CONVEY VECTOR PERSONALITIES – FPGA ACCELERATION WITH AN OPENMP-LIKE PROGRAMMING EFFORT?

Björn Meyer, Jörn Schumacher, Christian Plessl

Paderborn Center for Parallel Computing
University of Paderborn, Germany
christian.plessl@uni-paderborn.de

Jens Förstner

Department of Physics
University of Paderborn, Germany
jens.foerstner@uni-paderborn.de

ABSTRACT

Although the benefits of FPGAs for accelerating scientific codes are widely acknowledged, the use of FPGA accelerators in scientific computing is not widespread because reaping these benefits requires knowledge of hardware design methods and tools that is typically not available with domain scientists. A promising but hardly investigated approach is to develop tool flows that keep the common languages for scientific code (C, C++, and Fortran) and allow the developer to augment the source code with OpenMP-like directives for instructing the compiler which parts of the application shall be offloaded to the FPGA accelerator.

In this work we study whether the promise of effective FPGA acceleration with an OpenMP-like programming effort can actually be held. Our target system is the Convey HC-1 reconfigurable computer for which an OpenMP-like programming environment exists. As case study we use an application from computational nanophotonics. Our results show that a developer without previous FPGA experience could create an FPGA-accelerated application that is competitive to an optimized OpenMP-parallelized CPU version running on a two socket quad-core server. Finally, we discuss our experiences with this tool flow and the Convey HC-1 from a productivity and economic point of view.

1. INTRODUCTION

Many physical phenomena can be formally described as systems of partial differential equations. Due to the lack of an analytical solution many interesting problem instances can however only be solved by numerical integration of a spatially and temporally discretized finite-difference approximation. The numerical solvers operate iteratively on the grid, that is, they perform the same arithmetic operation for each spatial point and time or iteration step. Thereby, spatial neighbors are accessed in a regular manner. Obtaining high performance for such stencil computations is challenging, not because of the complexity of the arithmetic operations (in contrary, they are usually very simple) but because of the low number of executed operations per memory access. Stencil computations are therefore typically limited by the memory bandwidth and naive multi-core CPU implementations perform well below the peak floating-point performance. Due to the importance of stencil computations for computational sciences, considerable effort has been invested to develop methods for improving the performance on CPUs by exploiting the spatial and temporal locality of memory accesses [1–3]. Nevertheless, the numerical simulation of interesting problems usually still takes days or weeks on a standard workstation [4].

Hence, users from computational sciences show a strong interest in FPGAs or GPUs for accelerating their workloads. A significant factor preventing a widespread adoption of accelerator technologies in scientific computing is the need to re-implement the performance-critical application parts with unfamiliar programming languages, programming models and tool-flows. Blaming the reluctance of scientific users to use FPGA accelerators to a general aversion against new computer technology is unjustified, since the same community has quickly adopted OpenMP [5] as the preferred method for writing parallel programs for SMP multi-cores. It is more likely that the reluctance is caused by a perceived imbalance between the expected benefits and the expected effort for generating, validating, and maintaining an additional implementation of their already existing application.

Recently, a first generation of FPGA and GPU design tools have emerged that support an OpenMP-like approach for accelerator generation by keeping the programming language the same but augmenting it with compiler directives. The hardware/software partitioning and accelerator generation process is completely hidden from the developer. Hence, the promise of these tools is to achieve a significant acceleration with FPGA accelerators at a comparable effort to parallelizing applications with OpenMP. Also, the single source code concept that was a key factor in the success of OpenMP is retained.

The main contribution of this work is to study whether this OpenMP-like programming approach is effective in terms of programming effort and performance and whether
the approach is economically viable. We study these properties with an application from the area of computational nanophotonics, which implements the Finite-Difference Time Domain (FDTD) method for numerically solving Maxwell’s equations. The hardware platform we are targeting is the Convey HC-1 hybrid core computer, which is the first commercially available reconfigurable computer that specifically supports this OpenMP-like programming approach by providing a special configuration that implements a vector processor in FPGA logic and a corresponding vectorizing compiler.

