Experiences with High-Level Programming of FPGAs on Cray XD1

Thomas Steinke, Alexander Reinefeld, Thorsten Schütt
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

ABSTRACT: *We recently started to explore the potential of FPGAs for some application kernels that are important for our most demanding supercomputer users at ZIB. In this paper, we give an overview of our efforts and present current status achieved so far on a Cray XD1 system using the Mitrion-C programming environment.*

KEYWORDS: High-performance computing, Cray XD1, FPGA, Mitrion-C

1. Introduction

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) is a non-university research institute of the state of Berlin. In close interdisciplinary cooperation with the Berlin universities and scientific institutions it implements research and development in the field of information technology with a particular focus on application-oriented algorithmic mathematics and practical computer science. ZIB provides high-performance computer capacity as an accompanying service for researchers in Berlin, Germany and abroad.

Within the *North-German HPC Consortium (HLRN)*, ZIB operates an IBM p690 that consists of 16 p690 systems with 64Gbyte up to 256 Gbyte each, giving a total of 512 processors and 1.25 Tbyte of memory with a peak performance of 2.6 TFlops. Linked via dedicated fibre optics to its peer-system in Hanover, the two super-computers are operated as one virtual supercomputer.

FPGA-Based Computing at ZIB: The Cray XD1 System

With the procurement of an FPGA equipped Cray XD1 system in September 2004, ZIB was the first site in Germany to provide FPGA technology for its HPC users. Partly funded by the BMBF for the BCB groups[1] at ZIB, the Cray XD1 system is primilary used for applications in the area of bioinformatics and life sciences.

The motivation for exploring FPGA as application accelerators results from the following technology trends:

- The predominant quantitative approach in major HPC systems is accompanied by a sharp increase in the running costs due to a less efficient power utilization caused by increased processor frequency and massive use of underutilized commodity components. As an example, the operation costs for power consumption of the current HLRN system are in the range of EUR 250,000 per year plus running cost for cooling.
- The sustained performance of only about 10-15% of the peak performance for cache-based microprocessor architectures has been relatively constant over the last years—at an irritatingly low margin. Vector computers, on the other hand, deliver a higher sustained performance, but suffer from a decreasing application code base, because of their incompatibility with PCs and servers, on which code development is usually performed.
- Physical considerations alone (heat dissipation per Flop, increasing signal delays due to long pathways between functional units, etc.) leads us to believe that HPC with current microprocessor designs is not feasible for petaflop computers at reasonable operation costs.

The unsatisfactory application performance despite increased peak performance and increased running costs caused scientists to establish strategic partnerships with HPC manufacturers like the "Blue Planet" partnership between Lawrence Berkeley National Laboratory and IBM, which resulted in a set of architectural extensions

[1] BCB = Berlin Center for Genome Based Bioinformatics

referred to as ViVA (Virtual Vector Architecture) that positively affected the architectural design of the ASCI Purple system [1].

The lack of computer manufacturers in Europe forced us to pursue another strategy, namely to optimally exploit standard FPGA coprocessors as accelerator for the most important HPC application kernels. Note that contrasting other, more hardware-oriented research groups, we do not have any expertise in hardware description languages, neither do we want to dive into this subject. Instead, we build on high-level programming languages (like Mitrion-C) which allow us to focus on the development of highly efficient application kernels without the burden of having to implement highly specific VHDL code. Using Mitrion-C has the additional advantage, that non-CS experts such as our HPC users from various application domains may improve the FPGA kernels without additional help.

Applications of Interest

Compute intensive molecular simulations are one major class of application on ZIB's computing systems. These simulations cover a wide range of molecular systems and simulation techniques: from small molecules to large biomolecules, from highly sophisticated stationary or time-dependent electronic structure calculations to molecular dynamics calculations based on empirical potentials. The latter belongs to the class of applications where the calculation of non-bonding interactions is usually the most time consuming step. Today, molecular dynamics calculations are performed with optimized production code (e.g. NAMD, AMBER) or in-house developed codes having specialized features (e.g. hybrid Monte-Carlo).

