NTNU – Trondheim
Norwegian University of
Science and Technology

Analyzing the Performance of the Epiphany Processor

Trygve Aaberge

Master of Science in Computer Science
Submission date: August 2014
Supervisor: Anne Cathrine Elster, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Assignment

The Parallella computer by Adapteva is a new, small and energy efficient computer, that equips the 16 core co-processor Epiphany, also designed by Adapteva.

This project will focus on analyzing the performance of the Epiphany co-processor. Several test cases will be developed. Tests on how well they perform on the Epiphany in comparison to other processors, will be included.

Abstract

Our need for computational power steadily increases as we strive to solve more complex problems. As the increase of computational power of central processing units (CPUs) is slowing down, due to increasingly smaller and more complex designs, it may be a good idea to look into different processor designs.

At the same time, the need for low-power processors are increasing. There is many reasons for this. The desire to reduce costs and produce more environmentally friendly data centers is one. Another is to reduce the heat dissipation, which is more important the smaller the processor design is. Lastly, mobile devices are becoming increasingly popular.

In this project, I will look into the performance of the Epiphany processor using the Parallella device. This is a new design of a small and low-power processor. The processor I am testing contains 16 processor cores. There is also produced a version with with 64 cores, and future versions will have even more.

I will implement algorithms for suited problems and measure the run times of these on the Epiphany. To have a reference point for the performance of the Epiphany, similar implementations are run and measured on the main processor of the Parallella, a Xilinx Zynq7020.

Sammendrag

Behovet for prosessorkraft er stadig økende mens vi streber etter å løse problemer av større kompleksitet. Nå som økningen i prosessorkraft begynner å avta, på grunn av mindre og mer komplekse prosessordesign, kan det være en god idé å undersøke forskjellige prosessordesign.

På samme tid øker behovet for energieffektive prosessorer. Det er mange grunner til dette. Ønsket om å redusere kostnader og produsere mer miljøvennlige datasentere er en grunn. En annen er å redusere varmetapet, som er viktigere jo mindre prosessoren er. Til sist blir mobile enheter stadig mer populært.

I dette prosjektet vil jeg undersøke ytelsen til Epiphany-prosessoren ved å bruke Parallella-enheten. Dette er et nytt design for en liten og energieffektiv prosessor. Prosessoren jeg tester inneholder 16 prosessorkjerner. Det er også produsert en utgave med 64 kjerner, og fremtidige utgaver vil ha enda flere.

Jeg kommer til å implementere algoritmer for egnede problemer og måle kjøretiden av disse på Epiphany-prosessoren. For å ha et referansepunkt for ytelsen til Epiphany-prosessoren, vil liknende implementeringer kjøres og måles på hovedprosessoren til Parallella-enheten, en Xilinx Zynq7020.

Acknowledgments

I would like to express my appreciation and gratitude to Dr. Anne C. Elster for guidance and assistance throughout the project.

I would also like to thank Dr. Malik M. Zaki Khan and Rune E. Jensen for their help in this project.

Lastly, I would like to thank NTNU for their support of the HPC-lab directed by Dr. Anne C. Elster.

Contents

Abbreviations

COPRTHR CO-PRocessing THReads

MIMD multiple instruction, multiple data

OpenCL Open Computing Language

SDK software development kit

SIMD single instruction, multiple data

eSDK Epiphany Software Development Kit

Chapter 1

Introduction

1.1 Motivation

There are always interest in creating new methods of computing, and improving on existing ones. However, it may not always be possible to improve the design of a specific technology further.

This is becoming the case of traditional processor design now. Because of heat dissipation, it is not possible to increase the clock frequency of the processors used today.

There is also a change in usage. Mobile devices are becoming increasingly used, and requires more and more performance. However, improvements in the battery technology does not increase as fast as the performance.

For these reasons, we need to think in new ways.

1.2 The Parallella Device

The Parallella is a very small computer[8]. It is capable of performing all the tasks of any kind of computer, but it is especially suited for some tasks. The size of the device is about the size of a credit card in width and length, and about one centimeter high. It features a Dual-Core ARM central processing unit (CPU) and 1 GB of RAM, but its main trait is the co-processor it has. This co-processor is a special processor called Epiphany, which has many cores over a very small amount of physical space. It is created by Adapteva, the same company that created the Parallella. The Parallella is the first consumer device to equip this processor.

The Epiphany is a processor that is created for parallel tasks[5][3]. In contrast to standard processors that we find in todays computers, the Epiphany has many more cores. In this first version of the processor, it has 16 cores.

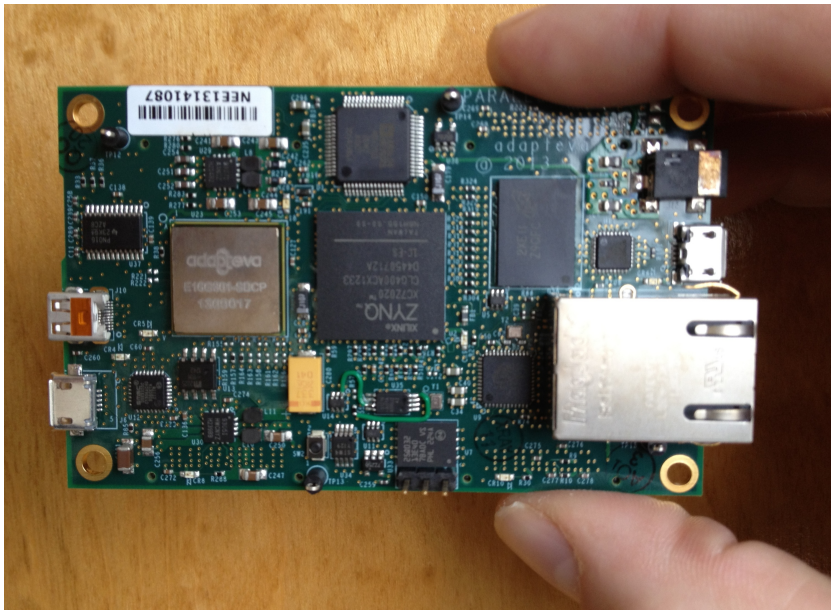

Figure 1.1: The Parallella Board[9].

They have recently finished another version, with 64 cores[10]. In the future, they will create future versions with even more cores, or combine multiple 64-core Epiphany processors in an array. This way, the platform should be suited for massively parallel tasks. Adapteva claims that the Epiphany architecture should support up to 4096 cores, all sharing a single memory space.

While the Epiphany is very parallel, it has some drawbacks compared to a traditional processor, which is the main reason that the Parallella uses it as its co-processor and has another processor as its main processor. The Epiphany uses a unique instruction set created specifically for this processor, which means that programs will have to be compiled specifically for this processor. However, it may still be programmed with ANSI C and it supports pthreads and Open Computing Language (OpenCL) as well.

The memory architecture of the Epiphany is unusual as it has no local caches or memory hierarchy. Instead, each core has a small amount of memory, and it is up to the programmer how it should be used. This is one of the ways the processor is able to employ so many cores on such a small space.

The cores of the processor uses a multiple instruction, multiple data (MIMD) architecture. It is therefore possible to assign each core with its own task, which does not have to be related to the tasks of the other cores. This is in contrast to a graphics processing unit (GPU), which uses a single instruction, multiple data (SIMD) architecture, where each core must work

on the same task simultaneously with different sets of data.

One of the other remarkable features of the Epiphany, apart from the many cores relative to its size, is the power consumption of the processor. Both the version of the processor with 16 cores and the one with 64 cores has a maximum power consumption of 2 Watts.

