BTECH 451

Parallella Epiphany

Mong-Fan Wang
6916087

ABSTRACT

This report is a summary of my progress in the fourth-year Bachelor of Technology project. I am working with a senior engineer in Compucon New Zealand, attempting to further the research and development on a relatively new technology named the Epiphany.

Parallel computing is a type of computation where there are many calculations carried out simultaneously. Unlike traditional computing where one problem generally gets solved with one processor, parallel computing takes the problem and divide them into smaller ones if possible, and they get processed at the same time.

Epiphany is a coprocessor with extremely low power consumption, this allows for a very high performance per watt when utilised in parallel computing. My project will verify the claims of the emerging technology, while also attempt in developing some applications for it.

ACKNOWLEDGEMENT

I would like to thank Dr Sathiamoorthy Manoharan for allowing me the opportunity to work with Compucon. I would also like to express my appreciation and gratitude for the Compucon General Manager, TN Chan as well as the Senior Engineer, Dave Fielder and all the staff at Compucon, for providing an extremely enjoyable work environment, assistance and guidance during the year-long project.

TABLE OF CONTENTS

COMPANY

Compucon New Zealand is a wholly New Zealand owned company since April 2011, that manufactures computer systems and has excellent reputation and quality in the reseller and user communities.

Partnering with world class component manufacturers, Compucon offers high quality and reliable computer system builds.

Compucon New Zealand also specialises in high performance parallel computing and this will be the main focus of this project [1].

The company has had many University of Auckland students since 2002 completing projects with them in the Bachelor of Technology program.

PROJECT BRIEF

PURPOSE

The aim of this project is to verify the claims of the Epiphany manufacturer and develop a number of applications to drive this new technology. The programs will be coded in the C or C++ language, as these are low level languages, they communicate with the chip much better than a high-level language like Python.

PEOPLE

TN Chan is the General Manager of Compucon New Zealand, as well as the full supervisor of this project. David Fielder is the Senior Engineer of the company who assists me and provides hands-on guidance in the Compucon House one day a week.

HARDWARE & BACKGROUND

EPIPHANY

A co-processor manufactured by Adapteva, the particular model I am working with has 16 high performance RISC CPU cores, programmable with both C/C++ and OpenCL. Advantages include very low wattage for power consumption and being very flexible in terms of scalability.

PARALLELLA

The co-processor runs on the Parallella board which is a credit card sized board, includes a Gigabit Ethernet connection, HDMI port and 1 GB of SDRAM [2].

EPIPHANY ARCHITECTURE

As the Epiphany is a co-processor, it cannot do everything that a normal CPU can. Instead, it is a simplified processor that carries out specialised tasks. This is a disadvantage of the Epiphany; however, it is also the advantage at the same time as being specific makes it much more energy efficient than a standard CPU. Memory on the Epiphany uses little-endian. Doubles are not supported.

THREADING

- Multi-Threading improving performance [3]
- CPU utilisation is better
- IO latency hiding

THREADED MPI

Threaded MPI is the go to architecture for the Parallella Epiphany. The fully divergent RISC cores allow high performance with inter-core data movement and maximise data re-use [4].

Brown Deer Technology claims that the programming is very easy, performance is great and the libraries are readily available. It is also only going to get easier as technology progresses. Part of my project is also to verify the claims of Brown Deer.

The power efficiency of the Epiphany rivals many other processes in the market today and threaded MPI works perfectly aligned with that goal.

COPRTHR SDK

This term stands for the CO-PRocessing THReads. It is a SDK that provides libraries and tools for developers that are developing multi-core applications. It provides support for the Parallella in OpenCL and STDCL for the Epiphany co-processor [5].

STDCL

This is a portable API for targeting compute offload accelerators and co-processors.