2. Cray XD1 System with Attached FPGA

At ZIB, one XD1 chassis is installed. It contains 12 AMD Opterons at 2.2 GHz and 12 Gbyte of memory. The 12 CPUs are organized in 6 nodes connected by the Cray RapidArray interconnect. Each node hosts one Xilinx Virtex-II Pro XC2VP50 FPGA with 16 Mbyte of QDR SRAM organized in 4 memory banks with 4 Mbyte each. The FPGA has a maximal frequency of 197 MHz. It comprises of

- 53136 logic cells,
- 4176 Kbits of block ram,
- 232 18x18 multipliers, and
- 2 PowerPC cores, which are not used with Mitrion-C.

3. Non-Bonding Interactions: The Lennard-Jones Potential

For a large ensemble of interacting particles the most time-consuming step in the approximate evaluation of the energy or mutual forces is the calculation of pairwise interaction term. A fast calculation schema is needed to simulate large particle ensembles. The Lennard-Jones (LJ) potential is a commonly used equation that expresses the non-bonding interaction of atomic or molecular particles. Its has the advantage that the non-specific attraction caused by the dispersion interaction and the repulsion caused by the Pauli repulsion for short distances are formulated in a compact equation.

For two particles, the energy U depends on the distance r separating both particles:

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^6\left(\left(\frac{\sigma}{r}\right)^6 - 1\right)\right]$$

The parameters σ and ε, respectively, are specific for each type of particle pair under consideration. The force F acting on the first particle due to the presence of the second particle at distance r:

$$F(r) = \frac{24\varepsilon}{r^2}\left[\left(\frac{\sigma}{r}\right)^6\left(2\left(\frac{\sigma}{r}\right)^6 - 1\right)\right]$$

Related Work

The MD-GRAPE project [2] is one of the most successful approaches which uses hardware acceleration for n-body simulations. The MD-GRAPE system is based on an ASIC design which comprises the implementation of multiple pipelines for the calculation of non-bonding interaction energies and forces. On the latest MD-GRAPE-3 chip 20 pipelines for calculation of forces are implemented. Operated with a clock rate of 300 MHz the chip performs 660 operations per cycle and has a peak performance of 198 GFlop/s [3].

A highly optimized VHLD design with two pipelines using IEEE 754 double precision floating-point units for the calculation of LJ potentials and forces on a Xilinx Virtex-IIPro125 chip was recently presented by Scrofano and Prasanna [4]. Their implementation achieves a performance of 3.9 GFlop/s on the XCV2Pro125-7, however, on a much larger FPGA than ours.

Recently, Kindratenko and Pointer [5] reported their effort in porting the molecular dynamics code NAMD to the SRC-6 platform. On the SRC-6 system, they achieved an *overall* speedup of 3 compared to a 2.8 GHz x86-CPU which is remarkable for a highly optimized production code.

4. High-Level Algorithm Design for FPGA with Mitrion-C

Traditionally, algorithms on FPGA are expressed by the behaviour of the hardware and described by HDLs such as VHDL or Verilog. Mitrion-C is a C-like high-level language with certain properties known from functional programming languages. We have choosen Mitrion-C because of its attractive underlying concept of hardware virtualization. Another issue was its earliest possible availability on the Cray XD1 platform. Mitrion currently supports Cray, Nallatech and SGI. Other C-to-VHDL approaches, for example Handel-C or Impulse-C, as well as other high-level approaches, e.g. from DSPlogic will be considered shortly.

Short Intro: Mitrion Programming Environment

We give an overview about the Mitrion programming environment from the application programmer's point of view. The concept of hardware abstraction with the Virtual Mitrion Processor are described in detail elsewhere [6, 7].