1.3 Problems suited for the Epiphany

There are some factors that make some problems well suited for the Epiphany, and other problems less suited. Of course, the tasks should be parallel, as the main feature of the Epiphany is its parallel nature. However, since the architecture is MIMD, it does not have to be a task that is only parallel in its data, though it very well may be. Another factor is the memory of the Epiphany. Since the processor has a so little amount of memory it should not be a task that requires very high amounts of data. Last, the data transfer between the main processor of the Parallella and the Epiphany is not very fast. This means that the Epiphany is best suited for generating data, or processing a small amount of data many times. We would want to transfer a small amount of data to the Epiphany, then work on this data in many iterations, and at last transfer the result back to the main processor.

One of the simplest parallel tasks are vector and matrix multiplication. For each pair of fields of the source vectors or matrices, there is done a calculation. Each of these calculations can be spread over equally many processors.

An interesting problem is the calculation of the Mandelbrot set. This is a complex calculation which takes some initial values and does iterations of a particular mathematical operation. Hence, we can do many calculations on the Epiphany, without transferring much data to it. The calculations operates independently on a grid of values and is therefore very parallelizable.

I also want to look at Conway's Game of Life[11]. This consists of a grid of values that are considered to be either dead or alive. It takes an initial input, and calculates a new set of values from this. The algorithm can run for an arbitrary number of frames, and each new frame is calculated from the past. Each value is calculated only by using its neighbours values, and it is therefore very parallelizable.

1.4 Methods of using the Epiphany

The Epiphany supports ANSI C with support for pthreads and OpenCL. There are two SDKs you may use to program for the Epiphany: The Epiphany Software Development Kit (eSDK) or the CO-PRocessing THReads (COPRTHR) software development kit (SDK).

The eSDK[6] is an SDK which enables you to compile C code for the Epiphany. The implementation provides support for assigning different tasks to different cores. It also implements pthreads.

The COPRTHR[2] SDK is another SDK which provides an OpenCL implementation. By using this, you may easily port OpenCL applications to the Epiphany.

1.5 Outline of Chapters

The rest of this report will be organized as follows:

- Chapter 2: Different programs for measuring performance and the results of running these on the Epiphany compared to the main processor of the Parallella.
- Chapter 3: Discussion of the results gathered in the previous chapter.
- Chapter 4: Short summary of the work done in this project and the results of this. Also includes thoughts of what related future projects may look into.
- Appendix: Includes the code written and used in this project.

Chapter 2

Results

In this chapter I will present what kind of implementations I made, and the results I got from running these. For comparison of run times relative to the Epiphany, I have also run similar implementations on the main processor of the Parallella, a Xilinx Zynq7020[8]. Since the implementations for the Epiphany uses special libraries to utilize its cores, I could not directly run the same code on the main processor. Therefore I implemented single threaded programs in standard C using the same algorithms.

This Chapter is organized as follows:

- Section 2.1: Results gathered using the COPRTHR SDK. This contains implementations of vector multiplication and computation of he Mandelbrot set.
- Section 2.2: Results gathered using the eSDK. This contains an implementation of Conway's Game of Life.

2.1 COPRTHR

Since I had some previous experience with OpenCL, the COPRTHR SDK seemed like the easiest of the two libraries to use.

One very simple parallel task is to multiply two vectors. We transfer each of the vectors onto the memory of the co-processor. Then, each processor handles its own part of the vector and multiplies each element in its part of one of the vectors with the corresponding element in the other vector. The result is stored in a third vector.

Unfortunately, the performance of each of the Epiphany cores seems to be 50 to 100 times slower than the core of the main processor of the Parallella.

If we combine all of the cores, it then gives 3 to 6 times worse performance than one of the cores of the main processor.

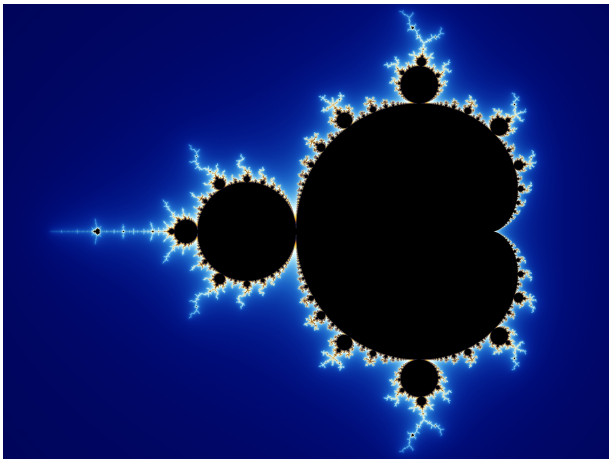

Figure 2.1: A visualization of the Mandelbrot set[1].

Another problem that may be suited for the Epiphany is calculating the Mandelbrot set. With this problem, we only have to transfer some initial parameters. We may then do many calculations based on these, and at last transfer the result back. For each of the values transferred back, we would have done many iterations on the Epiphany. Unfortunately, so far, the results I have found is very poor. The calculation on the Epiphany seems to be over a 1000 times slower than the calculation on the main processor.

2.2 eSDK

Because of the poor results I experienced with the COPRTHR SDK, I decided to test the eSDK. Initially, I had some issues with running code using this SDK. However, I was directed at an example of a implementation of computing the Mandelbrot set using it[7]. This code showed how to compile and run the code.

Since computation of the Mandelbrot set already was implemented, I decided that I would implement Conway's Game of Life. I made an implementation for the Epiphany processor, using the Mandelbrot example as help on how to implement it. For benchmarking, an implementation for another processor was also needed. I decided to implement a single threaded version using standard C. For comparison with the Epiphany, I ran this on the main processor of the Parallella.

Figure 2.2: A figure of a position in Conway's Game of Life[12].

In the single threaded implementation in C, I could use arrays to store values of each of the fields of the board. Since the main goal is to compare performance with the Epiphany, I also created an implementation that allocated a memory space and used that manually. This is in a similar fashion to how the implementation for the Epiphany is.

For the benchmarking, I used 180 rows, 180 columns and the same starting pattern for each run. The results are presented here. The first columns is the number of frames, or steps, the programs was running for. The rest of the columns is the results for respectively the parallel Epiphany implementation, the serial implementation on the main processor using arrays and the serial implementation on the main processor using an allocated memory space. The results are running time in seconds.

Frames	Epiphany	Xilinx, array	Xilinx, malloc
100	0.223	3.222	3.917
2000	4.429	59.506	79.031
10000	22.106	291.853	355.537

Chapter 3

Discussion

In this chapter, I will discuss the results presented in the previous chapter.

This Chapter is organized as follows:

- Section 3.1: Discussion of the results gathered using the COPRTHR SDK. Theories of why the performance using this SDK was poor.
- Section 3.2: Discussion of the results gathered using the eSDK.
- Section 3.3: Notes about potential issues of running code on the Epiphany.

3.1 COPRTHR

The results by using the COPRTHR SDK was fairly disappointing. If we can manage better performance from a standard ARM CPU that uses about the same amount of power, the selling point for the Epiphany is lost.

However, since the results using the eSDK is far better, we can see that the reason for the poor results here are not the CPU itself. The reason for the slow performance is unknown.

One reason for this may be that the COPRTHR SDK simply does not perform as well as the eSDK. Another reason may be that I could have used the COPRTHR SDK in a way that is not the most optimum for performance.

3.2 eSDK

Using the eSDK instead of the COPRTHR SDK yielded far better results. We can see that the Epiphany managed about 13 times better performance than a single thread of the Xilinx CPU when using arrays in the single threaded

implementation. When allocating a memory space manually for the data in the single threaded implementation, the Epiphany ran about 17 times faster.

The reason for better performance using arrays than an allocated memory space is probably because of some optimizations by the C compiler. The implementation using a memory space is the one closest to the implementation for the Epiphany, so if we want to compare raw computational power, this is the relevant comparison.

However, in a normal C implementation we would probably use arrays and then get a bit better performance. This means that if we want to compare the performance gain of this algorithm, we would want to look at this comparison.