EPIPHANY LAYOUT

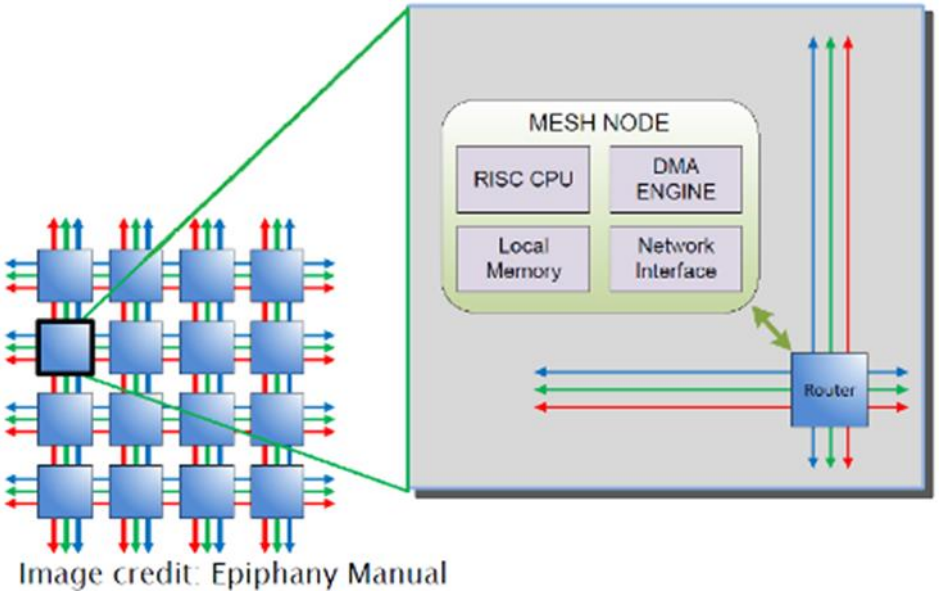


Image credit: Epiphany Manual

[7]

DEVICE SETUP

PARALLELLA SETUP

The following hardware are required:

- Parallella Board
- 4-Port Powered USB Hub
- 8 GB Micro-SD Card with an Adapter
- Micro-USB to USB (Female) Cable
- Micro-HDMI to HDMI (Female) Cable
- Crossover Ethernet Cable [6]

The following software is required:

- https://www.parallella.org/create-sdcard/
    - The Manufacturer has included 4 versions of the Ubuntu image:
        - Desktop Headless
        - Desktop with Display
        - Kickstarter Headless
        - Kickstarter with Display

    For the board I am working with, I will be using both the Desktop Headless and with Display.

SD CARD FORMATTING

The SD card houses the Operating System for the Parallella.

Use the SDFormatter to fully erase the SD card. Now use a Win32 Disk Imager program to load the file containing the Parallella image onto the SD card. It may appear that there are no files in the SD card from Windows Explorer, however, this is normal. Safely eject the SD card and pop it into the Parallella.

EXPANDING IMAGE

This command shows that only a small portion of the SD Card is available for use:

```
df -h
```

By entering the following series of commands, the image will be expanded so that the entire SD card's storage size can be utilised correctly.

```
dmesg | grep "root"
root=/dev/mmcblk0p2
```

`/dev/mmcblk0p2` is the root partition, expand this by entering:

```
fdisk /dev/mmcblk0
```

Enter 'm' for help. Delete partition 2 (root partition), then create a new partition 2. Enter 'd' followed by '2' to delete the root partition. Then 'n' followed by 'p' and 2'' to create a new partition 2. For the first and last sector, select default. Enter 'p' to confirm and write it to disk with 'w'.

Machine is then rebooted with:

```
sudo shutdown -r now
```

After reboot, enter:

```
resize2fs /dev/mmcblk0p2
```

This ensures the resize.

SSH SETUP

WINDOWS

Microsoft Windows does not have built in SSH, this means PuTTY for Windows is used. It can be downloaded from:

- http://www.putty.org/

LINUX

In Linux, SSH is built in the Terminal.

NETWORK CONNECTION

There are two ways to connect to the Parallella board. Finding the IP address assigned to the machine, or assigning a static address to it.

DYNAMIC IP

Find the IP address of the Parallella board by using any sort of network tool that displays all devices connected in a Local Area Network.