The current Mitrion release [8] includes

- the Mitrion-C to VHDL compiler,
- the Data Dependency Graph visualizer,
- the Mitrion simulator server,
- the Mitrion Hardware Abstraction Layer library (Mithal), and
- `parcheck` for summarizing and checking the result of the place and route design process with Xilinx ISE `par` [ise].

In the following, we focus on the two most important components of the Mitrion package.

The **Mitrion simulator**, introduced with rel. 1.1.0 (Feb 2006), substantially reduces the development time. It acts as a virtual FPGA device when the host program is linked with the simulator version of Mithal. In this way, the process of the host program and the simulator process interact logically in the same way as the host program with the FPGA device. The validation of the implementation (watch points in the Mitrion-C code) and the monitoring of I/O operations to/from external memory become fairly easy.

The **Mithal interface**, introduced with rel. 1.1.1 on Cray XD1 (Mar 2006) improves the portability of the implementation across different hardware platforms (e.g. Cray XD1 and SGI RASC).

In Mitrion-C, the most important data structures are collections. Collections are represented by two basic types: lists and vectors. List collections are used for streaming data and invoking pipeline parallelism. The vector collection implicitly represents multiple instances of the Virtual Mitrion Processor (processing elements) and therefore data parallelism. Thus, to implement an algorithm in Mitrion-C one needs to think in terms of transforming collections by means of self-written functions (of course, very short and biased view on the potential of Mitrion-C).

The Brief History of FPGA Programming at ZIB

After its first release, the Mitrion-C package shows an exceptional vivid life cycle (see "diary" in text box below). Based on experiences and feedback from user communities, several releases were made available in a short frame of time. Most releases had a major impact on the development either by introducing new functionality or changes in the Mithal library.

Here is my (Th.St.) Mitrion-C release "diary":

- *2005, Sept 28: got access to pre-release of Mitrion*
- *2005, Nov 9: got the first official Mitrion release 1.0.1 installed on our XD1, realized that we have an old Cray release (1.1) which is not supported by Mitrion 1.0.1*
- *2005, November: Cray Inc announces availability of Cray rel 1.3 at SC05, so it should be ready for download but it is not, urgent contacts to Cray Germany to solve the problem (students are waiting…)*
- *2005, Dec 20: Cray 1.3 finally installed with the help of R. Pennarun (Cray French), run demos from Mitrionics*
- *2006, Feb 12: Mitrion 1.1.0: comes with the simulator - great step forward!*
- *2006, Mar 20: Mitrion 1.1.1: Mithal introduced on Cray XD1*
- *2006, Apr 3: Mitrion 1.1.4: Mithal changed, explicit data transfer function removed from API, data transfer trigger is now hidden (I don't like that)*
- *2006, Apr 19: Mitrion 1.1.5: on XD1, access to host DRAM via FTR possible*

Bottleneck: Bandwith Host – FPGA

It is well documented that the quality of the host – FPGA interface determines the resulting overall performance improvement, and therefore, the acceptance of using FPGAs as accelerating devices.

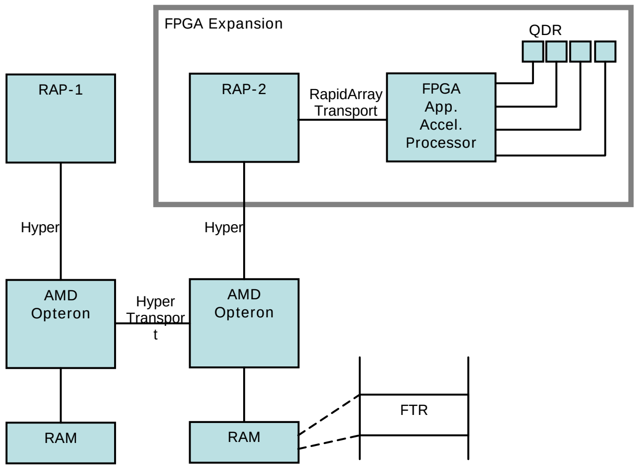

Figure 1: Interface between FPGA and Opteron

Figure 1 illustrates the architecture of the functional units within the FPGA expansion module and the corresponding memory layout. To become familiar with the Mitrion-C concept and the usage of the host API we performed some bandwidth measurements. The program codes were simple in their computational task, e.g. each entity in a data stream is incremented by a fixed number. They only differ in their kind of input/output interface to the SRAM modules and later into the FTR. The FTR, standing for FPGA Transfer Region, is the DRAM host memory (accessible with Mitrion rel. 1.1.4).