The Xilinx CPU on the Parallella is a fairly powerful processor in this category[13]. To see that we can gain many times better performance using the Epiphany, is very good.

Of course, the Xilinx CPU is a general processor, while the Epiphany is probably best suited for specific tasks. This means that the Epiphany will not replace the standard CPUs. It could mean that the Epiphany will be very interesting to use for the tasks that it is suited for.

The 64-core version of the Epiphany will have the same maximum power consumption as the version with 16-core used in this thesis[3][4]. In addition to having four times as many cores, they will also run on a slightly higher clock frequency. This means that this processor will have much higher performance, with about the same power consumption.

3.3 Programming issues

For the last part of the discussion, I would like to point out some issues I ran into, and some pointers to what is important to check.

It is very important that your environment is set up correctly. If you don't have the correct environment variables set, you may not be able to run programs on the Epiphany. The error messages received because of this is not always obvious. All of the necessary environment variables are set up correctly in the run file provided in A.5.

If you try to read or write from an invalid memory segment on the Epiphany, for example if you write to an address that is not aligned to a word, the Epiphany may hang, without any feedback.

Chapter 4

Conclusion and Future work

In this project we have made implementations to test the performance of the Epiphany using the two SDKs available. The algorithms implementation and used for testing was vector multiplication, computation of the Mandelbrot set and Conway's Game of Life. We saw that the performance of the Epiphany using the COPRTHR SDK was far worse than the main processor of the Parallella.

However, using the eSDK we saw that the performance of the Epiphany was 13–17 times better than the main processor. This result is considered very good. We also noted that there will come a version of the Epiphany with 64 cores which will have many times the performance of the version with 16 core, while still consuming about the same amount of power.

4.1 Future work

- As stated, there will come 64-core version of the Epiphany. When this is available, it would be interesting to test the performance of the processor compared to the version with 16 cores tested here.

- The power of this processor lies in its parallel nature. For problems that are far more parallel than for 16 or 64 cores, it would be interesting to see how the Epiphany would perform in a cluster. Either by networking many Parallella devices together, or even better, to combine many Epiphany processors on a single board. The goal could be to measure how this will perform in comparison with a super computer.

- The Epiphany is a processor with a very low power consumption. Maximum and typical power consumption are provided by Adapteva, but

this is very rough numbers. Another work could check the power consumption in detail for different computations, and compare this to other processors.

- Unlike GPUs, which has very many cores, each core of the Epiphany can do an individual computation simultaneously as all of the other cores. This means that the Epiphany could be well suited for task-parallel problems, as well as data-parallel problems. In this project, only data-parallel problems were explored. Another work could measure the performance of task-parallel problems using the Epiphany.

Bibliography

[1] Mandel zoom 00 mandelbrot set. https://commons.wikimedia.org/wiki/File:Mandel_zoom_00_mandelbrot_set.jpg#mediaviewer/File:Mandel_zoom_00_mandelbrot_set.jpg, August 2014. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons.

[2] Adapteva. Coprthr® api reference. http://www.browndeertechnology.com/docs/coprthr_api_ref.pdf, August 2014.

[3] Adapteva. E16g301 epiphany(tm) 16-core microprocessor datasheet. www.adapteva.com/docs/e16g301_datasheet.pdf, August 2014.

[4] Adapteva. E64g401 epiphany(tm) 64-core microprocessor datasheet. http://www.adapteva.com/docs/e64g401_datasheet.pdf, August 2014.

[5] Adapteva. Epiphany architecture reference. http://adapteva.com/docs/epiphany_arch_ref.pdf, August 2014.

[6] Adapteva. Epiphany sdk reference. http://adapteva.com/docs/epiphany_sdk_ref.pdf, August 2014.

[7] Adapteva. parallella-examples/mandelbrot - github. https://github.com/parallella/parallella-examples/tree/master/mandelbrot, August 2014.

[8] Adapteva. Parallella reference manual. http://www.parallella.org/docs/parallella_manual.pdf, August 2014.

[9] Adapteva Andreas Olofsson. The parallella board has arrived! http://www.parallella.org/2013/04/16/hello-world-my-name-is-parallella/, August 2014.

[10] Adapteva Andreas Olofsson. Update #53: The 64-core parallella is alive! http://www.parallella.org/2014/04/25/the-64-core-parallella-is-alive/, August 2014.

[11] Martin Gardner. Mathematical games - the fantastic combinations of john conway's new solitaire game "life". http://web.archive.org/web/20090603015231/http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm, August 2014.

[12] Kieff. Gospers glider gun. https://commons.wikimedia.org/wiki/File:Gospers_glider_gun.gif#mediaviewer/File:Gospers_glider_gun.gif, August 2014. Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons.

[13] Xilinx. Unmatched performance and power. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/performance.html, August 2014.

Appendix A

Code implementations

All of the code is also attached in a separate ZIP file.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/time.h>
5
6  #include "coprthr.h"
7
8  #define SIZE 1048576
9  #define SIZE16 "1048576 / 16"
10
11 char src[] = \
12         "#include <coprthr.h>\n" \
13         "void my_kern( float* a, float* b, float* c) {\n" \
14         "       int idx = coprthr_get_thread_index() * " SIZE16
               ";\n" \
15         "       int to = idx+" SIZE16 ";\n" \
16         "       for(int i=idx; i<to; i++) {\n" \
17         "               c[i] = a[i]*b[i];\n" \
18         "       }\n" \
19         "}\n";
20
21 double walltime()
22 {
23         static struct timeval t;
24         gettimeofday(&t, NULL);
25         return (t.tv_sec + 1e-6 * t.tv_usec);
26 }
27
28 int main()
29 {
30         int i;
31
```

```
32          int dd = coprthr_dopen(COPRTHR_DEVICE_E32,
                COPRTHR_O_STREAM);
33
34          if (dd<0) {
35                  printf("device open failed\n");
36                  exit(-1);
37          }
38
39
40          coprthr_program_t prg = coprthr_dcompile(dd, src, sizeof(
                src), "", 0);
41          coprthr_kernel_t krn = coprthr_getsym(prg, "my_kern");
42
43          float* a = (float*) malloc(SIZE*sizeof(float));
44          float* b = (float*) malloc(SIZE*sizeof(float));
45          float* c = (float*) malloc(SIZE*sizeof(float));
46
47          for(i=0; i<SIZE; i++) {
48                  a[i] = 1.0f * i;
49                  b[i] = 2.0f * i;
50                  c[i] = 0.0f;
51          }
52
53          coprthr_mem_t mema = coprthr_dmalloc(dd, SIZE*sizeof(
                float), 0);
54          coprthr_mem_t memb = coprthr_dmalloc(dd, SIZE*sizeof(
                float), 0);
55          coprthr_mem_t memc = coprthr_dmalloc(dd, SIZE*sizeof(
                float), 0);
56
57          coprthr_dwrite(dd, mema, 0, a, SIZE*sizeof(float),
                COPRTHR_E_NOWAIT);
58          coprthr_dwrite(dd, memb, 0, b, SIZE*sizeof(float),
                COPRTHR_E_NOWAIT);
59          coprthr_dwrite(dd, memc, 0, c, SIZE*sizeof(float),
                COPRTHR_E_NOWAIT);
60
61          unsigned int nargs = 3;
62          void* args[] = { &mema, &memb, &memc };
63          unsigned int nthr = 16;
64
65          double runtime;
66          double start = walltime();
67
68          coprthr_dexec(dd, krn, nargs, args, nthr, 0,
                COPRTHR_E_NOWAIT);
69
70          coprthr_dcopy(dd, memc, 0, memb, 0, SIZE*sizeof(float),
                COPRTHR_E_NOWAIT);
```