STATIC IP

The new headless image uses the following form for the file "/etc/network/interfaces"

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
```

```
source-directory /etc/network/interfaces.d
```

It is better to not alter this. This allows the contents of the folder "/etc/network/interfaces.d" to contain the files "etho" and "lo". The original "etho" reads

```
auto eth0
iface eth0 inet dhcp
```

and should be altered to read (for the Compucon network environment)

```
auto eth0
iface eth0 inet static
address 192.168.1.101
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.254
```

COPRTHR SETUP

WINDOWS

Run the Windows installer from https://github.com/browndeer/coprthr (libstdcl-1.4.0-win7-install.msi) and set the appropriate paths to use the headers and library.

LINUX

Pre-requistites:

- Linux Ubuntu
- libelf-0.8.13.tar.gz (www.mr511.de/software/libelf-0.8.13.tar.gz)
- libevent-2.0.18-stable.tar.gz (github.com/downloads/libevent/libevent/libevent-2.0.18-stable.tar.gz)
- libconfig-1.4.8.tar.gz (www.hyperrealm.com/libconfig/libconfig-1.4.8.tar.gz)
- m4-1.4.16.tar.gz (http://ftp.gnu.org/gnu/m4/)
- flex-2.5.35.tar.gz (http://flex.sourceforge.net/)
- bison-2.5.tar.gz (http://ftp.gnu.org/gnu/bison/)

Pre-compiled Package:

- coprthr-1.5.0-rc2-parallella.tgz

The libraries are unpacked by entering the following commands:

```
./configure
sudo make install
```

Unpacking the file will produce a directory browndeer/.

Enter following commands to remove previous installations as well as installing the new version:

```
sudo ./browndeer/uninstall_coprthr_parallella.sh
sudo ./browndeer/install_coprthr_parallella.sh
```

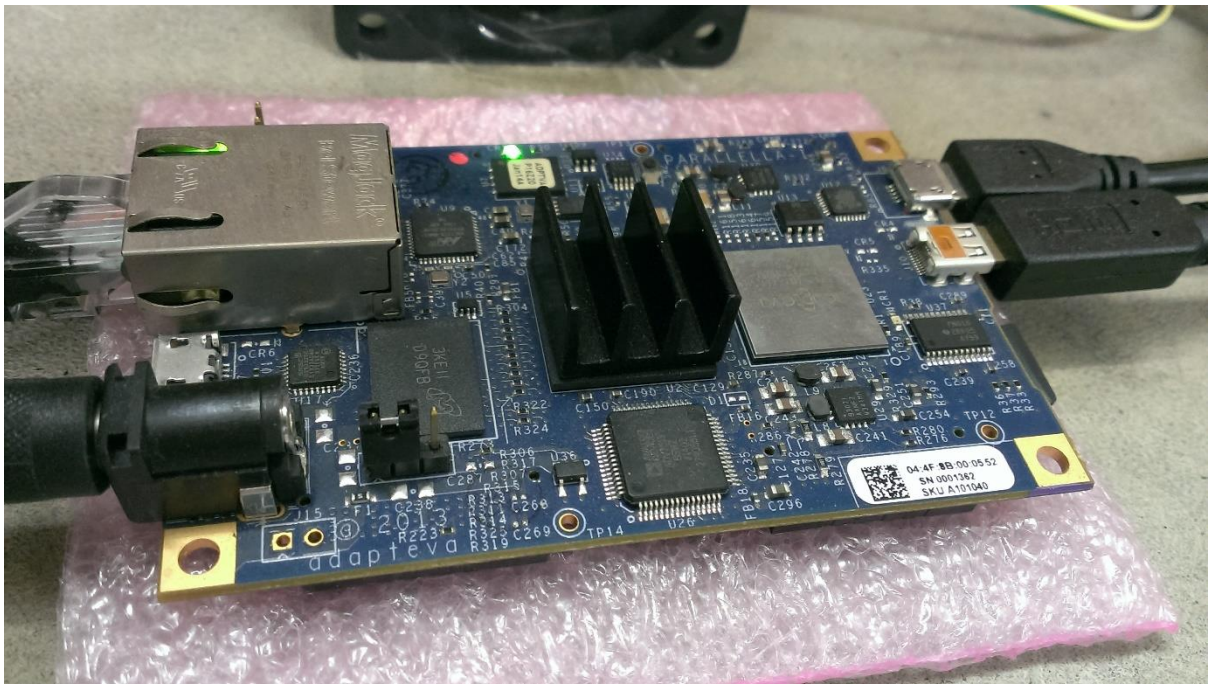
Finally, add the following environmental variables to PATH:

