TECHNISCHE UNIVERSITÄT DRESDEN

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF COMPUTER ENGINEERING
CHAIR OF COMPUTER ARCHITECTURE
PROF. DR. WOLFGANG E. NAGEL

Hauptseminar
"Rechnerarchitektur und Programmierung"

Adapteva Parallella: Crowd-funded Low-budget Open-source HPC

Tobias Frust
(Mat.-No.: 3749068)

Professor: Prof. Dr. Wolfgang E. Nagel
Tutor: Ronny Brendel

Dresden, September 21, 2015

Contents

1 Introduction

Beginning in 1971, when the first commercial microprocessor was released, to 2003, performance improvements have mainly been accomplished by increasing the clock frequency. By boosting the clock frequency, heat dissipation increases as well. This is why that measure is not applied anymore and the clock rate remained static. Hardware developers needed to find different ways how to speed up their processors. The solution was to integrate more than one processor on a chip. Nowadays, any device, be it a desktop pc, a laptop or a smartphone, has a multi-core CPU architecture. Attempts to achieve better speed resulted in various design methodologies. For this reason, parallel programming is necessary to exploit the maximum performance, whereas in the past, implementations increased their performance with every processor generation due to their serial speedup.
In this work, the Adapteva Parallella, a crowd-funded low-budget open-source High-Performance Computing (HPC) Platform, is presented. In Section 2 the history of the company Adapteva is presented. Furthermore, the architecture and characteristics of that board are introduced with a special focus on the Epiphany coprocessor. Section 3 shows how to implement the one-dimensional Fast Fourier Transform (FFT) on the Epiphany processor. A special focus will be the data transfer from host to device. Finally, the performance of this implementation will be measured. In Section 4 the Epiphany chip is compared to another many-core architecture: NVIDIA's Maxwell GPU architecture. To stay fair, the comparison is based on energy efficiency with respect to chip area. The last section will summarize the results and assesses the Parallella's potential as a HPC platform.

2 Adapteva Parallella

In this section Adpatava's company history and its Parallella board are presented. Furthermore, the architecture of the Epiphany chip is shown.

2.1 Company History

Adapteva was founded on February 2008 with the goal to develop and build the world's most energy efficient microprocessor chips and single board computers. A few months later in October, the first prototype layout of Epiphany core was completed. The tape-out of the Epiphany-I prototype in 65 nm structure width was in June 2009 and they were able to secure a 1.5M US $ Series-A funding from Bittware. Release of the first product, the Epiphany-III 65nm 16-core chip, was in May 2011. Tape-out of the first 64-core version in 28 nm structure width was in August 2011 and in the same year Adapteva reached break-even as a company. One year later in August, 50 GFLOPS/Watt energy efficiency could be demonstrated with the sampled product. The Parallella Kickstarter project was launched in October 2012 and the first prototypes were shipped by the end of the year. Dispatch of the Parallella Board to the Kickstarter backers began in July 2013 and was finished in April 2014. [4]

2.2 Parallella Kickstarter Program

On September 28, 2012 Adapteva launched the Parallella Kickstarter project. They were asking for help, because getting corporations to buy into parallel computing has proven challenging. Based on this knowledge they said, that the only way to create a sustainable parallel computing platform is a broad movement.
The goal was to implement a design that is based upon the following principles [3]:

- **Open Access:** no special access or non-disclosure agreements (NDA) shall be necessary. All architecture and SDK documents shall be available online.
- **Open Source:** Parallella platform shall be based on free and open-source development tools. All board design files will be open-source once the Parallella board is released.
- **Affordability:** The Parallella board shall not cost more than 100 $. The barrier for customers, who want to develop high performance applications, shall be removed.

They wanted to democratize access to parallel computing, close the knowledge gap in parallel programming and build the most energy-efficient computing architecture. The Kickstarter project gained 898,921$ from 4,965 backers. Basis of the Parallella board was the Epiphany chip, developed by Adapteva since 2008 [3].

2.3 Parallella Board

The Parallella board was released during a Kickstarter project and is presented in the next section.

2.3.1 General Overview

The Parallella board is a credit card-sized, high performance computer based on Adapteva's Epiphany multi-core chips (Figure 1).

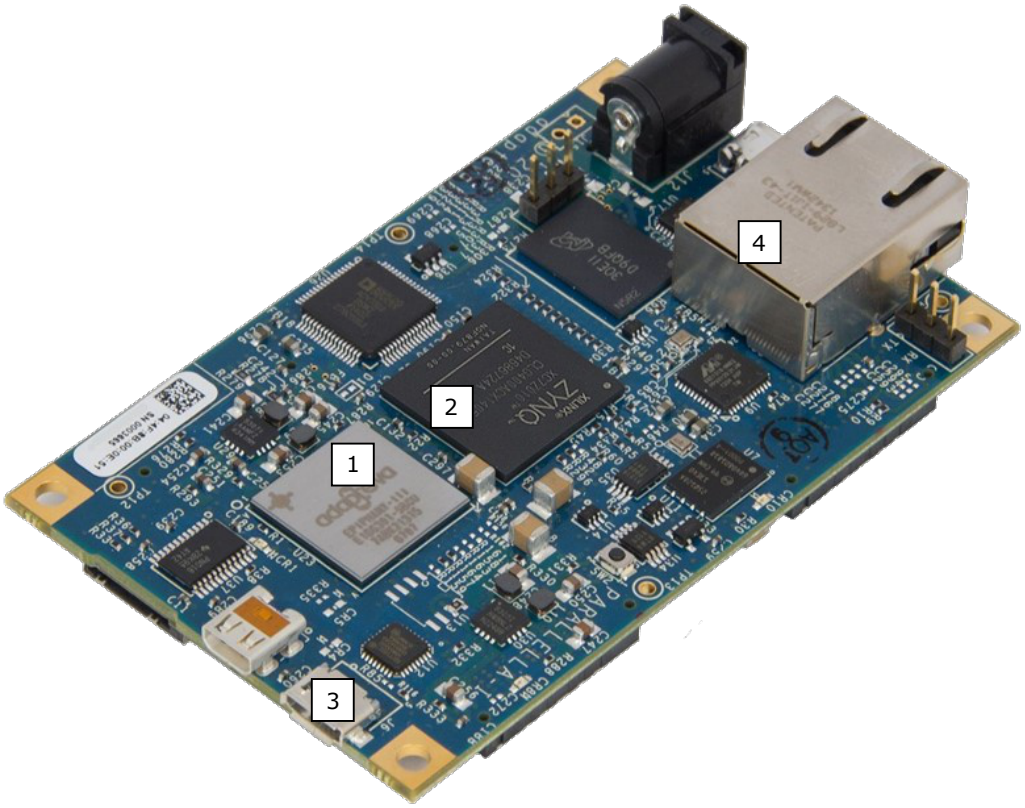


Figure 1: The Parallella Board [12].

The main elements of this board are the dual-core ARM Cortex A9 host processor (2) and the 16-core Epiphany coprocessor (1). It includes one Gigabyte RAM, Gigabit Ethernet (4) and a micro SD card (3) as storage. The operating system is Linux, e. g. Ubuntu. The Epiphany chip has a peak performance of 26 GFLOP/s in single-precision, and has an energy consumption of two Watts. Figure 2 shows the system overview of the Parallella. It consists of two parts, the Zynq host processor with a dual-core ARM A9, and a programmable logic. This FPGA is a configurable interconnect network, which is the connector between the host and the Epiphany coprocessor. This example shows the configuration with 64 cores. The interconnection bandwidth from ARM dual-core to the Programmable Logic is denoted with 2.4 GB/s, and between Epiphany and programmable logic 1.4 GB/s. Later in this work the transfer bandwidth will be examined in detail (Section 3.2).