```
71
72          coprthr_dread(dd, memc, 0, c, SIZE*sizeof(float),
                COPRTHR_E_NOWAIT);
73
74          coprthr_dwait(dd);
75
76          runtime = walltime() - start;
77          printf("Runtime: %8.3f ms\n", runtime*1e3);
78
79          //for(i=0; i<SIZE; i++)
80          //      printf("%f * %f = %f\n", a[i], b[i], c[i]);
81
82          coprthr_dfree(dd, mema);
83          coprthr_dfree(dd, memb);
84          coprthr_dfree(dd, memc);
85
86          free(a);
87          free(b);
88          free(c);
89
90          coprthr_dclose(dd);
91  }
```

Listing A.1: Implementation of vector multiplication using COPRTHR

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/time.h>
5
6  #define SIZE 1048576
7
8  double walltime()
9  {
10         static struct timeval t;
11         gettimeofday(&t, NULL);
12         return (t.tv_sec + 1e-6 * t.tv_usec);
13 }
14
15 int main()
16 {
17         int i;
18
19         float* a = (float*) malloc(SIZE*sizeof(float));
20         float* b = (float*) malloc(SIZE*sizeof(float));
21         float* c = (float*) malloc(SIZE*sizeof(float));
22
23         for(i=0; i<SIZE; i++) {
24                 a[i] = 1.0f * i;
```

```
25                      b[i] = 2.0f * i;
26                      c[i] = 0.0f;
27              }
28
29              double runtime;
30              double start = walltime();
31
32              for (i = 0; i < SIZE; i++) {
33                      c[i] = a[i] * b[i];
34              }
35
36              runtime = walltime() - start;
37              printf("Runtime: %8.3f ms\n", runtime*1e3);
38
39              //for(i=0; i<SIZE; i++)
40              //      printf("%f + %f = %f\n", a[i], b[i], c[i]);
41
42              free(a);
43              free(b);
44              free(c);
45      }
```

Listing A.2: Serial implementation of vector multiplication

```
1  /*
2   *
3   *  Originally created for CUDA by Ruben Spaans for Anne C Elster
        and her parallel programming class.
4   *  Modified to COPRTHR for the Parallella device by Trygve
        Aaberge.
5   *
6   */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10 #include <sys/time.h>
11 #include <string.h>
12 #include "coprthr.h"
13 #include "coprthr_cc.h"
14
15 /* Problem size */
16 #define XSIZE 16
17 #define YSIZE 16
18
19 #define MAXITER 255
20
21 double xleft=-2.01;
22 double xright=1;
23 double yupper,ylower;
```

```
24 double ycenter=1e-6;
25 double step;
26
27 int host_pixel[XSIZE*YSIZE];
28 int device_pixel[XSIZE*YSIZE];
29
30 typedef struct {
31         double real,imag;
32 } my_complex_t;
33
34 #define PIXEL(i,j) ((i)+(j)*XSIZE)
35
36 /********** SUBTASK1: Create kernel device_calculate
       ************************/
37 char src[] = \
38         "#include <coprthr.h>\n" \
39         "#define XSIZE 16\n" \
40
41         "typedef struct {\n" \
42         "       double real,imag;\n" \
43         "} my_complex_t;\n" \
44
45         "void my_kern(int* a, double* xleft, double* yupper,
              double* step) {\n" \
46         "       int j = coprthr_get_thread_index();\n" \
47         "       for (int i = 0; i < XSIZE; i++) {\n" \
48         "               my_complex_t c,z,temp;\n" \
49         "               int iter=0;\n" \
50         "               c.real = (*xleft + *step*i);\n" \
51         "               c.imag = (*yupper - *step*j);\n" \
52         "               z = c;\n" \
53         "               while(z.real*z.real + z.imag*z.imag <
              4.0) {\n" \
54         "                       temp.real = z.real*z.real - z.
              imag*z.imag + c.real;\n" \
55         "                       temp.imag = 2.0*z.real*z.imag + c
              .imag;\n" \
56         "                       z = temp;\n" \
57         "                       if(++iter==255) break;\n" \
58         "               }\n" \
59         "               a[i+j*XSIZE]=iter;\n" \
60         "       }\n" \
61         "}\n";
62
63 /********** SUBTASK1 END
       ************************************************/
64
65 void host_calculate()
66 {
```

```
67          for(int j=0;j<YSIZE;j++) {
68                  for(int i=0;i<XSIZE;i++) {
69                          /* Calculate the number of iterations
                               until divergence for each pixel.
70                           If divergence never happens, return
                               MAXITER */
71                          my_complex_t c,z,temp;
72                          int iter=0;
73                          c.real = (xleft + step*i);
74                          c.imag = (yupper - step*j);
75                          z = c;
76                          while(z.real*z.real + z.imag*z.imag <
                                4.0) {
77                                  temp.real = z.real*z.real - z.
                                        imag*z.imag + c.real;
78                                  temp.imag = 2.0*z.real*z.imag + c
                                        .imag;
79                                  z = temp;
80                                  if(++iter==MAXITER) break;
81                          }
82                          host_pixel[PIXEL(i,j)]=iter;
83                  }
84          }
85  }
86
87  typedef unsigned char uchar;
88
89  /* save 24-bits bmp file, buffer must be in bmp format: upside-
        down */
90  void savebmp(char *name,uchar *buffer,int x,int y)
91  {
92          FILE *f=fopen(name,"wb");
93          if(!f) {
94                  printf("Error writing image to disk.\n");
95                  return;
96          }
97          unsigned int size=x*y*3+54;
98          uchar header[54]={'B','M',size&255,(size>>8)&255,(size
                >>16)&255,size>>24,0,
99                  0,0,0,54,0,0,0,40,0,0,0,x&255,x>>8,0,0,y&255,y
                        >>8,0,0,1,0,24,0,0,0,0,0,0,
100                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
101         fwrite(header,1,54,f);
102         fwrite(buffer,1,x*y*3,f);
103         fclose(f);
104 }
105
106 /* given iteration number, set a colour */
107 void fancycolour(uchar *p,int iter)
```

```
108 {
109         if(iter==MAXITER);
110         else if(iter<8) { p[0]=128+iter*16; p[1]=p[2]=0; }
111         else if(iter<24) { p[0]=255; p[1]=p[2]=(iter-8)*16; }
112         else if(iter<160) { p[0]=p[1]=255-(iter-24)*2; p[2]=255;
                }
113         else { p[0]=p[1]=(iter-160)*2; p[2]=255-(iter-160)*2; }
114 }
115
116 /*
117  * Get system time to microsecond precision (ostensibly, the same
        as MPI_Wtime),
118  * returns time in seconds
119  */
120 double walltime ()
121 {
122         static struct timeval t;
123         gettimeofday(&t, NULL);
124         return (t.tv_sec + 1e-6 * t.tv_usec);
125 }
126
127 int main(int argc,char **argv)
128 {
129         if(argc==1) {
130                 puts("Usage: MANDEL n");
131                 puts("n decides whether image should be written
                        to disk (1=yes, 0=no)");
132                 return 0;
133         }
134         double start;
135         double hosttime=0;
136         double devicetime=0;
137         double memtime=0;
138
139         int dd = coprthr_dopen(COPRTHR_DEVICE_E32,
                COPRTHR_O_STREAM);
140
141         if (dd<0) {
142                 printf("device open failed\n");
143                 exit(-1);
144         }
145
146         coprthr_program_t prg = coprthr_dcompile(dd,src,sizeof(
                src),"",0);
147         coprthr_kernel_t krn = coprthr_getsym(prg,"my_kern");
148
149         /* Calculate the range in the y-axis such that we
                preserve the
150          aspect ratio */
```

```
151         step=(xright-xleft)/XSIZE;
152         yupper=ycenter+(step*YSIZE)/2;
153         ylower=ycenter-(step*YSIZE)/2;
154
155         /* Host calculates image */
156         start=walltime();
157         host_calculate();
158         hosttime+=walltime()-start;
159
160         /********** SUBTASK2: Set up device memory
                ******************************/
161
162         coprthr_mem_t mema = coprthr_dmalloc(dd, XSIZE*YSIZE*
                sizeof(int),0);
163         coprthr_mem_t memxleft = coprthr_dmalloc(dd, sizeof(
                double),0);
164         coprthr_mem_t memyupper = coprthr_dmalloc(dd, sizeof(
                double),0);
165         coprthr_mem_t memstep = coprthr_dmalloc(dd, sizeof(double
                ),0);
166
167         memset(device_pixel, 0, XSIZE*YSIZE*sizeof(int));
168         coprthr_dwrite(dd,mema,0,device_pixel,XSIZE*YSIZE*sizeof(
                int),COPRTHR_E_NOWAIT);
169         coprthr_dwrite(dd,memxleft,0,&xleft,sizeof(double),
                COPRTHR_E_NOWAIT);
170         coprthr_dwrite(dd,memyupper,0,&yupper,sizeof(double),
                COPRTHR_E_NOWAIT);
171         coprthr_dwrite(dd,memstep,0,&step,sizeof(double),
                COPRTHR_E_NOWAIT);
172
173         /********** SUBTASK2 END
                **********************************************/
174
175         start=walltime();
176         /********** SUBTASK3: Execute the kernel on the device
                ******************/
177
178         unsigned int nargs = 4;
179         void* args[] = { &mema, &memxleft, &memyupper, &memstep
                };
180         unsigned int nthr = YSIZE;
181
182         coprthr_dexec(dd,krn,nargs,args,nthr,0,COPRTHR_E_NOWAIT);
183
184         coprthr_dwait(dd);
185
186         /********** SUBTASK3 END
                **********************************************/
```

```
187         devicetime+=walltime()-start;
188
189         start=walltime();
190         /********** SUBTASK4: Transfer the result from device to
                device_pixel[][][]*/
191
192         coprthr_dread(dd,mema,0,device_pixel,XSIZE*YSIZE*sizeof(
                int),COPRTHR_E_NOWAIT);
193
194         coprthr_dwait(dd);
195
196         /********** SUBTASK4 END
                ***********************************************/
197         memtime+=walltime()-start;
198
199         /********** SUBTASK5: Free the device memory also
                ***********************/
200
201         coprthr_dfree(dd,mema);
202
203         /********** SUBTASK5 END
                ***********************************************/
204
205         int errors=0;
206         /* check if result is correct */
207         for(int i=0;i<XSIZE;i++) {
208                 for(int j=0;j<YSIZE;j++) {
209                         int diff=host_pixel[PIXEL(i,j)]-
                                device_pixel[PIXEL(i,j)];
210                         if(diff<0) diff=-diff;
211                         /* allow +-1 difference */
212                         if(diff>1) {
213                                 if(errors<10)
214                                         printf("Error on pixel %d
                                                %d: expected %d,
                                                found %d\n",
215                                                 i,j,host_pixel[
                                                    PIXEL(i,j)],
                                                    device_pixel[
                                                    PIXEL(i,j)]);
216                                 else if(errors==10)
217                                         puts("...");
218                                 errors++;
219                         }
220                 }
221         }
222         if(errors>0) printf("Found %d errors.\n",errors);
223         else puts("Device calculations are correct.");
224
```

```
225         printf("\n");
226         printf("Host time:           %7.3f ms\n",hosttime*1e3);
227         printf("Device calculation: %7.3f ms\n",devicetime*1e3);
228         printf("Copy result:         %7.3f ms\n",memtime*1e3);
229
230         if(strtol(argv[1],NULL,10)!=0) {
231                 /* create nice image from iteration counts. take
                       care to create it upside
232                  down (bmp format) */
233                 unsigned char *buffer=(unsigned char *)calloc(
                       XSIZE*YSIZE*3,1);
234                 for(int i=0;i<XSIZE;i++) {
235                         for(int j=0;j<YSIZE;j++) {
236                                 int p=((YSIZE-j-1)*XSIZE+i)*3;
237                                 fancycolour(buffer+p,device_pixel
                                       [PIXEL(i,j)]);
238                         }
239                 }
240                 /* write image to disk */
241                 savebmp("mandel1.bmp",buffer,XSIZE,YSIZE);
242         }
243         return 0;
244 }
```

