for( int i = start; i < end; i++ ) {
   sum += val[i] * vin[coli[i]];
  }
  vout[row] = sum;
 }
}
```

Fig. 16. Sparse matrix-vector multiplication (SpMV) routine for CSR data format

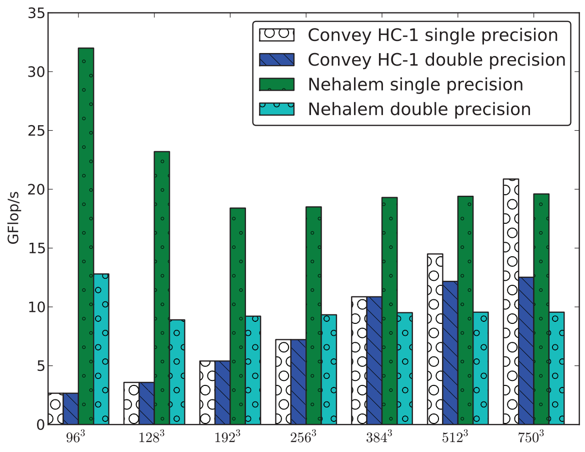

Fig. 17. Performance of a 3-dim. 7-point Laplace stencil (one grid update is counted as 8 Flop) on the Convey HC-1 and on a 2-way Intel Nehalem processor with 2.53 GHz using 8 cores

irregular memory access patterns. With Convey's personalities the actual hardware configuration can be adapted to, and optimized for specific application needs. With its HC-1 platform, Convey brings FPGAs closer to high performance computing. However, we have failed to port more complex applications originating in numerical simulation due to the failure to obtain acceptable speed for sparse matrix-vector multiplication.

The HC-1 has the potential to be used for general purpose applications. Although the HC-1 falls behind the impressive performance numbers of GPU systems and the latest multicore CPUs, it provides an innovative approach to asymmetric processing, to compiler-based parallelization, and in particular to portable programming solutions. Only a single code base is necessary for x86-64 and FPGA platforms which facilitates maintainability of complex codes. In contrast to GPUs, memory capacity is not limited by a few GB and FPGAs connected by direct networks come in reach. A great opportunity lies in the possibility to develop custom personalities – if time, knowledge and costs permit.

Convey's approach represents emerging technology with some deficiencies but also with a high level of maturity. Major drawbacks arise from limitations for floating point arithmetics on FPGAs, compiler capabilities for automatic vectorization, and the usage of Intel's obsolete FSB communication infrastructure. In our experience, typical code bases still show room for code and compiler improvements. While major benefits have been reported for specific workloads in bioinformatics, the HC-1 also provides a viable means for floating point-dominated and bandwidth-limited numerical applications. Despite its high acquisition costs, this breakthrough technology needs further attention.

Acknowledgements

The Shared Research Group 16-1 received financial support by the Concept for the Future of Karlsruhe Institute of Technology in the framework of the German Excellence Initiative and by the industrial collaboration partner Hewlett-Packard.

References

[1] P. J. Ashenden, *The VHDL Cookbook*. Dept. Computer Science, Univ. Adelaide, S. Australia. [Online]. Available: http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf
[2] D. Thomas and P. Moorby, *The Verilog Hardware Description Language*. Springer, 2008.
[3] J. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, "Compiling for reconfigurable computing: A survey," *ACM Comput. Surv.*, vol. 42, pp. 13:1–13:65, June 2010. [Online]. Available: http://doi.acm.org/10.1145/1749603.1749604
[4] A. Shan, "Heterogeneous processing: A strategy for augmenting Moore's law," *Linux J.* [Online]. Available: http://www.linuxjournal.com/article8368
[5] T. M. Brewer, "Instruction set innovations for the Convey HC-1 Computer," *IEEE Micro*, vol. 30, pp. 70–79, 2010.
[6] O. Storaasli and D. Strenski, "Cray XD1 – exceeding 100x speedup/FPGA: Timing analysis yields further gains," in *Proc. 2009 Cray User Group, Atlanta GA*, 2009.
[7] J. Bakos, "High-performance heterogeneous computing with the Convey HC-1," *Computing in Science Engineering*, vol. 12, no. 6, pp. 80 –87, 2010.
[8] J. M. Kunkel and P. Nerge, "System performance comparison of stencil operations with the Convey HC-1," Research Group Scientific Computing, University of Hamburg, Tech. Rep. 2010-11-16, 2010.
[9] Convey Computer, http://www.conveycomputer.com/resources/ConveyBioinformatics_web.pdf.
[10] Basic Linear Algebra Subprograms (BLAS), http://www.netlib.org/blas/.
[11] S. Williams, R. Vuduc, L. Oliker, J. Shalf, K. Yelick, and J. Demmel, "Optimizing sparse matrix-vector multiply on emerging multicore platforms," *Parallel Computing (ParCo)*, vol. 35, no. 3, pp. 178–194, March 2009.
[12] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, Dec. 2008.
[13] W. Augustin, V. Heuveline, and J.-P. Weiss, "Optimized stencil computation using in-place calculation on modern multicore systems," in *Euro-Par 2009 Parallel Processing, LNCS 5704*, 2009, pp. 772–784.

Preprint Series of the Engineering Mathematics and Computing Lab