Table 1 summarizes our performance results. The first benchmark case shows the performance when the Opteron triggers the data transfer between the SRAM and its own (host) memory. Note the asymmetry: Reading data from SRAM to the host is 156-times slower than sending. When using the FPGA to trigger the data transfer, see the next line, the results are only slightly worse.

Table 1: In/Out Bandwidth Measured with Mitrion-C

Type/Mithal-Version	Data Transfer Chain	Bandwidth [Mbytes/s]
stream/1.1.1	host memory → SRAM	764.9
	SRAM → host memory	4.9
	total throughput	9.2
dram/1.1.5	FPGA → host DRAM	675.1

Hardware Abstraction Layer: Extension to Support Fortran

On the XD1 system, the API for resource allocation and communication between the host and the FPGA is provided by Cray's `ufplib` [10]. On top of the Cray API, Mitrionics implemented their own hardware abstraction layer, named Mithal [11].

Most of the important applications considered here are implemented in Fortran. Both APIs, the one for host-to-FPGA communication and Mithal have a C interface. Especially the Mithal interface is not directly accessible from Fortran due to the use of character string arguments for SRAM and DRAM bank identification. Since the large majority of our HPC applications use Fortran, we implemented a thin wrapper around both APIs and made them accessible to Fortran as subroutine calls. [12], see Figure 2.

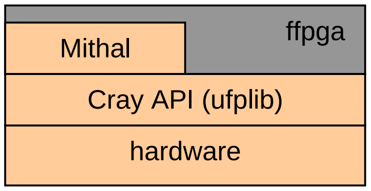

Figure 2: Thin Fortran API layer `ffpga` wrapping the Mithal and the Cray API.

With the ffpga functions, we hide any FPGA specific data structures and housekeeping functions from the Fortran application programmer. A summary of our Fortran interface can be found in Table 2.

Table 2: Fortran `ffpga` interface: API definition and short functional description

`ffpga_init (bitfile, stat)`	allocate the FPGA device, load bitfile into FPGA
`ffpga_reg_buffer(id, nbytes, mode, stat)`	returns an address of registered buffer space in host memory
`ffpga_start ( stat )`	start the FPGA
`ffpga_wait ( stat )`	wait for FPGA finalization
`ffpga_close ( stat )`	close the FPGA device
`ffpga_wrreg (id, ptr, stat)`	write data to FPGA register
`ffpga_wdta (id, ptr, offset, nbytes, stat)`	transfer data from registered buffer to SRAM
`ffpga_rdta (id, ptr, offset, nbytes, stat)`	transfer data from SRAM to host buffer

The bottom two API functions are available for Mithal releases up to 1.1.4. They trigger the explicit data transfer to/from SRAM banks from/to pre-registered host memory space. With the `ffpga` layer one can easily switch between the two FPGA vendor API without re-coding the host program. We take advantage of this in some phases of the development process.

Reconfigurable Computing Taken Seriously

One outstanding feature of reconfigurable computing devices is its ability to reconfigure functional units at different levels between jobs, processes, and during runtime.

One possible approach is to migrate a part of an algorithm onto the reconfigurable device. Then only *one* computational functionality of a given application can be used at runtime.

Another approach is to reconfigure the device on-the-fly during job or process execution. This is attractive for multi-step applications, because the limited space resources on todays FPGA restrict the size of the application kernels that can moved onto the FPGA. The frequency of FPGA reconfigurations, however, is limited by the time needed for loading the bitfile into the FPGA.