Listing A.3: Serial implementation and implementation using COPRTHR of mandelbrot

```
1  ESDK=$(EPIPHANY_HOME)
2  ELIBS=$(ESDK)/tools/host/lib
3  EINCS=$(ESDK)/tools/host/include
4  ELDF=$(ESDK)/bsps/current/internal.ldf
5  EXES=main epiphany.srec
6  OBJS=epiphany.elf
7
8  all: $(EXES)
9
10
11 main: host.c shared_data.h
12         gcc -O3 host.c -o main -I $(EINCS) -L $(ELIBS) -le-hal -
               lrt
13
14 epiphany.elf: epiphany.c shared_data.h
15         e-gcc -O3 -funroll-loops -ffast-math -T $(ELDF) epiphany.
               c -o epiphany.elf -le-lib
16
17 epiphany.srec: epiphany.elf
18         e-objcopy --srec-forceS3 --output-target srec epiphany.
               elf epiphany.srec
19
```

```
20 clean:
21         rm $(EXES) $(OBJS)
```

Listing A.4: Makefile for implementations using eSDK

```
1  #!/bin/bash
2
3  set -e
4
5  ESDK=${EPIPHANY_HOME}
6  ELIBS=${ESDK}/tools/host/lib:${LD_LIBRARY_PATH}
7  EHDF=${EPIPHANY_HDF}
8
9  setterm -blank 0 -cursor off
10 sudo -E LD_LIBRARY_PATH=${ELIBS} EPIPHANY_HDF=${EHDF} ./main
```