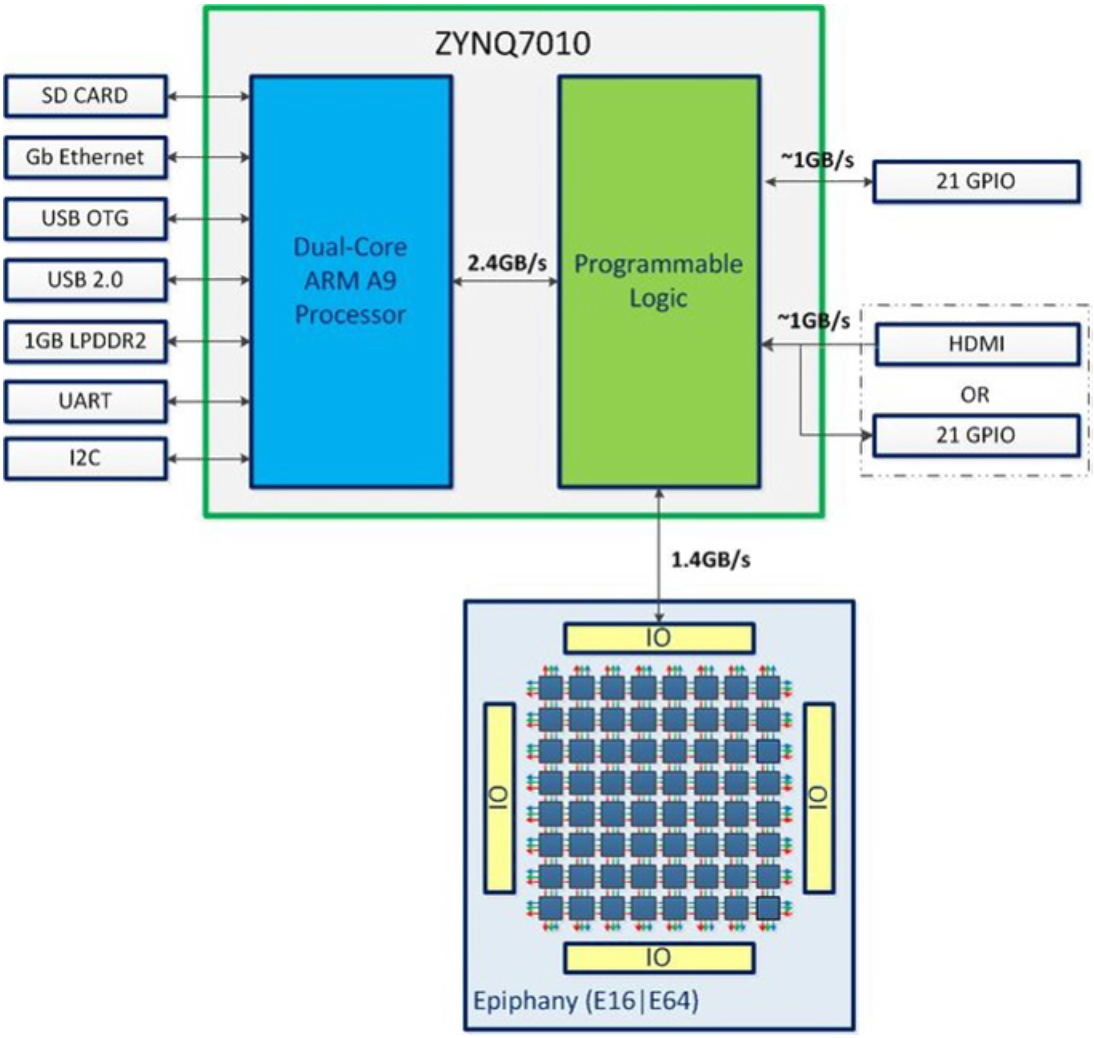


Figure 2: System overview of the Parallella Board [2].

2.3.2 Epiphany Architecture

Adapteva assumes that current architectures are hitting a wall and that the future of computing is parallel and heterogeneous [2]. Therefore the Epiphany architecture, which serves as coprocessor to ARM or Intel CPUs, was released. Figure 3 shows the system overview of the 64-core version of the Epiphany chip. The cores are arranged in a quadratic mesh. Any core is only connected to its direct neighbors in east, south, west and north direction. Each processing element is a RISC design with in-order execution. It consists of one integer ALU, a floating-point unit with support for fused multiply-add and a 64-word register file. Every core has its own local memory to store program and data in. The size of this memory is limited and amounts in 32 Kilobytes per core in case of the 16-core Epiphany. It is separated into four eight KB memory banks. For this reason, those resources need to be treated carefully and data partitioning is necessary. Optimal performance can only be achieved if data is placed in local memory. The Epiphany chip features a magic address translation, which means, that any core can access all memory locations, be it transfers to DRAM, other cores or off-chip.

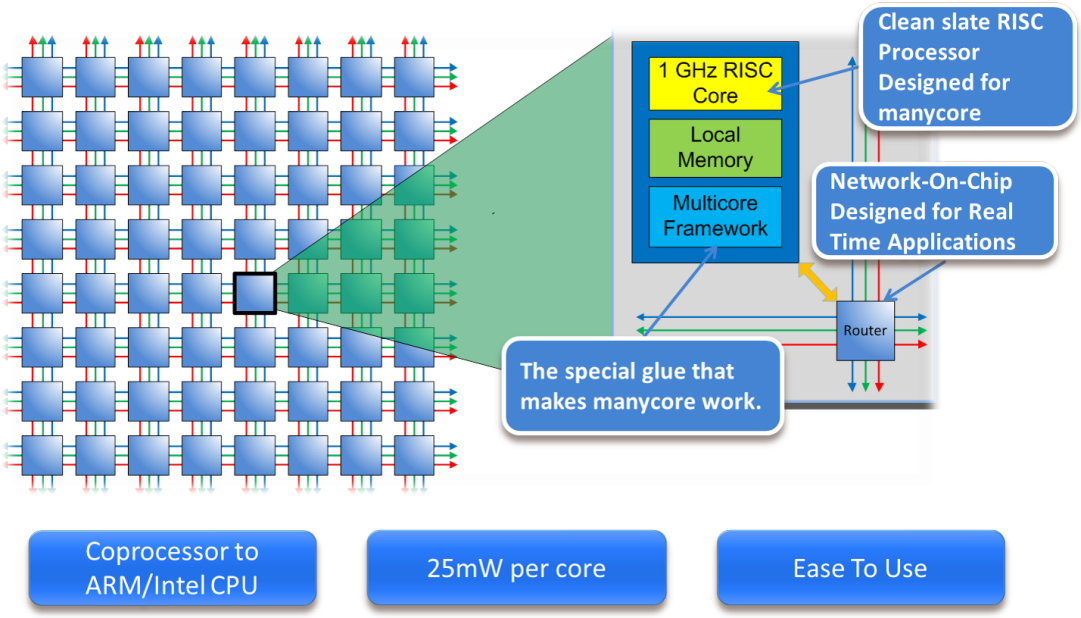


Figure 3: The Epiphany Architecture [2].