The Cray XD1 ufplib supports only files as a source for the bitfile while other vendors support memory access as well [13]. On the Cray, the average loading time from disk is about 1.68 s for a 2.27 MB bitfile, corresponding to 3.7E+9 cycles on the 2.2 GHz Opteron CPU. Note that the load time is *not* limited by the disk access time: Loading the bitfile from RAM disk we measured slightly larger loading times of about 1.82 s. The time to load the FPGA is limited by the I/O interface between the Opteron CPU and the programming port of the FPGA (LPC bus performance bottleneck [14]).

Therefore, on-the-fly reconfiguration of FPGAs during is only useful when the time period between the execution of different computational functions on the FPGA is in the order of minutes.

5. Mitrion-C Implementation of LJ Force Calculation

Test and Benchmark Environment

For testing as well as benchmarking our Lennard-Jones implementation, we choose to use a very simple scenario. We implemented the LJ particle code in Fortran as described by Haberlandt et. al. [15]. Our `p1MD` implementation acts as our host program for subsequent FPGA implementations. The implementation is reduced to the core functionality found in most applications dealing with empirical force-fields.

The timing profile in Figure 3 illustrates that the time consuming step, i.e. the calculation of the forces between interacting particles, is represented by the two dominating subroutines `nforce` and `force`, respectively: both subroutines consume over 90% of the computational time in this test scenario (again we point out, this profile is measured for the test case under investigation; in a practical scenario the proportion will be lower, still the qualitative picture will not change).

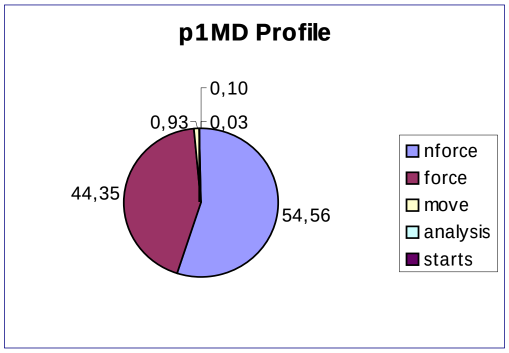

Figure 3: Timing profile for benchmark p1MD, 1000 steps for 1000 particles on standard processor.

Design Principles

Our design/implementation of the LJ was guided by the following principles:

- Stream data directly from/to RAM modules.
- Use deep as pipelines as possible.
- Support maximal data input/output bandwidth.
- Keep it simple.

32bit Implementation

Some of the empirical force-field implementations are based on 32bit processing. This is motivated by the fact, that the simulation of molecular dynamics by integrating Newtons equation of motion is a chaotic process. The numerical noise caused by the physics overlaps the noise caused by rounding errors, therefore some implementations prefer a 32bit implementation with short trajectories (for example in hybrid Monte-Carlo approaches).

We started with the implementation of an IEEE 32bit version for the calculation of the Lennard-Jones potential and forces. We noticed soon that in our first naïve approach the number of required flip-flops, multipliers and subsequently slices exceeded the available resources on the XC2VP50. Since floating point division and square root operations are resource demanding, we put these operations into the host program. Later, we will try to perform the division in the FPGA.

The memory interfaces to SRAM banks and DRAM is 64bit wide on the Cray XD1. At each cycle we read or write a full 64bit word per RAM bank. After reading, this word is transformed into a list of two 32bit members. For both the read and write operations simple I/O functions are written in Mitrion-C.

The core of the computational engine on FPGA is designed to calculate the LJ energy part as well as the force factor for a given particle pair. The three data input streams deliver the distance (or reciprocal distance) and the pair-specific parameters ε and σ. These three data streams are fed into the function which computes the energy part and the force factor. Both 32bit results are packed into 64bit words and the later are streamed out to SRAM or DRAM. This described procedure represents one pipeline. Depending on the number of functional components on the FPGA chip it is easily extensible to more pipelines in Mitrion-C by introducing a vector collection.