Listing A.5: Run-file for implementations using eSDK

```
1  /*
2  Copyright (c) 2013-2014, Shodruky Rhyammer
3  Copyright (c) 2014, Trygve Aaberge
4  All rights reserved.
5
6  Redistribution and use in source and binary forms, with or
       without modification,
7  are permitted provided that the following conditions are met:
8
9          Redistributions of source code must retain the above
               copyright notice, this
10         list of conditions and the following disclaimer.
11
12         Redistributions in binary form must reproduce the above
               copyright notice, this
13         list of conditions and the following disclaimer in the
               documentation and/or
14         other materials provided with the distribution.
15
16         Neither the name of the copyright holders nor the names
               of its
17         contributors may be used to endorse or promote products
               derived from
18         this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS "AS IS" AND
21 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
        THE IMPLIED
22 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
       PURPOSE ARE
```

```
23 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
       CONTRIBUTORS BE LIABLE FOR
24 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
       CONSEQUENTIAL DAMAGES
25 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
       OR SERVICES;
26 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
       CAUSED AND ON
27 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
       OR TORT
28 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
        USE OF THIS
29 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30
31 Originally code for Mandelbrot computation, licenced as stated
       above, from:
32 https://github.com/parallella/parallella-examples/tree/master/
       mandelbrot
33 Used as basis and modified for this Game of Life implementation.
34 */
35
36
37 #include <stdio.h>
38 #include <string.h>
39 #include <stdlib.h>
40 #include <unistd.h>
41 #include <fcntl.h>
42 #include <sys/ioctl.h>
43 #include <sys/mman.h>
44 #include <stdint.h>
45 #include <time.h>
46 #include <e-hal.h>
47 #include "shared_data.h"
48
49 #define BUF_OFFSET 0x01000000
50
51 static inline void nano_wait(uint32_t sec, uint32_t nsec)
52 {
53         struct timespec ts;
54         ts.tv_sec = sec;
55         ts.tv_nsec = nsec;
56         nanosleep(&ts, NULL);
57 }
58
59 int main(int argc, char *argv[])
60 {
61         e_platform_t eplat;
62         e_epiphany_t edev;
63         e_mem_t emem;
```

```
64          static msg_block_t msg;
65          memset(&msg, 0, sizeof(msg));
66          struct timespec time;
67          double time0, time1;
68
69          e_init(NULL);
70          e_reset_system();
71          e_get_platform_info(&eplat);
72          e_alloc(&emem, BUF_OFFSET, sizeof(msg_block_t));
73
74          unsigned int xres = XRES;
75          unsigned int yres = YRES;
76          msg.info.xres = xres;
77          msg.info.yres = yres;
78
79          unsigned int core = 0;
80          unsigned int row = 0;
81          unsigned int col = 0;
82          volatile unsigned int vepiphany[CORES];
83          volatile unsigned int vcoreid = 0;
84          unsigned int vhost[CORES];
85          for (core = 0; core < CORES; core++) {
86                  vepiphany[core] = 0;
87                  vhost[core] = 0;
88          }
89
90          FILE *file = fopen("grid.txt", "r");
91          if (file == NULL) {
92                  fprintf(stderr, "Could not open grid.txt\n");
93                  exit(1);
94          }
95
96          int x, y, c;
97          int eol = 0;
98          for (row = 0; row < ROWS; row++) {
99                  for (y = 0; y < yres; y++) {
100                         for (col = 0; col < COLS; col++) {
101                                 core = row * COLS + col;
102                                 for (x = 0; x < xres; x++) {
103                                         if (!eol) {
104                                                 c = fgetc(file);
105                                                 if (c == '\n') {
106                                                         eol = 1;
107                                                 }
108                                         }
109                                         if (c == '1') {
110                                                 msg.pixels[core][
                                                        y][x] = 1;
111                                         } else {
```

```
112                                                     msg.pixels[core][
                                                            y][x] = 0;
113                                             }
114                                     }
115                             }
116                             eol = 0;
117                     }
118             }
119
120             e_open(&edev, 0, 0, ROWS, COLS);
121             e_write(&emem, 0, 0, 0, &msg, sizeof(msg));
122             e_reset_group(&edev);
123             e_load_group("epiphany.srec", &edev, 0, 0, ROWS, COLS,
                    E_TRUE);
124             nano_wait(0, 100000000);
125             clock_gettime(CLOCK_REALTIME, &time);
126             time0 = time.tv_sec + time.tv_nsec * 1.0e-9;
127
128             unsigned int frame = 0;
129             while (1) {
130                     for (row = 0; row < ROWS; row++) {
131                             for (col = 0; col < COLS; col++) {
132                                     core = row * COLS + col;
133                                     while (1) {
134                                             e_read(&emem, 0, 0, (
                                                    off_t)((char *)&msg.
                                                    msg_d2h[core] - (char
                                                     *)&msg), &msg.
                                                    msg_d2h[core], sizeof
                                                    (msg_dev2host_t));
135                                             vepiphany[core] = msg.
                                                    msg_d2h[core].value;
136                                             if (vhost[core] -
                                                    vepiphany[core] >
                                                    ((~0u) >> 1)) {
137                                                     break;
138                                             }
139                                             nano_wait(0, 1000000);
140                                     }
141                                     vhost[core] = vepiphany[core];
142                                     vcoreid = msg.msg_d2h[core].
                                            coreid;
143                                     //printf("%x row:%d col:%d\n",
                                            vepiphany[core], (vcoreid >>
                                            6), (vcoreid & 0x3f));
144                             }
145                     }
146                     for (row = 0; row < ROWS; row++) {
147                             for (col = 0; col < COLS; col++) {
```

```
148                                     e_resume(&edev, row, col);
149                             }
150                     }
151                     frame++;
152                     if (frame > FRAMES) {
153                             break;
154                     }
155             }
156
157             for (core = 0; core < CORES; core++) {
158                     while (1) {
159                             e_read(&emem, 0, 0, (off_t)((char *)&msg.
                                    msg_d2h[core] - (char *)&msg), &msg.
                                    msg_d2h[core], sizeof(msg_dev2host_t)
                                    );
160                             if (msg.msg_d2h[core].finished == 1) {
161                                     break;
162                             }
163                             nano_wait(0, 1000000);
164                     }
165                     e_read(&emem, 0, 0, (off_t)((char *)&msg.pixels[
                            core] - (char *)&msg), &msg.pixels[core],
                            sizeof(uint32_t[yres][xres]));
166             }
167
168             clock_gettime(CLOCK_REALTIME, &time);
169             time1 = time.tv_sec + time.tv_nsec * 1.0e-9;
170             printf("rows: %d, cols: %d\n", ROWS * yres, COLS * xres);
171             printf("frames: %d\n", FRAMES);
172             printf("time: %f sec\n", time1 - time0);
173
174             for (row = 0; row < ROWS; row++) {
175                     for (y = 0; y < yres; y++) {
176                             for (col = 0; col < COLS; col++) {
177                                     core = row * COLS + col;
178                                     for (x = 0; x < xres; x++) {
179                                             printf("%d ", msg.pixels[
                                                    core][y][x]);
180                                     }
181                             }
182                             printf("\n");
183                     }
184             }
185
186             e_close(&edev);
187             e_free(&emem);
188             e_finalize();
189             return 0;
190     }
```