eMesh Network on Chip

All Epiphany cores are connected by a 2D mesh topology with only nearest neighbor direct connections in east, south, west and north, that is called eMesh Network on Chip. Figure 4 shows the structure of this network. There are three separate mesh structures, that serve different types of transactions. The *cMesh* is used for on-chip write transactions. Its maximum bidirectional throughput is eight bytes/cycle in each direction. The *rMesh* is used for all read requests. The maximum throughput is one read transaction every eight clock cycles in each routing direction. The *xMesh* is used for off-chip write transactions and for passing through transactions destined for another chip in a multi-chip system. The maximum throughput depends on the available off-chip I/O bandwidth. Write transactions move through the network with a

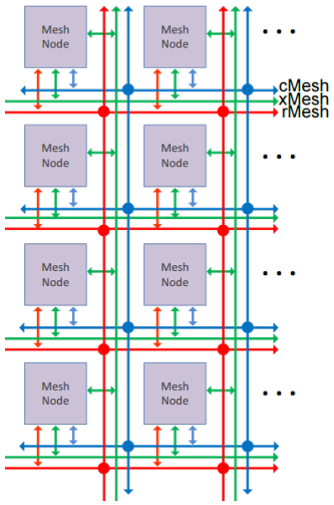


Figure 4: Structure of the eMesh Network on Chip [1].

latency of 1,5 cycles latency per hop. In a 64-core configuration the connection from the left to the right edge would thus take twelve clock cycles. The *cMesh* has lower latency than the *xMesh*. This is why tasks with significant inter-core communication should be placed together on the same chip for better performance [1].
Furthermore, write transaction are optimized over read transactions. Writes are approximately 16 times quicker than reads for on-chip communication. The network is free of deadlocks, because of the separation of read and write meshes together with a fixed xy-routing scheme. The network can be scaled to very large arrays, it is only limited by the size of the address space [1].

3 One-Dimensional Fast Fourier Transform

In the course of this work the one-dimensional Fast Fourier Transform (FFT) is implemented and parallelized on the Parallella board. The FFT is a very important algorithm in many fields, e.g. signal processing. The filtered backprojection is an algorithm to reconstruct images from projections. In computed tomography, there are rays traversing a measurement field from the source to the detector. The detector measures the intensities of the rays after interacting with the measurement area. To obtain the original image it is reconstructed from the set of projections. As Figure 5 shows, the filtered backprojection consists of two parts: filtering and backprojection. Filtering can be realized in two different ways. The first possibility is to convolute with the filter function in time domain. The second approach is to convert a signal from the time domain to a representation in the frequency domain via the FFT. Then, the signal can be multiplied with the filter function element-wise. Next, it needs to be reconverted via the Inverse FFT. To obtain the measured cross-section it finally needs to be back-projected.
In this section it will be shown how the FFT is implemented and parallelized on the Parallella board. Finally, performance measurements will be presented.

3.1 Implementation

The FFT is a widely used algorithm, thus it makes sense to examine its performance on this architecture. The implemented program works as shown in Figure 6. The input data is stored as TIF image. The first step is to read the image. For this, the so called *LibTiff*-Library [15] is used. Once, the data is in main memory, it can be transfered to the local memory of each used Epiphany core. Since the chip contains 16 cores, there are 16 memory transfers from host to device memory. Each transfer copies two lines of the image to the Epiphany. Then, the parallel computation can start. The first step is to calculate the Fast Fourier Transform of each line. Every element is multiplied with the filter function. Finally, the inverse FFT converts the data back to the domain domain. Once the computation is finished, the data is transferred back from local memory to the main memory. Via the *LibTiff*-Library [15] the filtered data is

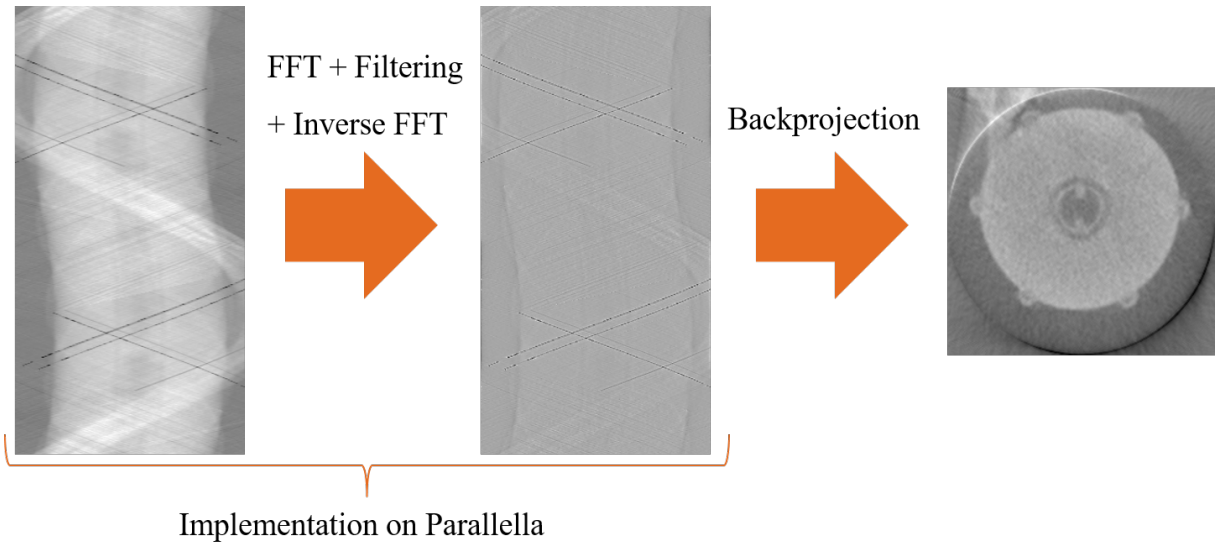


Figure 5: Principle of the Filtered Backprojection Algorithm.

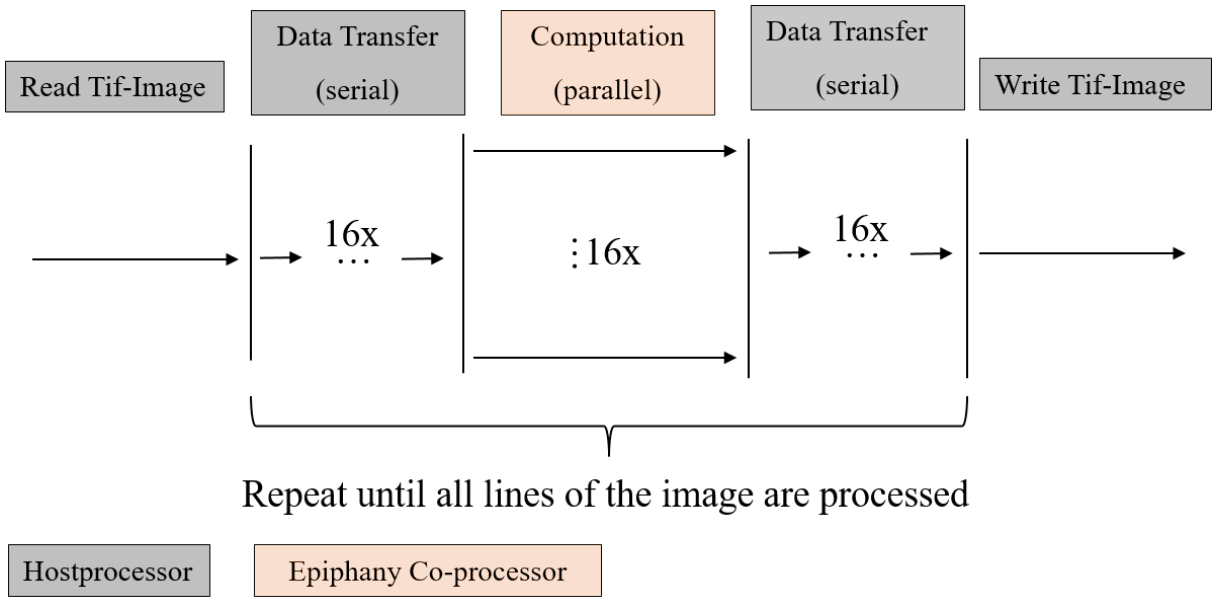


Figure 6: Workflow of the filtering procedure.