The host program generates the pair list based on a given cutoff distance criterion, which is common practice in particle simulations to decrease the complexity from $O(N^2)$ to $O(N \log N)$. The host gathers the pair parameters ε and σ and stores the three objects in a buffer. Then the FPGA is started by an asynchronous thread and it is waited for the finalization of the FPGA. Our current implementation does not support the overlapping of data processing on the host site with the computation on the FPGA. This is planned for a forthcoming version.

Based on our simulations with the Mitrion simulator, the startup latency for reading the first word is 5 cycles, with the 14 cycle the first datum arrives at the `efALJ`-function. The `fA`-function is 16 cycles deep so that after 29 cycles, the first result is passed to the output. Finally, after 35 cycles, the first result is written to the FTR interface. This pipeline involves 15 IEEE single precision floating-point operations (multiply, add). After the pipeline is filled, it will provide a sustained performance of 1.5 GFlop/s (single precision).

64bit Implementation

Furthermore, we implemented one pipeline for the calculation of the force factor with full IEEE 64bit support. The implementation involves 10 floating point operations (add, multiply) and one division. After the pipeline is filled, it will provide a sustained performance of 1.1 GFlop/s.

For each cycle, the three operands: The distance (`r`) and the interaction parameters ε (`eps`) as well as σ (`sig`) are each read from one of the SRAM banks A, B, and C, respectively. After the pipeline is filled, in each cycle one 64bit results is written into the FTR interface[2].

Based on our simulations with the Mitrion simulator, the startup latency for reading the first word is 4 cycles, after 11 cycles the first datum arrives at the `fA`-function pipe. The `fA`-function is 15 cycles deep so that after 25 cycles, the first result is passed to the output. Finally, after 27 cycles, the first result is written to the FTR interface. Figure X shows the fA evaluation part of the total pipeline of 27 cycles.

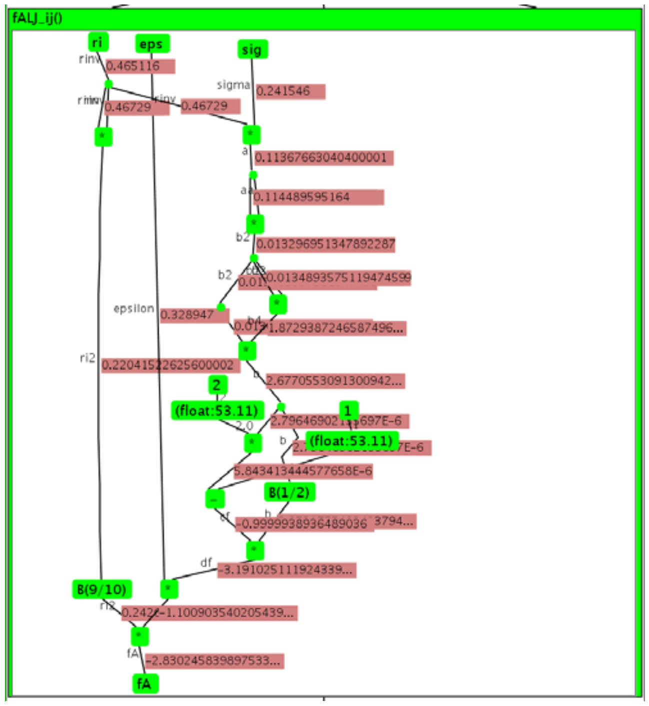

Figure 4: Data dependency graph of the 15-step pipeline for the calculation of the force factor (visualization with Mitrion).

Application Performance Results

Reducing the wall-clock time is the only issue application scientists are interested in. For a fair comparison of the application performance with and without FPGAs, the time measurement must include the overheads associated with additional housekeeping (e.g. loading the bitfile into FPGA), and the communication time between host and FPGA.