Listing A.6: Implementation of Game of Life using eSDK, host.c

```
1  /*
2  Copyright (c) 2013-2014, Shodruky Rhyammer
3  Copyright (c) 2014, Trygve Aaberge
4  All rights reserved.
5
6  Redistribution and use in source and binary forms, with or
       without modification,
7  are permitted provided that the following conditions are met:
8
9          Redistributions of source code must retain the above
               copyright notice, this
10         list of conditions and the following disclaimer.
11
12         Redistributions in binary form must reproduce the above
               copyright notice, this
13         list of conditions and the following disclaimer in the
               documentation and/or
14         other materials provided with the distribution.
15
16         Neither the name of the copyright holders nor the names
               of its
17         contributors may be used to endorse or promote products
               derived from
18         this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS "AS IS" AND
21 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
        THE IMPLIED
22 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
       PURPOSE ARE
23 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
       CONTRIBUTORS BE LIABLE FOR
24 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
       CONSEQUENTIAL DAMAGES
25 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
       OR SERVICES;
26 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
       CAUSED AND ON
27 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
       OR TORT
28 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
        USE OF THIS
29 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30
```

```
31 Originally code for Mandelbrot computation, licenced as stated
       above, from:
32 https://github.com/parallella/parallella-examples/tree/master/
       mandelbrot
33 Used as basis and modified for this Game of Life implementation.
34 */
35
36
37 #include "e_lib.h"
38 #include "shared_data.h"
39
40 #define BUF_ADDRESS 0x8f000000
41 #define PAGE_OFFSET 0x2000
42 #define PAGE_SIZE 0x2000
43
44 unsigned int row, col, core;
45 unsigned int xres, yres;
46
47 unsigned int fetch_node(char *src, int x, int y, int x_diff, int
       y_diff) {
48         int get_global = 0;
49         int col_new = col;
50         int row_new = row;
51         int x_new = x + x_diff;
52         if (x_new < 0) {
53                 col_new--;
54                 if (col_new < 0) {
55                         return 0;
56                 }
57                 x_new = xres - 1;
58                 get_global = 1;
59         } else if (x_new == xres) {
60                 col_new++;
61                 if (col_new >= COLS) {
62                         return 0;
63                 }
64                 x_new = 0;
65                 get_global = 1;
66         }
67         int y_new = y + y_diff;
68         if (y_new < 0) {
69                 row_new--;
70                 if (row_new < 0) {
71                         return 0;
72                 }
73                 y_new = yres - 1;
74                 get_global = 1;
75         } else if (y_new == yres) {
76                 row_new++;
```

```
77                  if (row_new >= ROWS) {
78                          return 0;
79                  }
80                  y_new = 0;
81                  get_global = 1;
82          }
83
84          char *src2 = src + (y_new * yres + x_new) * 4;
85          if (get_global) {
86                  src2 = e_get_global_address(row_new, col_new,
                        src2);
87          }
88          return *src2;
89  }
90
91  int main(void)
92  {
93          e_coreid_t coreid;
94          coreid = e_get_coreid();
95          e_coords_from_coreid(coreid, &row, &col);
96          core = row * COLS + col;
97          unsigned int frame = 1;
98          unsigned int page = 1;
99          volatile msg_block_t *msg = (msg_block_t *)BUF_ADDRESS;
100         xres = msg->info.xres;
101         yres = msg->info.yres;
102
103         // For simplicity, each value is stored in a word. Since
                a word is 4 bytes,
104         // and the only possible values are 0 and 1, it would be
                possible to store 32
105         // values for each word.
106
107         char *src[2] = {(char *)(PAGE_OFFSET), (char *)(
                PAGE_OFFSET + PAGE_SIZE)};
108         unsigned int *pixel_last = (unsigned int *)src[0];
109         unsigned int *pixel_cur = (unsigned int *)src[1];
110         unsigned int x, y;
111         for (y = 0; y < yres; y++) {
112                 for (x = 0; x < xres; x++) {
113                         *pixel_last = msg->pixels[core][y][x];
114                         *pixel_cur = 0;
115                         pixel_last++;
116                         pixel_cur++;
117                 }
118         }
119
120         while (1) {
121                 msg->msg_d2h[core].coreid = coreid;
```

```
122                     msg->msg_d2h[core].value = frame;
123                     __asm__ __volatile__ ("trap 4");
124
125                     pixel_cur = (unsigned int *)src[page];
126                     page = page ^ 1;
127
128                     if (frame > FRAMES) {
129                             break;
130                     }
131                     frame++;
132
133                     for (y = 0; y < yres; y++) {
134                             for (x = 0; x < xres; x++) {
135                                     int is_live = fetch_node(src[page
                                            ], x, y, 0, 0);
136                                     int nr_live = 0;
137                                     nr_live += fetch_node(src[page],
                                            x, y, -1, -1);
138                                     nr_live += fetch_node(src[page],
                                            x, y, -1, 0);
139                                     nr_live += fetch_node(src[page],
                                            x, y, -1, 1);
140                                     nr_live += fetch_node(src[page],
                                            x, y, 0, -1);
141                                     nr_live += fetch_node(src[page],
                                            x, y, 0, 1);
142                                     nr_live += fetch_node(src[page],
                                            x, y, 1, -1);
143                                     nr_live += fetch_node(src[page],
                                            x, y, 1, 0);
144                                     nr_live += fetch_node(src[page],
                                            x, y, 1, 1);
145                                     if (is_live) {
146                                             if (nr_live < 2 ||
                                                    nr_live > 3) {
147                                                     *pixel_cur = 0;
148                                             } else {
149                                                     *pixel_cur = 1;
150                                             }
151                                     } else if (nr_live == 3) {
152                                             *pixel_cur = 1;
153                                     } else {
154                                             *pixel_cur = is_live;
155                                     }
156
157                                     pixel_cur++;
158                             }
159                     }
160             }
```

```
161
162         pixel_cur = (unsigned int *)src[page];
163         for (y = 0; y < yres; y++) {
164                 for (x = 0; x < xres; x++) {
165                         msg->pixels[core][y][x] = *pixel_cur;
166                         pixel_cur++;
167                 }
168         }
169
170         msg->msg_d2h[core].finished = 1;
171
172         return 0;
173 }
```

Listing A.7: Implementation of Game of Life using eSDK, epiphany.c

```
1  /*
2  Copyright (c) 2013-2014, Shodruky Rhyammer
3  Copyright (c) 2014, Trygve Aaberge
4  All rights reserved.
5
6  Redistribution and use in source and binary forms, with or
       without modification,
7  are permitted provided that the following conditions are met:
8
9          Redistributions of source code must retain the above
               copyright notice, this
10         list of conditions and the following disclaimer.
11
12         Redistributions in binary form must reproduce the above
               copyright notice, this
13         list of conditions and the following disclaimer in the
               documentation and/or
14         other materials provided with the distribution.
15
16         Neither the name of the copyright holders nor the names
               of its
17         contributors may be used to endorse or promote products
               derived from
18         this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS "AS IS" AND
21 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
       THE IMPLIED
22 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
       PURPOSE ARE
23 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
       CONTRIBUTORS BE LIABLE FOR
```

```
24 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
       CONSEQUENTIAL DAMAGES
25 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
       OR SERVICES;
26 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
       CAUSED AND ON
27 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
       OR TORT
28 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
        USE OF THIS
29 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30
31 Originally code for Mandelbrot computation, licenced as stated
       above, from:
32 https://github.com/parallella/parallella-examples/tree/master/
       mandelbrot
33 Used as basis and modified for this Game of Life implementation.
34 */
35
36
37 #include <stdint.h>
38
39 #define FRAMES 100
40 #define ALIGN8 8
41 #define CORES 16
42 #define ROWS 4
43 #define COLS 4
44 #define XRES 45
45 #define YRES 45
46
47 typedef struct __attribute__((aligned(ALIGN8))) {
48         uint32_t value;
49         uint32_t coreid;
50         uint32_t finished;
51 } msg_dev2host_t;
52
53 typedef struct __attribute__((aligned(ALIGN8))) {
54         uint32_t value;
55 } msg_host2dev_t;
56
57 typedef struct __attribute__((aligned(ALIGN8))) {
58         uint32_t xres;
59         uint32_t yres;
60 } info_t;
61
62 typedef struct {
63         msg_host2dev_t msg_h2d[CORES];
64         msg_dev2host_t msg_d2h[CORES];
65         info_t info;
```

```
66          uint32_t pixels[CORES][YRES][XRES];
67  } msg_block_t;
```