written back to the TIF files.

3.2 Optimization

To run an application on the Epiphany device you need to initialize it. Figure 7 shows how this is done. The first step is to reset the Epiphany device. Afterwards, a workgroup needs to be opened (line 5) and finally the executable needs to be loaded into the group (line 11). The next step is to copy data from host memory to the local memory of each core. Once copying is finished, computation can start. Due to the limited amount of per-core local memory, only few image lines can be computed in parallel. Copying and computation has to be repeated, until the whole image is computed. In a first approach this could be realized by repeating the whole initialization procedure again, but this is very time consuming.
This is why it is realized differently. The Epiphany is started only once and runs in continuous loop. Any time, data copy is finished, the Epiphany gets a signal, that computation may start. Without this intuitive approach the hardware could not be utilized efficiently.

Optimization of Data Transfer

During the process of implementation and optimization a severe bottleneck was identified: the data transfer. Figure 8 shows the portion of the computation part versus the portion of the read and write part. It shows, that the data transfer consumes more than one third of the overall execution time which, is a big drawback. Another peculiarity is that the read time is more than twice the write time. To examine the reasons for that peculiarity, measurements concerning the memory bandwidth had been performed. One idea to explain the low transfer rates was that only a small amount of data was copied. Figure 9 shows the measured values of the memory bandwidth for different data sizes. It is shown, that the maximum transfer rate of 15 Megabytes per second is already reached with 128 byte of copied data and the read bandwidth is less than half the write bandwidth. These maximum values are equal to those measured in the implementation. So the reason for the limited transfer rate is not the size of transfered

```
e_init(nullptr);
e_reset_system();

//start workgroup with all e-cores
if(e_open(&dev, 0, 0, Y, X) != E_OK){
    printf("e_open failed\n");
    exit(1);
}

//load epiphany program into workgroup, but don't start it yet
if(e_load_group(epiphanyExecutable, &dev, 0, 0, Y, X, E_FALSE) == E_ERR){
        printf("epiphany program could not be loaded into workgroup\n");
        exit(1);
}
```

Figure 7: Initialization procedure of the Epiphany device

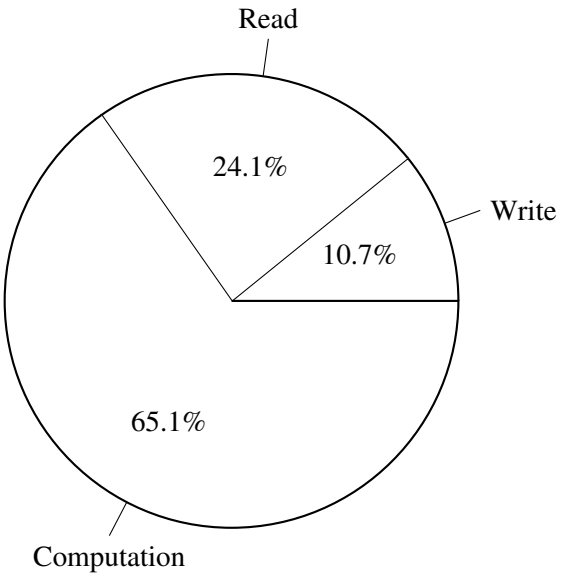


Figure 8: Ratios of time consumed by data transfer versus computation (execution on one core).

data. Adapteva offers an own bandwidth test. Its output is presented in Figure 10. The first and second values represent the same transfer, from host memory to the Epiphany chip, as performed in Figure 9. Adapteva's measurements are equal to the results of this work. A closer look at the measurements reveals, that there is a different way to achieve a higher memory bandwidth. If you use a special memory, called ERAM, as a temporary storage between host and Epiphany device, a higher transfer rate can be achieved. The memory bandwidth from host memory to ERAM is 75 MB per second for write and 105 MB per second for read transactions. Data copies from ERAM to an Epiphany core result in a transfer rate of 240 MB per second for write and 88 MB/s for read. The results show, that there exists a possibility to reach higher transfer rates. Nevertheless, the results are far from the number presented in Figure 2, where the bandwidth is denoted with 1.4 GB/s for transfers to the Epiphany chip. Furthermore, there is no documentation which explains how to achieve the maximum rates.

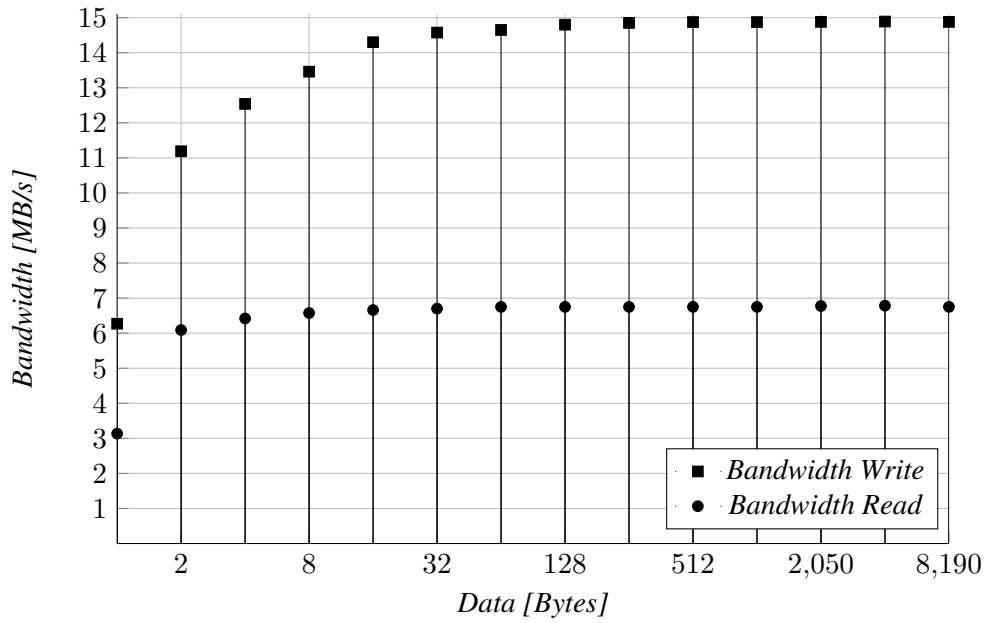


Figure 9: Measurement of memory bandwidth for different data sizes.

```
ARM Host     --> eCore(0,0) write spead          =   14.13 MB/s
ARM Host     --> eCore(0,0) read spead           =    6.55 MB/s
ARM Host     --> ERAM write speed                =   75.14 MB/s
ARM Host     <-- ERAM read speed                 =  105.46 MB/s
ARM Host     <-> DRAM: Copy speed                =  194.72 MB/s
eCore (0,0) --> eCore(1,0) write speed (DMA) = 1280.04 MB/s
eCore (0,0) <-- eCore(1,0) read speed (DMA)  =  390.01 MB/s
eCore (0,0) --> ERAM write speed (DMA)       =  240.24 MB/s
eCore (0,0) <-- ERAM read speed (DMA)        =   87.61 MB/s
```

Figure 10: Results of Adapteva's bandwidth test.