```
export PATH=/usr/local/browndeer/bin:$PATH
export
LD_LIBRARY_PATH=/usr/local/browndeer/lib:/usr/local/lib:$LD_L
IBRARY_PATH
```

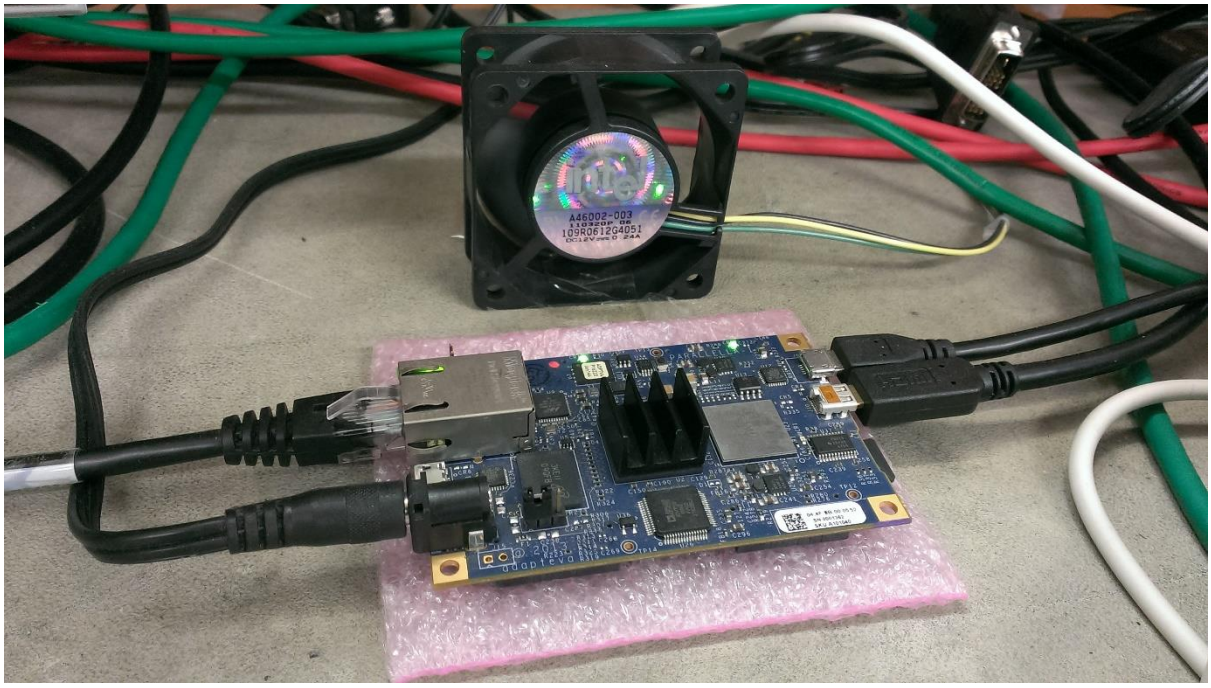
SOFTWARE

The current version of the Parallella Epiphany runs on Linux Ubuntu 15.04, while I will be mainly experimenting with the headless image, meaning no display is utilised. For me to get any sort of feedback is to do the SSH setup I have mentioned earlier to connect to the device and communicate through another computer on the same network. The reason that the image with the HDMI output is not used is due to the fact that it is extremely outdated at 2014 and the latest ESDK and power saving features are not implemented due to challenges with the FPGA HDMI integration.

THE BOARD



This is the parallella board with the Epiphany co-processor. The bottom left is the power input, the top left is the Ethernet port and the two connections on the right are the USB and HDMI connections.



As shown here, a fan must be used at all times to ensure the board and chip do not overheat.

PROGRAMMING

The Parallella uses a host/device structure, meaning every application needs a corresponding program for each side.

While the programs are separate, all files are created and stored on the host (in this case the ARM chip) Below are examples from the Parallella GitHub that I have first initially ran to test for performance and get an idea on how programs are executed on the Parallella Epiphany.

HELLO WORLD EXAMPLE - NATIVE ESDK

DEVICE PROGRAM - E_HELLO_WORLD.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "e_lib.h"

int main(void) {
    const char         ShmName[] = "hello_shm";
    const char         Msg[] = "Hello World from core
0x%03x!";
    char               buf[256] = { 0 };
    e_coreid_t         coreid;
    e_memseg_t         emem;
    unsigned           my_row;
    unsigned           my_col;


    coreid = e_get_coreid();
    e_coords_from_coreid(coreid, &my_row, &my_col);

    if ( E_OK != e_shm_attach(&emem, ShmName) ) {
        return EXIT_FAILURE;
    }

    snprintf(buf, sizeof(buf), Msg, coreid);

    if ( emem.size >= strlen(buf) + 1 ) {
        e_write((void*)&emem, buf, my_row, my_col, NULL,
strlen(buf) + 1);
    } else {
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

HOST PROGRAM - HELLO_WORLD.C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <e-hal.h>

const unsigned ShmSize = 128;
const char ShmName[] = "hello_shm";
const unsigned SeqLen = 20;

int main(int argc, char *argv[])
{
    unsigned row, col, coreid, i;
    e_platform_t platform;
    e_epiphany_t dev;
    e_mem_t   mbuf;
    int rc;

    srand(1);

    e_set_loader_verbosity(H_D0);
    e_set_host_verbosity(H_D0);

    e_init(NULL);