Table 3 summarizes the wall-clock times for various processing phases in one step for the calculation of the pair wise interaction energies and the corresponding force factor for about 28600 pairs. All timing measurements are based on the PAPI 3 [16] wall-clock time or mflops functions. The reference and host application was compiled with Portland Group PGI 5.2 version at optimization level "-O3 –fastsse" on our Cray XD1 system running S/W release v.1.3-45. The timing data is averaged over multiple runs.

[2] 1 cycle corresponds to 10 ns in Mitrion based 100 MHz designs.

Table 3: Wall-clock times (cumulative) for various code blocks in the subroutine `nforce` for non-bonding properties, processing 28599 interactions.

Version	nforce code block	time [ms]
non-FPGA	- loops	21.51
	call	21.61
1.1.1 03/06	- init	0.61
	- - wdta	0.61
	- - run	2.40
	- - rdta	43.50
	- - update	0.63
	- loops	69.45
	call	70.20
total speedup: 0.3		
1.1.5 05/06	- init	0.86
	- - run	0.77
	- - update	0.73
	- loops	19.82
	call	20.81
total speedup: 1.1		

The average time of the non-FPGA for the calculation of the non-bonding data, i.e. pair wise interaction energy and force factor, is 21 ms.

With Mitrion/Mithal 1.1.1, the FPGA runtime (*run*) is about 2.4 ms for 28600 particle interactions[3]. The additional overhead for housekeeping (*init*, *update*) is relatively small. The data transfer from host memory to SRAM (*wdta*) takes 0.615 ms, corresponding to 558 Mbyte/s. As the table shows, the data transfer back from the SRAM to host memory (*rdta*) kills the speedup achieved by the computational part on the FPGA. It results in a performance decrease by a factor of 3.2 for the total subroutine call (*call*). The poor communication performance from the FPGA to the Opteron is due to the limited read performance [14] (ref. Table 1).

With Mitrion/Mithal 1.1.5, a notable improvement of the communication performance was achieved by using the FPGA transfer region to store the results. The FPGA runtime (*run*) includes the data transfer operations and corresponds to a sustained performance of 558 Mflop/s. At the time of writing it is not clear why the total time for processing the interactions (loops), which includes the code blocks *run* and *update*, is still large. The overall performance is now slightly better than the non-FPGA version.

[3] Note that 15*28600/2.4ms = 178 MFlop/s is a factor 10 below the estimate of the Mitrion simulator.

6. Using Mitrion-C in Education

To promote the usage of parallel FPGA hardware, we conducted an experimental student project seminar in the winter term 2005/06. The seminar was targeted at the more advanced MSc students with knowledge on computer architecture and HPC programming. After the initial exploration of the Cray XD1 hardware and the Mitrion-C environment the students were offered to implement either of three applications: the above described Lennard-Jones interaction, a variant of the Smith-Waterman sequence alignment algorithm, and a kernel to solve the n-puzzle.

The students could implement these kernels according to their own concepts. Interestingly, one group decided to implement the LJ code with a fixed point high-precision mode while another used 32bit floating point. In the project seminar, the availability of the simulation software was found most beneficial, since compilation and synthetisation took very long and resources were limited. For a forthcoming course, it is planned to implement truly parallel FPGA kernels, that communicate directly via HyperTransport without going through the host processor. This is, however, not possible with the current Mitrion-C.

7. Conclusion

The Mitrion-C product targets the community of HPC application developers who want to implement algorithms on FPGA by means of a high-level language. It is a lively evolving package that supports the implementation of algorithms on FPGA without the need to learn and apply a hardware description language such as VHDL, which is normally unfamiliar to HPC users. For beginners, the learning curve of Mitrion-C is fairly steep. With some experience in writing Mitrion-C code it is possible to implement computational kernels within a couple of hours.

The availability of the simulator is greatly appreciated by our users. It is used to verify Mitrion-C designs without the need to generate bitfiles, a time- and memory consuming step.