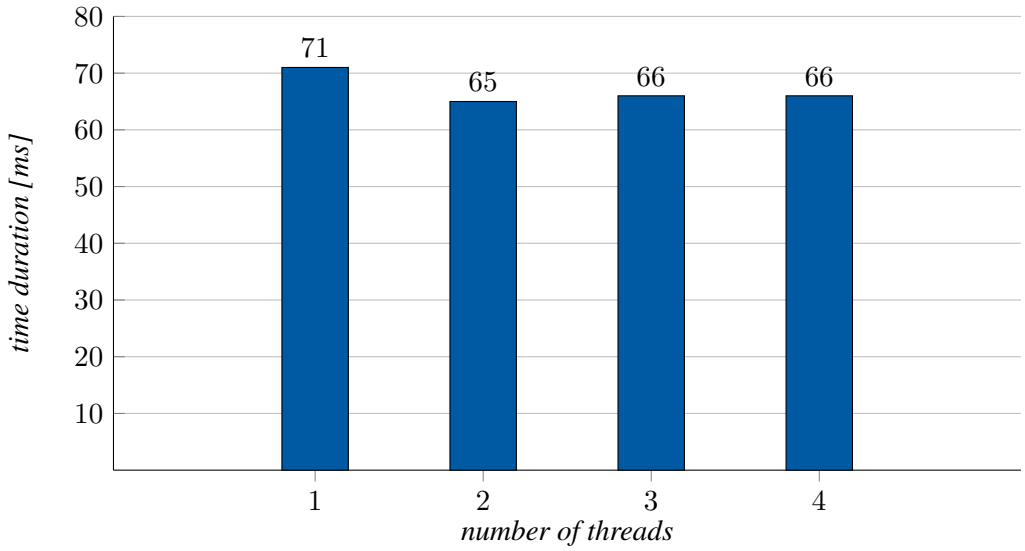


Figure 11: Time duration of data transfer with OpenMP and variable number of threads.

According to Adapteva's bandwidth test, the memory transfer from host to device should be accelerated by using the ERAM as a temporary memory. Due to technical problems and lack of good documentation, this measure could not be implemented successfully.
Another approach to improve the memory bandwidth is to use OpenMP to parallelize data transfer to the device. Figure 11 shows the time improvements that can be achieved by the use of multiple threads. The total transfer time can be reduced by some ten percent if two threads are used. If more than two threads are used, the results are still better compared to the solution with one thread, but there is no further acceleration.

3.3 Performance Measurements

To identify bottlenecks and to examine the performance of the implementation, measurements have been done. Table 1 gives an overview about the most important measured metrics. Again, the results show

	1 core	16 cores
Write Time	65 ms	65 ms
Computation Time	394 ms	25 ms
Read Time	146 ms	148 ms
Number of Floating Point Operations	$13.9 \cdot 10^6$	$13.9 \cdot 10^6$
MFLOP/s in Computation Part	35 MFLOP/s	556 MFLOP/s
MFLOP/s in Whole Program	23 MFLOP/s	57 MFLOP/s
Transfer Rate: Write	15.4 MB/s	15.4 MB/s
Transfer Rate: Read	7.01 MB/s	6.9 MB/s

Table 1: Absolute numbers of measured metrics.

that the main bottleneck is the data transfer. Especially in the 16-core version the portion of computation amounts in only ten percent. On the other hand the speedup of the computation part is close to 16 and therefore nearly optimal. If only the computation part is considered, 556 MFLOP/s could be reached, which is about two percent of the theoretical peak performance of the Epiphany chip. The implementation of the FFT includes the calculation of a sine function, that is very time consuming. This measure might be increased, if the sine is replaced by precalculated values in a table. On the other hand the size of local memory and the transfer rates are limited. For this reason, there needs to be a verification if the reduced computation time justifies the increased time for the data transfer of the sine values. The drawbacks could be avoided, if the trigonometric functions would be realized in hardware.

Speedup

The Speedup of the implementation is calculated according to Formula 1 [8]. T_1 is the time for the serial version and T_p the time for the parallel execution time with p processors.

$$S_\mathrm{p} = \frac{T_1}{T_\mathrm{p}} \tag{1}$$

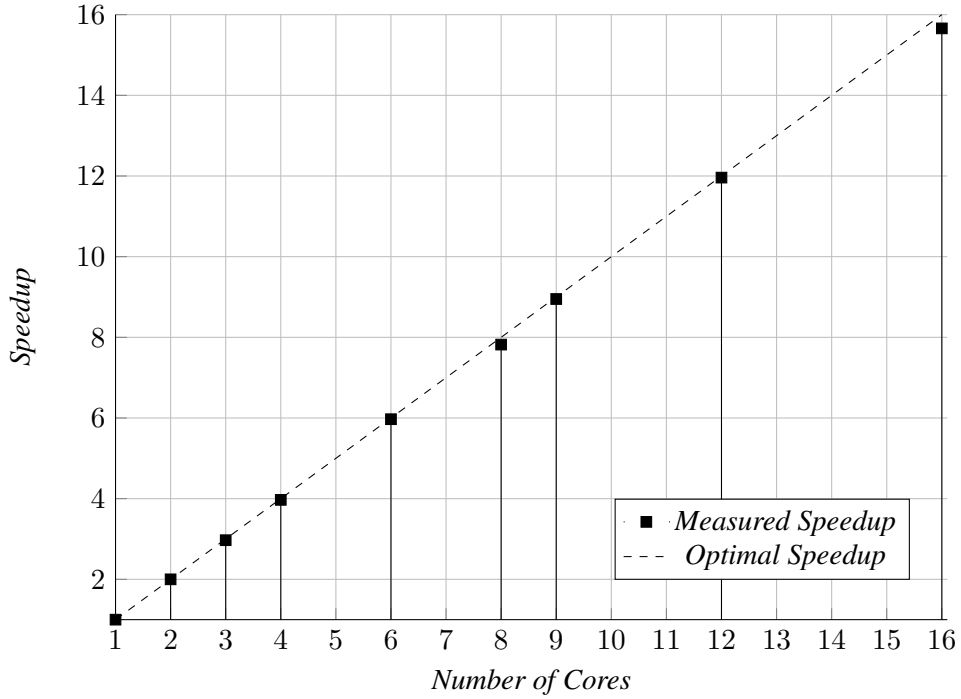


Figure 12: Speedup of the computation part in comparison with optimal speedup.