The remainder of this paper is structured as follows: After a brief overview of related work in Section 2 we present the architecture and the OpenMP-like tool flow of the Convey HC-1 in Section 3. In Section 4 we introduce our case study application from the field of computational nanophotonics. In Section 5 we discuss selected details for three different implementations targeting multi-core CPUs and the Convey HC-1 architecture. In Section 6 we compare the performance of these implementations for varying problem sizes. Finally, we draw conclusions and discuss the merits of the approach in Section 7.

2. RELATED WORK

The fundamentals of the Convey HC-1 system architecture and its capability to implement instruction set extensions and custom personalities have been described in the works of Brewer [6] and Bakos [7]. The work by Augustin et al. [8] studies the suitability of the Convey HC-1 for kernels from linear algebra and compares the performance to CPUs and GPUs. Their work also uses the vector personality and compiler infrastructure. To our knowledge this is the first study that evaluates the productivity and effectiveness of Convey’s OpenMP-like tool flow for a complete application.

3. CONVEY HC-1 ARCHITECTURE AND TOOL FLOW

In this section we provide an overview of the Convey HC-1 architecture, the Convey “Vector Personality” configuration, and the corresponding parallelizing compilation tool flow.

3.1. Hardware Architecture

A schematic overview of the Convey HC-1 architecture [6] is presented in Figure 1. At its heart, the Convey HC-1 is a dual socket server system, where one socket is populated with an Intel Xeon CPU while the other socket is connected to a stacked coprocessor board. The two boards communicate using the Intel Front-Side Bus (FSB) protocol. Both processing units have their own dedicated physical memory, which can be transparently accessed by the other unit through a common cache-coherent virtual address space. The coprocessor consists of multiple individually programmable FPGAs. One FPGA, the “Application Engine Hub” implements the infrastructure that is shared by all coprocessor configurations. These functions include the physical FSB interface and cache coherency protocol, configuration and execution management for user programmable FPGAs and command dispatch logic that relays commands received from the host processor to the application-specific logic on the coprocessor.

For implementing the application-specific functionality four high-density Xilinx Virtex-5LX330 FPGAs are available. These FPGAs, denoted as “Application Engines”, are user-programmable and Convey has coined the term Personality for a particular configuration. As distinctive feature of the Convey architecture is that it lets personalities offer their functionality either as custom instructions for instruction-like operations operating on registers, or as function calls that may execute for many cycles and operate primarily on data exchanged via the memory subsystem. Users can create their own personalities or use personalities that can be licensed from Convey.

A distinctive feature of the HC-1 architecture is the availability of a fast multi channel memory interface which provides the application engines with access to 8 independent memory banks through 8 dedicated memory controllers with an aggregated bandwidth of 80 GB/s. Each memory controller accesses two DIMM memory modules. In total 16 DIMMs can be installed resulting in a maximum of 128 GB coprocessor memory. The contents of the accelerator memory are kept in sync with the CPU memory using a cache.
coherency protocol. The coprocessor does not provide any caches, hence memory accesses have a high latency which can and must be compensated with a large number of outstanding memory requests at any point in time. In order to further increase the memory performance, the coprocessor uses a weakly ordered memory model that can be synchronized on demand with fence instructions (see Section 5).

Besides standard memory modules Convey also offers custom-made scatter-gather modules which allow accessing memory efficiently in 8-byte data blocks while standard modules are optimized for 64-byte blocks. Additionally, the architecture allows for using a 31-31 (instead of binary) data interleaving scheme for obtaining peak memory bandwidth for array accesses with almost any stride size at the expense of a 6% reduction in overall memory capacity.

## 3.2. Vector Personality

For accelerating floating-point dominated scientific codes with SIMD characteristics Convey provides a “Vector Personality” [9], which configures the application engines to implement a programmable vector processor. The vector personality exists in two variants that differ in the supported floating-point precision (single vs. double precision). The vector personality is programmable, that is, it has a custom vector instruction set which is exposed to the x86 host processor as an instruction set extension.