Listing A.8: Implementation of Game of Life using eSDK, shared_data.h

```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdlib.h>
4   #include <unistd.h>
5   #include <fcntl.h>
6   #include <sys/ioctl.h>
7   #include <sys/mman.h>
8   #include <stdint.h>
9   #include <time.h>
10
11  #define FRAMES 100
12  #define XRES 180
13  #define YRES 180
14
15  unsigned int xres, yres;
16  unsigned int pixels[2][YRES][XRES];
17
18  static inline void nano_wait(uint32_t sec, uint32_t nsec)
19  {
20          struct timespec ts;
21          ts.tv_sec = sec;
22          ts.tv_nsec = nsec;
23          nanosleep(&ts, NULL);
24  }
25
26  unsigned int fetch_node(unsigned int page, int x, int y, int
        x_diff, int y_diff) {
27          int x_new = x + x_diff;
28          if (x_new < 0 || x_new >= xres) {
29                  return 0;
30          }
31          int y_new = y + y_diff;
32          if (y_new < 0 || y_new >= yres) {
33                  return 0;
34          }
35          return pixels[page][y_new][x_new];
36  }
37
38  int main(int argc, char *argv[])
39  {
40          struct timespec time;
41          double time0, time1;
42          xres = XRES;
43          yres = YRES;
```

```
44
45          unsigned int row = 0;
46          unsigned int col = 0;
47
48          FILE *file = fopen("grid.txt", "r");
49          if (file == NULL) {
50                  fprintf(stderr, "Could not open grid.txt\n");
51                  exit(1);
52          }
53
54          int x, y, c;
55          int eol = 0;
56          for (y = 0; y < yres; y++) {
57                  for (x = 0; x < xres; x++) {
58                          if (!eol) {
59                                  c = fgetc(file);
60                                  if (c == '\n') {
61                                          eol = 1;
62                                  }
63                          }
64                          if (c == '1') {
65                                  pixels[0][y][x] = 1;
66                          } else {
67                                  pixels[0][y][x] = 0;
68                          }
69                          pixels[1][y][x] = 0;
70                  }
71                  eol = 0;
72          }
73
74          clock_gettime(CLOCK_REALTIME, &time);
75          time0 = time.tv_sec + time.tv_nsec * 1.0e-9;
76
77          unsigned int frame = 1;
78          unsigned int page_last = 0;
79          unsigned int page_cur = 1;
80          while (1) {
81                  if (frame > FRAMES) {
82                          break;
83                  }
84                  frame++;
85
86                  for (y = 0; y < yres; y++) {
87                          for (x = 0; x < xres; x++) {
88                                  int is_live = fetch_node(
                                        page_last, x, y, 0, 0);
89                                  int nr_live = 0;
90                                  nr_live += fetch_node(page_last,
                                        x, y, -1, -1);
```

```
91                                  nr_live += fetch_node(page_last,
                                        x, y, -1, 0);
92                                  nr_live += fetch_node(page_last,
                                        x, y, -1, 1);
93                                  nr_live += fetch_node(page_last,
                                        x, y, 0, -1);
94                                  nr_live += fetch_node(page_last,
                                        x, y, 0, 1);
95                                  nr_live += fetch_node(page_last,
                                        x, y, 1, -1);
96                                  nr_live += fetch_node(page_last,
                                        x, y, 1, 0);
97                                  nr_live += fetch_node(page_last,
                                        x, y, 1, 1);
98                                  if (is_live) {
99                                          if (nr_live < 2 ||
                                                nr_live > 3) {
100                                                 pixels[page_cur][
                                                        y][x] = 0;
101                                         } else {
102                                                 pixels[page_cur][
                                                        y][x] = 1;
103                                         }
104                                 } else if (nr_live == 3) {
105                                         pixels[page_cur][y][x] =
                                                1;
106                                 } else {
107                                         pixels[page_cur][y][x] =
                                                is_live;
108                                 }
109                         }
110                 }
111
112                 page_last = page_cur;
113                 page_cur = page_cur ^ 1;
114         }
115
116         clock_gettime(CLOCK_REALTIME, &time);
117         time1 = time.tv_sec + time.tv_nsec * 1.0e-9;
118         printf("rows: %d, cols: %d\n", yres, xres);
119         printf("frames: %d\n", FRAMES);
120         printf("time: %f sec\n", time1 - time0);
121
122         for (y = 0; y < yres; y++) {
123                 for (x = 0; x < xres; x++) {
124                         printf("%d ", pixels[page_last][y][x]);
125                 }
126                 printf("\n");
127         }
```

```
128
129         return 0;
130 }
```

Listing A.9: Serial implementation of Game of Life, using arrays

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4 #include <unistd.h>
 5 #include <fcntl.h>
 6 #include <sys/ioctl.h>
 7 #include <sys/mman.h>
 8 #include <stdint.h>
 9 #include <time.h>
10
11 #define FRAMES 100
12 #define XRES 180
13 #define YRES 180
14
15 unsigned int xres, yres;
16
17 static inline void nano_wait(uint32_t sec, uint32_t nsec)
18 {
19         struct timespec ts;
20         ts.tv_sec = sec;
21         ts.tv_nsec = nsec;
22         nanosleep(&ts, NULL);
23 }
24
25 unsigned int fetch_node(char *src, int x, int y, int x_diff, int
      y_diff) {
26         int x_new = x + x_diff;
27         if (x_new < 0 || x_new >= xres) {
28                 return 0;
29         }
30         int y_new = y + y_diff;
31         if (y_new < 0 || y_new >= yres) {
32                 return 0;
33         }
34         char *src2 = src + (y_new * yres + x_new) * sizeof(
              unsigned int);
35         return *src2;
36 }
37
38 int main(int argc, char *argv[])
39 {
40         struct timespec time;
41         double time0, time1;
```

```
42          xres = XRES;
43          yres = YRES;
44
45          unsigned int row = 0;
46          unsigned int col = 0;
47
48          char *src[2] = {
49                  malloc(xres * yres * sizeof(unsigned int)),
50                  malloc(xres * yres * sizeof(unsigned int))
51          };
52
53          FILE *file = fopen("grid.txt", "r");
54          if (file == NULL) {
55                  fprintf(stderr, "Could not open grid.txt\n");
56                  exit(1);
57          }
58
59          int x, y, c;
60          int eol = 0;
61          unsigned int *pixel_last = (unsigned int *)src[0];
62          unsigned int *pixel_cur = (unsigned int *)src[1];
63          for (y = 0; y < yres; y++) {
64                  for (x = 0; x < xres; x++) {
65                          if (!eol) {
66                                  c = fgetc(file);
67                                  if (c == '\n') {
68                                          eol = 1;
69                                  }
70                          }
71                          if (c == '1') {
72                                  *pixel_last = 1;
73                          } else {
74                                  *pixel_last = 0;
75                          }
76                          *pixel_cur = 0;
77                          pixel_last++;
78                          pixel_cur++;
79                  }
80                  eol = 0;
81          }
82
83          clock_gettime(CLOCK_REALTIME, &time);
84          time0 = time.tv_sec + time.tv_nsec * 1.0e-9;
85
86          unsigned int frame = 1;
87          unsigned int page = 1;
88          while (1) {
89                  pixel_cur = (unsigned int *)src[page];
90                  page = page ^ 1;
```

```
91
92                  if (frame > FRAMES) {
93                          break;
94                  }
95                  frame++;
96
97                  for (y = 0; y < yres; y++) {
98                          for (x = 0; x < xres; x++) {
99                                  int is_live = fetch_node(src[page
                                        ], x, y, 0, 0);
100                                 int nr_live = 0;
101                                 nr_live += fetch_node(src[page],
                                        x, y, -1, -1);
102                                 nr_live += fetch_node(src[page],
                                        x, y, -1, 0);
103                                 nr_live += fetch_node(src[page],
                                        x, y, -1, 1);
104                                 nr_live += fetch_node(src[page],
                                        x, y, 0, -1);
105                                 nr_live += fetch_node(src[page],
                                        x, y, 0, 1);
106                                 nr_live += fetch_node(src[page],
                                        x, y, 1, -1);
107                                 nr_live += fetch_node(src[page],
                                        x, y, 1, 0);
108                                 nr_live += fetch_node(src[page],
                                        x, y, 1, 1);
109                                 if (is_live) {
110                                         if (nr_live < 2 ||
                                                nr_live > 3) {
111                                                 *pixel_cur = 0;
112                                         } else {
113                                                 *pixel_cur = 1;
114                                         }
115                                 } else if (nr_live == 3) {
116                                         *pixel_cur = 1;
117                                 } else {
118                                         *pixel_cur = is_live;
119                                 }
120
121                                 pixel_cur++;
122                         }
123                 }
124         }
125
126         clock_gettime(CLOCK_REALTIME, &time);
127         time1 = time.tv_sec + time.tv_nsec * 1.0e-9;
128         printf("rows: %d, cols: %d\n", yres, xres);
129         printf("frames: %d\n", FRAMES);
```

```
130         printf("time: %f sec\n", time1 - time0);
131
132         pixel_cur = (unsigned int *)src[page];
133         for (y = 0; y < yres; y++) {
134                 for (x = 0; x < xres; x++) {
135                         printf("%d ", *pixel_cur);
136                         pixel_cur++;
137                 }
138                 printf("\n");
139         }
140
141         free(src[0]);
142         free(src[1]);
143
144         return 0;
145 }
```

Listing A.10: Serial implementation of Game of Life, using a memory space