Figure 12 shows the speedup of the computation part in comparison to the optimal speedup. The values could only be measured for multiples of two and three. The Epiphany Software Development Kit offers functions for building groups of threads that work in parallel. These groups can only be rectangular. The cores of the Epiphany are structured in a mesh with four cores in x- and y-dimension. This is the reason why it was not possible with reasonable effort to execute the program with an arbitrary number of cores. Figure 12 shows that the measured speedup is nearly linear, if only the computation part is considered. This corresponds to the expected behavior, because there is no communication necessary between the different cores and thus, they do not influence each other.

Figure 13 shows the speedup of the overall program if the data transfer is considered, too. The data transfer could not be parallelized and therefore the parallel fraction is only 65%. The optimal speedup according to Amdahl can be calculated according to Formula 2 [5].

$$S_{pA} = \frac{1}{(1-f) + \frac{1}{p}f} \tag{2}$$

f is the parallel fraction, p the number of processors and S_{pA} the speedup according to Amdahl. The maximum speedup according to Amdahl is thus 2.9. Figure 13 presents the measured values in comparison with the maximum speedup for the number of cores according to Amdahl. Again, the measured values are very close to the calculated optimal speedup. This matches the expected behavior. In the parallel part the cores do not need to communicate or access shared memory locations. This is why the speedup correlates very well with the expectations.

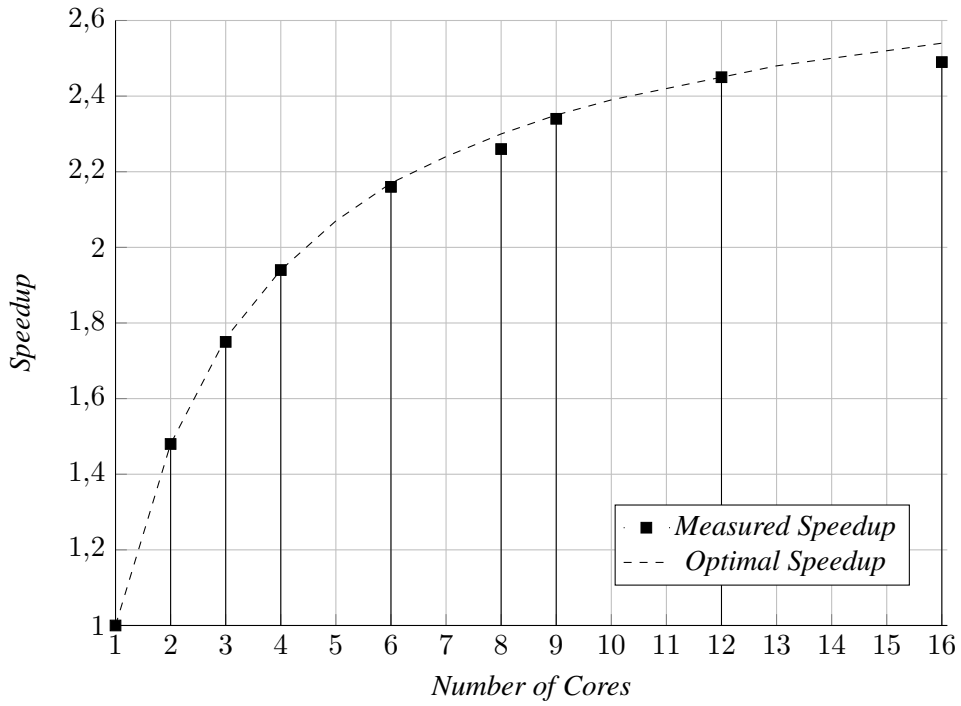


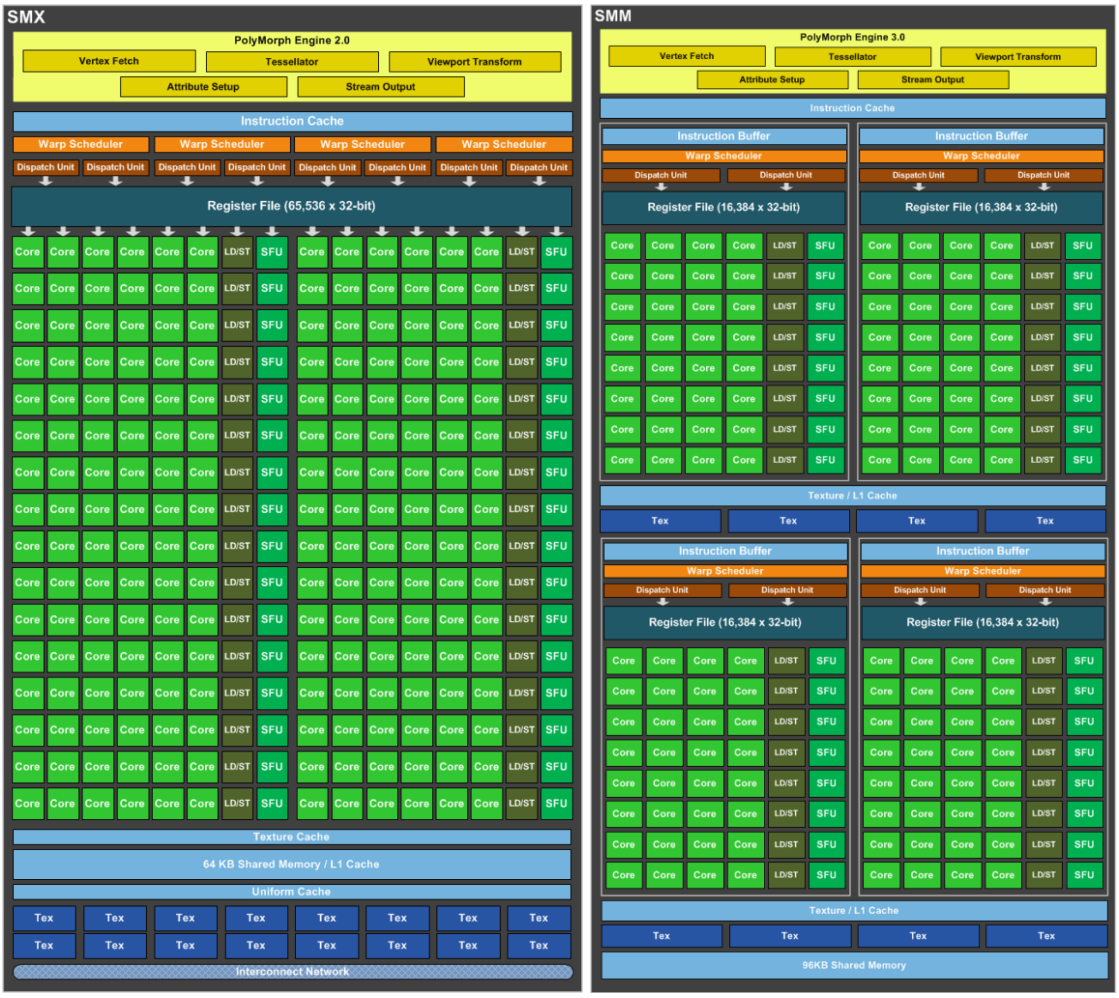
Figure 13: Overall Speedup in comparison with the optimal speedup according to Amdahl.

4 Comparison with NVIDIA GeForce GTX 750 Ti

In this section the Parallella board is compared with the NVIDIA GeForce GTX 750 Ti in terms of energy efficiency and die size. The architecture of a system with a CPU, the host processor, and a GPU, the device, is similar to the architecture of the Parallella board. There, the host is the ARM processor and the device is the Epiphany chip.