Lessons Learned

- The high-level Mitrion-C interface allows application programmers to implement numerical algorithms on FPGA within a couple of hours. The data dependency graph viewer is an effective tool to trace and analyse data streams.
- The Mitrion simulator substantially reduces the time used for the implementation and validation

of algorithms. It greatly improves the productivity of software development.

- The data transfer bandwidth between host memory (DRAM) and memory closely attached to the FPGA device (SRAM banks) is a critical issue and can limit the overall performance.
- One major concern are the limited resources on the FPGA. The implementation of floating-point intensive computational kernels with Mitrion-C may fail due to limited functional components (BRAM, multiplier) on FPGA or constraints. This is especially painful on the elder and smaller devices such as our XCVIIPro50.

8. Acknowledgments

We are especially grateful to Stefan Möhl, Pontus Bergendahl, and Johan Lövdahl, all from Mitrionics A.B., for their guidance in using Mitrion-C, their efforts to improve the software and their support. We would like to thank Steve Margerm and Dave Strenski, all from Cray Inc., for their help and support in understanding technical details of the Cray XD1 system. Thanks also to the participants of our project seminar "FPGA programming for bioinformatics applications on the Cray XD1".

9. About the Authors

Thomas Steinke is leader of the BCB Junior Research Group "Alignment and Threading on Massively-Parallel Computers", Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustrasse 7, D-14195 Berlin-Dahlem, Germany, email: steinke@zib.de.

Alexander Reinefeld heads the computer science department at ZIB and holds a chair for Parallel and Distributed Systems at the Humboldt-Universität zu Berlin. He can be reached at ZIB (www.zib.de), email: ar@zib.de.

Thorsten Schütt is a research consultant and PhD student at ZIB; his email is: schuett@zib.de.

10. References

[1] H. Simon et al., *Science-Driven System Architecture: A New Process for Leadership Class Computing*, Journal of the Earth Simulator, vol. 2, March 2005, 2–10.

[2] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takeda, A. Konagaya, *Protein explorer: A petaflop special-purpose computer system for molecular dynamics simulations*, in: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, New York, ACM Press, 2003

[3] http://mdgrape.gsc.riken.jp/modules/tinyd0/

[4] R. Scrofano, V.K. Prasanna, *Computing Lennard-Jones Potentials and Forces with Reconfigurable Hardware*, International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), June 2004

[5] V. Kindratenko, D. Pointer, *A case study in porting a production scientific supercomputing application to a reconfigurable computer*, In: *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*, April 2006, Napa, California

[6] Stefan Möhl, The Mitrion-C Programming Language, v. 1.0.1-100, Mitrionics A.B., Lund, 2006

[7] A. Dellson, *Turning FPGAs Into Supercomputers — Debunking the Myths About FPGA-based Software Acceleration*, CUG 2006

[8] Mitrion rel. 1.1.5, April 2006, Mitrionics A.B. Lund, Sweden

[9] Xilinx Integrated Software Environment (ISE) 7.1-4i, 2005, Xilinx Inc.

[10] Cray XD1 FPGA Development, S–6400–131, Cray Inc, 2005

[11] The Mitrionic Host Abstraction Layer API, 1.1.0-001, Mitrionics A.B., 2006

[12] Th. Steinke, *ffpga interface*, ZIB, 2006

[13] Stefan Möhl , private communication, Mitrionics, Mar 2006

[14] Steve Margerm, private communication, Cray Inc., May 2006

[15] R. Haberlandt, S. Fritzsche, G. Peinel, K. Heinzinger: *Molekulardynamik – Grundlagen und Anwendungen*. Vieweg Verlag, Braunschweig/Wiesbaden, 1995

[16] PAPI - Performance Application Programming Interface; Browne, S., Dongarra J., Garner N., London K., and Mucci, P., *A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters*, Proc. SC'2000, November 2000, http://icl.cs.utk.edu/papi/