### 3.3. Compilation Tool Flow

Programming the vector personality is supported by vectorizing compilers that supports C, C++ and Fortran. The developer needs to specify which functions of the code shall be vectorized and implemented with the vector personality by annotating the source code with pragma directives. This workflow is very similar to the OpenMP parallelization workflow. The annotated code is then compiled by Convey’s compilers. After the compilation the developer gets a vectorization report that specifies how many instructions of the vectorized code are analyzed by the compiler to identify instructions that are executable on the vector units of the coprocessor.

In addition to these directives, there exists a handful of other compiler directives that allow for controlling and fine-tuning the compilation process, for example, by specifying the unrolling factor for loops, suppressing memory fences, or removing false loop carried dependencies.

### 4. COMPUTATIONAL NANOPHOTONICS SIMULATION

Our work is motivated by the desire to accelerate simulations from the domain of computational nanophotonics on the Convey HC-1 computer. Therefore, we will briefly present the specifics of our model and its simulation with the Finite-Difference Time-Domain (FDTD) method [10,11] that solves Maxwell’s partial differential equations describing the electrodynamic propagation of light. The access characteristics of the stencil for this application in conjunction with the Convey HC-1 architecture properties allow us to employ optimizations as described in Section 5.

The primary goal of the considered algorithms is the simulation of the light field propagation in nano structures. The light is represented by the electric and magnetic vector fields $E$ and $H$, which are discretized on a regular cartesian grid for numerical simulations.

A nano structure, well suited for a non-synthetic benchmark, is the microdisk cavity in a perfect metallic environment (see Figure 2). This setup avoids open boundaries, which would complicate implementations. The numerical stability of such FDTD simulations and the well-known result, in the

```c
void compute(double *grid, int n) {
    int k;
    for(k = 0; k < n*n; k++)
        grid[k] = (grid[k] + grid[(k+1)%n])/2;
}
```

Listing 1: Convey compiler directives for controlling the vectorization process

and `end_coproc` directives. Within such regions, only functions that are compiled with the `dual_target` or `dual_target_nowrap`-option are callable. All code sections, whether they start with the `dual_target` or the `begin_coproc` pragma, are analyzed by the compiler to identify instructions that are executable on the vector units of the coprocessor.
Fig. 2: Geometry of a microdisk cavity and a micro-circular-cylinder, respectively, in a perfect metallic environment

form of Whispering Gallery Modes, are both further reasons for its attractiveness. The cylinder is modeled as air surrounded by an ideal metal material (perfectly electric conductor, PEC). Depending on the material, e.g., air or metal, at a point in space, different equations hold to describe the temporal evolution of the electromagnetic fields. Besides the 3D simulation, we investigate a 2D simulation, which is simply given by a plane sliced off of the cube, as denoted in Figure 2a.

As already discussed in more details our previous work [12], all fields are initialized to zero in the beginning and during the first few steps of the simulation a point (for 2D) or line (for 3D) source introduces a light field with a Gaussian temporal envelope. Within the Yee FDTD scheme, the E and H field are spatially and temporally interleaved and are updated iteratively-alternating using these stencils:

\[
E_x^{n+\frac{1}{2},j,k} = c_a(m_{i,j,k}) \cdot E_x^{n-\frac{1}{2},j,k} + c_h(m_{i,j,k}) \cdot H_z^{n+\frac{1}{2},j,k} + H_z^{n+\frac{1}{2},j,k} + H_y^{n+\frac{1}{2},j,k} - H_y^{n+\frac{1}{2},j,k+1} - J_{source,i,j,k}^{n+\frac{1}{2},j,k+1} \cdot \Delta \tag{1}
\]

and

\[
H_z^{n+\frac{1}{2},j,k} = d_a(m_{i,j,k}) \cdot H_z^{n, i,j,k} + d_h(m_{i,j,k}) \cdot E_x^{n+\frac{1}{2},j,k} + E_x^{n+\frac{1}{2},j,k} + E_y^{n+\frac{1}{2},j,k} - E_y^{n+\frac{1}{2},j,k} - M_{source,i,j,k}^{n+\frac{1}{2},j,k} \cdot \Delta \tag{2}
\]

Equations for \(E_y\), \(E_z\), \(H_x\), \(H_y\) are be obtained by cyclic rotation of the coordinates \(x,y,z\). For the implementation half-integer coordinates are mapped to integer array indexes, see Taflove et al. [11] for a detailed derivation. Material properties enter via the constants \((c_a,c_h,d_a\text{ and }d_h)\), the kind of material at a spatial position \((i,j,k)\) is determined by \(m_{i,j,k}\), thereby defining the geometrical setup.