4.1 Maxwell GPU Architecture

The Maxwell GPU Architecture is said to have twice the energy efficiency of that of the Kepler GPU architecture due to the new design of the Streaming Multiprocessor (SMM). The new datapath organization and improved instruction scheduler provide more than 40% higher delivered performance per CUDA core. Figure 14 shows the Streaming Multiprocessors of Maxwell and Kepler. Immediately it can be seen, that there is a significant difference in the layout between the SMX and the SMM. The SMM is now heavily partitioned. In an SMX the four warp scheduler would share most of their execution resources whereas in an SMM the warp schedulers are removed from each other and given complete dominion over a far smaller collection of execution resources. Now they do not need to share CUDA cores,



(a) Kepler Streaming Multiprocessor (SMX) (b) Maxwell Streaming Multiprocessor (SMM)

Figure 14: Block diagram of Kepler and Maxwell Streaming Multiprocessor [13].

load/store units or special function units, because each of them is duplicated in each partition. Besides the goal to achieve better energy efficiency, NVIDIA wanted to increase the space efficiency. The SMM can achieve 92% of the performance with only 128 cores compared to 192 cores in an SMX. [7][13]

4.2 NVIDIA GeForce GTX 750 Ti

The NVIDIA GeForce GTX 750 Ti is one of the first releases of devices with the Maxwell GPU architecture in the high volume market. This GPU includes 5 SMMs with 640 CUDA cores and 2 MB of L2 cache. It is paired with 2 GB of GDDR5-Memory on a 128 Bit bus.

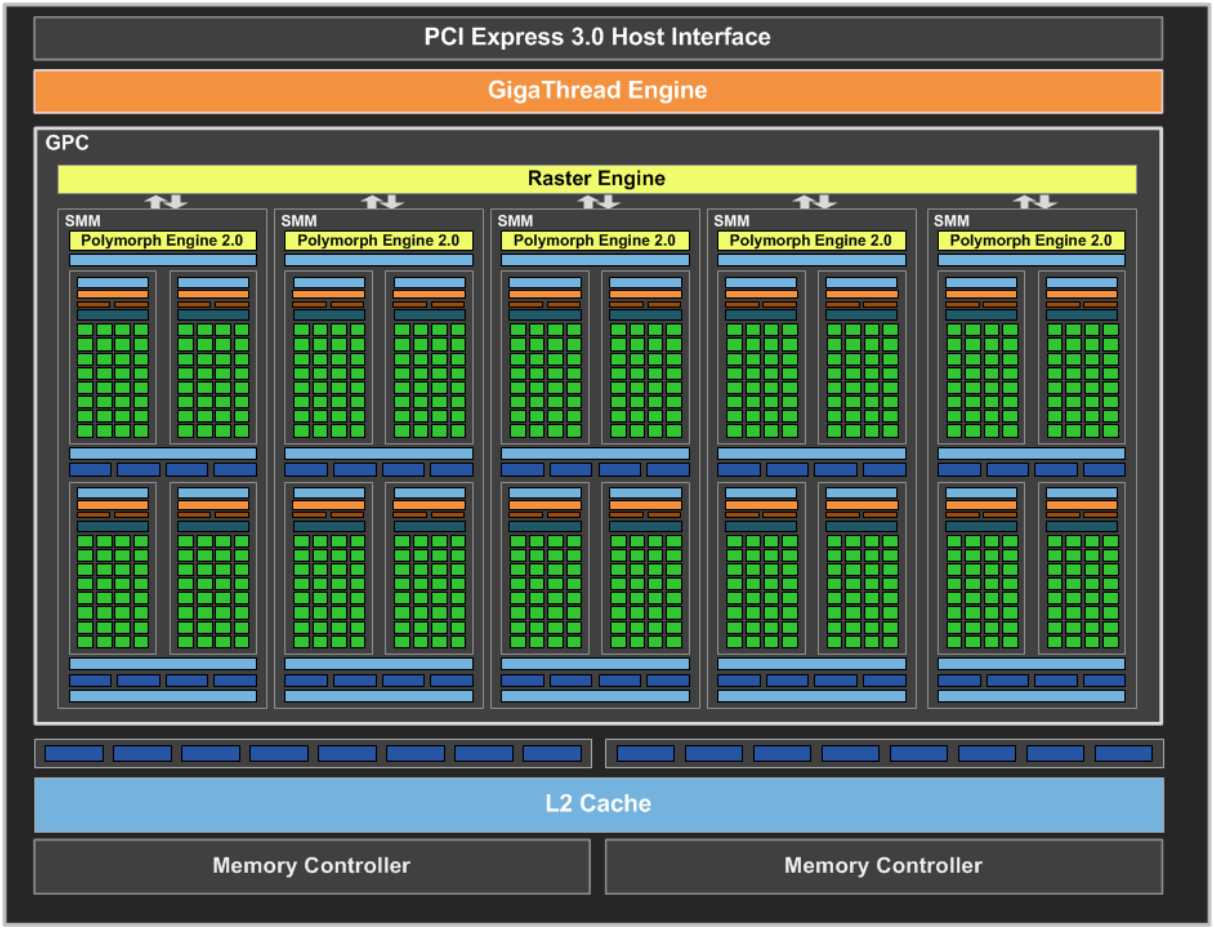


Figure 15: Block diagram of the Nvidia GTX 750 TI [13].

The most important specifications of the NVIDIA GeForce GTX 750 Ti are summarized in Table 2. It has a single-precision peak performance of 1,472 TFLOP/s. With a Thermal Design Power of only 60 Watts, the card does not necessarily need an additional power supply and can be fed only by PCI-Express. Nevertheless, many vendors add an additional power supply.[13]

Peak Performance Single Precision	1,472 TFLOP/s
Number of SMMs	5
Number of CUDA cores	640
Memory Bandwidth	86.4 GB/s
Architecture	Maxwell
Memory Bus Interface	128 Bit
Manufacturing Process	TSMC 28nm
Thermal Design Power	60 Watt
Launch Date	02/08/2014
Die size	148 mm^2

Table 2: NVIDIA GeForce GTX 750 Ti GPU Specification [13].

4.3 Comparison of Parallella and NVIDIA GeForce GTX 750 TI

The Epiphany chip and the a GPU are compared in terms of energy efficiency and chip are in this section. The evaluation GPU is a NVIDIA GeForce GTX 750 TI, that was presented in section 4.2. The implementation of the algorithm is the same as presented in Figure 6. The implementation is based on CUDA, a parallel computing platform and application programming interface developed by NVIDIA. It allows to use a CUDA-enabled GPU for general purpose programming. The FFT is implemented using NVIDIA's cuFFT-Library [9].

Test Setup

To evaluate the energy efficiency of both the GPU and the Epiphany chip, the Thermal Design power is used. The energy consumption is calculated according to Formula 3.

$$E = t_{\text{runtime}} P_{\text{TDP}} \tag{3}$$

t_{runtime} is the runtime of the computation part of the implementation and P_{TDP} is the Thermal Design Power. The Thermal Design Power of the 16-core Epiphany is 2 Watts respectively 60 Watts for the NVIDIA GeForce GTX 750 Ti. Of course, the use of the Thermal Design Power is a simplification, but it was not possible to measure the energy consumption more precisely with reasonable effort. The function `clock_gettime` measures the execution times.

Energy efficiency

For the following examinations, only the computation part of the algorithm is considered. If the data transfer is taken into account, too, a fair comparison is not possible. The transfer rate of the Parallella is not comparable to those from CPU to GPU via PCI-Express. Thus, the following results are only valid under the assumption, that only the computation part is considered.
In Table 3 the energy consumption per picture of the Parallella and GPU implementation are presented.

Number of Pictures	Parallella	GPU
1	50 mJ	10,200 mJ
100	50 mJ	114 mJ
1000	50 mJ	35 mJ
5000	50 mJ	26 mJ
10000	50 mJ	25 mJ

Table 3: Energy consumption per Picture of the Parallella and GPU implementation.

The Parallella is already completely utilized with one picture, and thus the energy consumption per picture stays constant. On the other hand, the measurements show that the energy consumption per picture of the GPU implementation improves, if the number of pictures increases. If only one picture is calculated, the Parallella clearly outperforms the GPU in terms of energy efficiency. The reason is, that the initialization phase of cuFFT is very time consuming and the GPU is not utilized entirely. If more pictures are computed, the GPU beats the Parallella and provides a lower energy consumption per picture. The implementation on the Parallella can be improved, if for example the time consuming computation of the sine function is removed or further improvement techniques like loop unrolling are used. It can be stated, that in this implementation the energy consumption per picture is better on the GPU for a big amount of pictures and can be achieved with lower effort. On the other hand the Parallella implementation can be improved and thus, the energy consumption per picture. The optimization is more challenging compared to the GPU, due to the limited documentation and local memory size. Furthermore, the manufacturing process needs to be considered. The Epiphany is a 65 nm chip, whereas the GPU is manufactured in 28 nm. For this reason, the energy efficiency of the Epiphany improves, if the manufacturing process would be 28 nm.

Energy efficiency of a 1024-core Epiphany chip

In this section an example configuration of Adapteva for an Epiphany chip with 1024 cores [11], shall be compared to the GPU. Table 4 shows the specification of that chip. Due to the equal manufacturing process of 28 nm and the similar die size, both the 1024-core Epiphany and the GPU (Table 2) are comparable to each other.

Number of cores	1024
Manufacturing Process	28 nm
Max. Operating Frequency	700 MHz
Peak Performance (Single Precision)	1,408 GFLOP/s
Die Size	131 mm^2
Thermal Design Power	20 Watt

Table 4: Specification of an example configuration of an Epiphany chip with 1024 cores [11].

The energy efficiency is approximately six times better than that of the 16-core Epiphany used in this work. There, the energy efficiency per picture would be better than that of the GPU.

5 Conclusion

In this work the Adapteva Parallella board was presented. In the first section the architecture and specification of the board, as well as the structure of the Epiphany chip were shown. On this basis, the FFT algorithm was implemented. Furthermore, the performance was examined and the data transfer from host to device was analyzed in detail. Finally, the implementation was repeated for the NVIDIA GeForce GTX 750 Ti. Both solutions were compared to each other in terms of energy efficiency.
The results show, that the Parallella is very affordable. For only 99$ you get a credit card-sized standalone computer or an embedded device. This is the reason why this tool is nice for gathering experience in parallel programming. Institutions or interested people could buy this hardware at low cost and practice parallel programming. The Epiphany chip is energy efficient. Any core has its own small sized local memory, that is no managed cache; the user has full control over this memory. This is an advantage compared to architectures with a hardware managed cache, because you can optimize and control the access to that very fast memory. On the other hand the developer has more responsibilities and needs to put more effort into its memory accesses. Additionally there is no cache coherency latency. Especially for a large amount of cores this produces a large overhead, e. g. the Intel Xeon Phi [6].
On the other hand, the main disadvantage for real high-speed applications is the data transfer bottleneck. Due to the limited local memory size, for code as well as data, small memory transfers from host to device are necessary and should be as quick as possible. The measurements in this work show, that data transfer is time consuming and the main bottleneck. Furthermore, the Parallella has a small user base and an expandable documentation. For this reason, it is hard to do a good implementation in reasonable time and for beginners, a good documentation is essential, too.
All in all, the Adapteva Parallella is a board that is not yet ready for serious high-performance implementations. It might be well suited as an entrance to parallel programming, but for this a better documentation and support is necessary. The Epiphany is a good idea and an efficient architecture. It is unclear, if Adapteva's story can continue, but probably the ideas will live on in newer architectures.

References

[1] Adapteva. *Epiphany Architecture Reference*. Ed. by Adapteva. 2011. URL: http://www.adapteva.com/docs/epiphany_arch_ref.pdf.

[2] Adapteva. *Introduction*. Ed. by Adapteva. 2015. URL: http://www.adapteva.com/introduction/ (visited on 08/03/2015).

[3] Adapteva. *Parallella: A Supercomputer For Everyone*. Ed. by Adapteva. Sept. 28, 2012. URL: https://www.kickstarter.com/projects/adapteva/parallella-a-supercomputer-for-everyone/description (visited on 03/08/2015).

[4] Adapteva. *The Adapteva Story*. Ed. by Adapteva. 2015. URL: http://www.adapteva.com/about-us/the-adapteva-story/ (visited on 08/03/2015).

[5] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities." In: *AFIPS Conference Proceedings* (1967), pp. 483–485.

[6] Jianbin Fang et al. "An Empirical Study of Intel Xeon Phi." In: *CoRR* abs/1310.5842 (Dec. 20, 2013).

[7] Mark Harris. *Maxwell: The Most Advanced CUDA GPU Ever Made*. Ed. by NVIDIA. Sept. 18, 2014. URL: http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/ (visited on 09/12/2015).

[8] John L. Hennesy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 2012, pp. 46–47. ISBN: 978-0-12-383872-8.

[9] NVIDIA. *cuFFT*. Ed. by NVIDIA. 2015. URL: http://docs.nvidia.com/cuda/cufft/.

[10] A. Olofsson, T. Nordstroem, and Z. Ul-Abdin. "Kickstarting High-performance Energy-efficient Manycore AArchitecture with Epiphany." In: *arXiv preprint arXiv:1412.5538* (2014).

[11] A. Olofsson, R. Trogan, and O. Raikhman. *A 1024-core 70 GFLOP/W Floating Point Manycore Microprocessor*. Ed. by Adapteva. 2011. URL: http://www.adapteva.com/wp-content/uploads/2011/10/adapteva_hpec11.pdf.

[12] Parallella. *The Parallella Board*. Ed. by Parallella. 2014. URL: https://www.parallella.org/board/.

[13] Ryan Smith and T. S. Ganesh. *The NVIDIA GeForce GTX 750 Ti and GTX 750 Review: Maxwell Makes Its Move*. Ed. by Anandtech. Feb. 18, 2014. URL: http://www.anandtech.com/show/7764/the-nvidia-geforce-gtx-750-ti-and-gtx-750-review-maxwell/ (visited on 09/12/2015).

[14] A. Varghese et al. "Programming the Adpateva Epiphany 64-Core Network-on-Chip Coprocessor." In: *Parallel and Distributed Processing Symposium Workshops* (2014), pp. 984–992.

[15] Frank Warmerdam et al. *LibTIFF - TIFF Library and Utilities*. Ed. by LibTIFF. 2015. URL: http://www.libtiff.org/ (visited on 09/19/2015).