Figure 3 shows the next-neighbor local access pattern of the stencil, which will be use in our implementations to optimize memory layout and cache strategies.

### 5. IMPLEMENTATION

In order to compare the performance of the FPGA-accelerated Convey HC-1 implementation with an OpenMP-parallelized multi-core implementation we have implemented our computational nanophotonics case study in three implementation variants. To provide a fair comparison, we have optimized the multi-core as well as the Convey implementations with a comparable effort. The implementations and the most important optimizations are described in the following.

#### 5.1. General Implementation Considerations

All implementations share a common simulation flow. In a first phase, the simulation is initialized, that is, all fields are initialized to zero and the material constants for Eq. 1 and 2 are computed. Also the number of simulation steps is determined which depends on the chosen spatial resolution and physical constants. The second phase is the most time consuming phase of the simulation which needs to be accelerated. In this phase, the propagation of electromagnetic fields is simulated by cyclically updating the E- and H-fields with Eq. 1 and 2 for the desired number of time steps. After each time step, the excitation of the fields is computed (source) and superimposed to the fields. Since we use a point source this process is not computationally expensive. The third and final phase starts towards the end of...
the simulation time, when the energy density of the fields is computed. As this step requires to evaluate the respective field in each grid point there is a potential for parallelization. However, since the energy is integrated only over a short time period, the contribution of this process to the overall simulation time is comparatively low.

5.2. Naive Parallel OpenMP Implementation

Our naive OpenMP implementation uses a direct implementation of the FDTD algorithm with basic optimizations only. For example, as a basic cache optimization we use flat arrays for all fields and static OpenMP scheduling to ensure that each spatial point is updated the same CPU core in every iteration. Furthermore, we prevent cache line invalidation caused by read-modify-write operations by using separate input and output arrays for all fields, which are switched (by pointer switching) after each time iteration.

5.3. Spatial Tiling Optimization

Since the performance of stencil computations is typically memory-bound in CPU implementations, we have optimized our FDTD multi-core implementation by applying a spatial tiling (also denoted as cache blocking) strategy, which improves the utilization of the multi-core cache hierarchy [1], [2], [3]. This tiling strategy decomposes 2D and 3D grids into 2D tiles. The optimal length and width of the tiles depends on the CPU architecture and the memory access pattern of the stencil. For determining the optimal size we have measured the performance for a wide range of length and width configurations, see Figure 5. The figure illustrates that finding the optimal tile configuration in an analytical way is hard, hence we used an auto-tuning approach.

Since we are running the OpenMP implementation on a 2-socket SMP machine with NUMA characteristic (that is, each processor has its own dedicated memory which can be accessed faster than the memory on the other socket) we have chosen a NUMA-aware first-touch memory allocation policy to enforce a strong relationship between cores and data processed by the cores. Since we are using static OpenMP scheduling each tile is processed by the same core every time.

5.4. Convey HC-1 Vector Personality Implementation

The implementation for the Convey HC-1 vector personality is based closely on the naive OpenMP multi-core implementation, because the spatial tiling optimization does not provide any performance benefits due to the lack of caches on the HC-1 coprocessor. The implementation effort for this basic implementation is very low. The only required addition to the source code is adding directives to specify that the computational intensive kernels, that is, the updated equations for the E- and H-fields, shall be moved to a coprocessor region. For improving the performance we have made the following optimizations.
5.4.1. Memory allocation on coprocessor
While the unified virtual address space provides the accelerator with transparent access to host memory the Convey HC-1 still has a NUMA architecture, that is, the performance of a memory access depends on where the memory is physically allocated. Large data should thus be allocated on the resource which accesses the memory most frequently. Hence, we use Convey-specific memory allocators to bind the memory for the E- and H-fields to the accelerator.

5.4.2. Loop unrolling
Convey’s vectorizing compilers can perform loop unrolling automatically but allow the developer to override the unrolling factor. We have empirically determined that using an unrolling factor of 16 yields better results than the default unrolling factor for our case study.

5.4.3. Masked code execution
Computing the propagation of the electromagnetic fields in our case study requires to consider which material (air, metal) is present at each grid point of the simulated structure. Hence, when updating the E- and H-fields the FDTD method selects the appropriate update equation using conditionally executed code. A limitation of the current version of Convey’s vectorizing compiler is that conditional code execution in a loop prevents the vectorization of the loop. This restriction is unexpected, since the vector processor supports masked operations, which are usually used for implementing conditionals in vectorized code. To work around this compiler limitation, we emulate vector masking in our application code by computing both possible field updates and select the appropriate results using a binary mask that can be multiplied for each grid point. We have used the same approach in previous work for optimizing a GPU implementation of the FDTD method.

5.4.4. Memory Barriers
A peculiarity of the Convey coprocessor memory interface is the use of a weakly ordered memory model. This model is used because the absence of caches requires to have many concurrent memory requests in flight at any point in time and implementing a strongly ordered memory model would cause too much overhead. In consequence, whenever a consistent memory state is required, the compiler inserts memory fence operations. Detecting the need for fences is a difficult problem, thus the compiler has to be conservative and tends to insert more fence operations than actually needed.

The data access patterns of our actual application are particularly good-natured for a weakly ordered memory model, since the E- and H-field values from the previous iteration are accessed solely read-only and the new H- and E-field values are written to a different array. It would thus suffice to insert a fence only after a time step when all grid points have been updated. Still, the compiler is not able to detect that a single fence after a time step is sufficient and inserts superfluous fences within the update loop. We avoid this behavior by using the compiler directive #pragma cony no_fence(within).

6. EXPERIMENTAL SETUP AND RESULTS
In this section we compare the performance the three implementations described in the previous section for the described 2D and 3D nano structures for different grid sizes.

6.1. Experimental Setup
We evaluate our implementations on three different computer systems presented in Table 1. The last one is a dual-socket Intel X86 workstation with two quad-core Intel Xeon Westmere-EP CPUs. For evaluating the effect of different memory-subsystem configurations, we evaluate the FPGA-accelerated implementation with two Convey systems.

<table>
<thead>
<tr>
<th></th>
<th>Convey HC-1</th>
<th>Convey HC-1</th>
<th>Workstation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon 5138</td>
<td>Xeon L5408</td>
<td>Xeon E5620</td>
</tr>
<tr>
<td>Clock [GHz]</td>
<td>2.13</td>
<td>2.13</td>
<td>2.4</td>
</tr>
<tr>
<td>Cores</td>
<td>2</td>
<td>4</td>
<td>2 × 4</td>
</tr>
<tr>
<td>Cache [KB]</td>
<td>4096</td>
<td>6144</td>
<td>2 × 12288</td>
</tr>
<tr>
<td>Memory [GB]</td>
<td>24</td>
<td>64</td>
<td>12</td>
</tr>
<tr>
<td>DIMM type</td>
<td>standard</td>
<td>scatter-gather</td>
<td>standard</td>
</tr>
</tbody>
</table>

Table 1: Hardware Platforms

The multi-core workstation is running a 64 bit Linux version 2.6.32 kernel. All OpenMP algorithms are compiled using GCC 4.4.3 with the -O3 optimizations. The Convey machines are running a 64 bit Linux version 2.6.18 kernel adapted by Convey. The code for the Convey machine is compiled with the Convey cnyCC compiler version 2.4.0 with the -O3 optimizations.

We execute the FDTD algorithm for 1000 time-iterations. The E- and H-fields are computed in double-precision floating point arithmetic. As performance comparison metric we have chosen MStencils/s instead of GFlops. A stencil update is defined as the computation of one time step in the FDTD method for a single grid point.

6.2. Performance of 2D simulations
As expected, for small problem sizes the OpenMP multi-core implementations outperform the Convey implementation due to small data that fit into the CPU cache. Hence, we can observe peak MStencils/s rates as shown in Figure
6. For larger problem sizes, the situation changes. For example, the $4096 \times 4096$ grid size needs 99.15 seconds in the naive version, 55.02 seconds in the tiled version and 38.13 seconds using the Convey HC-1 with scatter-gather RAM and binary interleaving. We achieve 337 MStencils/s with the naive, 627 MStencils/s with the tiled and 880 MStencils/s using the Convey implementation with scatter-gather RAM and binary interleaving (Figure 6).

Figure 7 compares the achieved speedups over the naive OpenMP baseline implementation for different problem sizes. For the tiled multi-core implementation, we use the tiling configuration that results in the best performance. The used tiling configuration is labeled in parentheses in the graph (e.g., $(x$-axes-dimension$,y$-tile-dimension$))

We notice that the performance of the tiled multi-core version for the largest 2D case is slightly better than the Convey version running with standard DIMM modules. In this case, the memory interface suffers from the memory access pattern of the FDTD algorithm in conjunction with the standard RAM modules that are optimized for cacheline-sized 64-byte data transfers. The best performance is achieved with the Convey machine in combination with scatter-gather RAM in binary interleaving mode. Compared to the naive implementation, we achieve a speedup of up to $2.60 \times$ for the 2D case with the Convey HC-1.

6.3. Performance of 3D simulations

Also for small 3D problems, both OpenMP implementations outperform the HC-1 implementation because of cache friendly data sizes. The computation of the $256 \times 256 \times 256$ grid takes 230.02 seconds with the naive, 122.5 seconds with the tiled and only 102.94 seconds with the Convey implementation using binary interleaving.

When comparing the MStencils/s performance for 2D and 3D grids, we observe a noticeable performance decrease for 3D computation. For increasing problem sizes the performance saturates as in the 2D case. However maximum performance of 326 MStencils/s in the 3D case is approximately $2.7 \times$ lower than for the 2D case.

![Fig. 6: Achieved MStencils/s for varying 2D grid sizes on multi-core and Convey HC-1 implementation](image)

![Fig. 7: Speedup compared to the naive OpenMP implementation for 2D setups with varying grid sizes](image)

![Fig. 8: Achieved MStencils/s for varying 3D grid sizes on multi-core and Convey HC-1 implementation](image)

Obviously, the arithmetic intensity, that is floating-point operations per data access, differs for 2D and 3D setups because of the varying number of equations for modeling and spatial points accessed in the nearest neighborhood. Hence, the memory access intensity and pattern changes and raises the pressure on the memory interfaces.

Consequently, the speedup curve is flatter but steadily rising for the 3D simulation. Compared to the naive im-
implementation, we get a speedup for $128 \times 128 \times 128$ grid sizes and above (Figure 9). For all considered grid sizes, the Convey run with scatter-gather RAM performs slightly better than the tiled multi-core implementation. As in the 2D case, binary interleaving yields the best performance.

![Fig. 9: Speedup compared to the naive OpenMP implementation for 3D setups with varying grid sizes](image)

7. CONCLUSION

Convey has taken on the challenge to provide an integrated solution for a reconfigurable computer that is programmable with an OpenMP-like tool flow and aimed explicitly at scientific computing. In this work we intended to get insight into two main questions 1) is Convey’s approach of programming FPGA-accelerated HPC systems with an OpenMP-like tool flow feasible and efficient; and 2) is the generated implementation competitive with competing technologies in terms of performance and cost?

From our experience and the results reported in this paper, we can answer the first question positively. We have exposed a student without previous FPGA development experience to this platform and tool flow and he was able to generate, optimize and validate an FPGA-accelerated solution within a few weeks.

The second question question needs a more differentiated answer. Our results have shown that the performance of the implementation for the Convey vector personality is competitive with a multi-core implementation running on a two-socket workstation. For larger problem sizes, which are more representative of actual use-cases from computational nanophotonics research, the Convey HC-1 implementation consistently outperforms the multi-core implementation. However, the margin by which the Convey implementation outperforms the multi-core implementation is comparatively small, that is, 40% for the largest 2D problem and 20% for the largest 3D problem. Considering that the Convey HC-1 is approximately 10 times more expensive than the used multi-core system it is evident that the use of an HC-1 machine is not economic for this particular application unless the increase in performance is absolutely critical.

8. REFERENCES