    e_reset_system();
    e_get_platform_info(&platform);

    rc = e_shm_alloc(&mbuf, ShmName, ShmSize);
    if (rc != E_OK)
        rc = e_shm_attach(&mbuf, ShmName);

    if (rc != E_OK) {
        fprintf(stderr, "Failed to allocate shared memory.
Error is %s\n",
                strerror(errno));
        return EXIT_FAILURE;
    }

    for (i=0; i<SeqLen; i++)
    {
        char buf[ShmSize];

        row = rand() % platform.rows;
        col = rand() % platform.cols;
        coreid = (row + platform.row) * 64 + col +
platform.col;
        printf("%3d: Message from eCore 0x%03x (%2d,%2d): ",
i, coreid, row, col);
```

```
        e_open(&dev, row, col, 1, 1);
        e_reset_group(&dev);

        if ( E_OK != e_load("e_hello_world.elf", &dev, 0, 0,
E_TRUE) ) {
            fprintf(stderr, "Failed to load
e_hello_world.elf\n");
            return EXIT_FAILURE;
        }

        usleep(10000);

        e_read(&mbuf, 0, 0, 0, buf, ShmSize);

        printf("\"%s\"\n", buf);
        e_close(&dev);
    }

    e_shm_release(ShmName);
    e_finalize();

    return 0;
}
```

```
linaro@linaro-nano: ~/epiphany-examples/apps/hello-world
File  Edit  Tabs  Help
linaro@linaro-nano:~/epiphany-examples/apps/hello-world$ ./build.sh
linaro@linaro-nano:~/epiphany-examples/apps/hello-world$ ./run.sh
  0: Message from eCore 0x8ca ( 3, 2): "Hello World from core 0x8ca!"
  1: Message from eCore 0x84b ( 1, 3): "Hello World from core 0x84b!"
  2: Message from eCore 0x84b ( 1, 3): "Hello World from core 0x84b!"
  3: Message from eCore 0x888 ( 2, 0): "Hello World from core 0x888!"
  4: Message from eCore 0x849 ( 1, 1): "Hello World from core 0x849!"
  5: Message from eCore 0x88b ( 2, 3): "Hello World from core 0x88b!"
  6: Message from eCore 0x88b ( 2, 3): "Hello World from core 0x88b!"
  7: Message from eCore 0x8ca ( 3, 2): "Hello World from core 0x8ca!"
  8: Message from eCore 0x80a ( 0, 2): "Hello World from core 0x80a!"
  9: Message from eCore 0x808 ( 0, 0): "Hello World from core 0x808!"
 10: Message from eCore 0x8c8 ( 3, 0): "Hello World from core 0x8c8!"
 11: Message from eCore 0x8c9 ( 3, 1): "Hello World from core 0x8c9!"
 12: Message from eCore 0x88a ( 2, 2): "Hello World from core 0x88a!"
 13: Message from eCore 0x88b ( 2, 3): "Hello World from core 0x88b!"
 14: Message from eCore 0x8cb ( 3, 3): "Hello World from core 0x8cb!"
 15: Message from eCore 0x84a ( 1, 2): "Hello World from core 0x84a!"
 16: Message from eCore 0x88a ( 2, 2): "Hello World from core 0x88a!"
 17: Message from eCore 0x84b ( 1, 3): "Hello World from core 0x84b!"
 18: Message from eCore 0x848 ( 1, 0): "Hello World from core 0x848!"
 19: Message from eCore 0x8ca ( 3, 2): "Hello World from core 0x8ca!"
linaro@linaro-nano:~/epiphany-examples/apps/hello-world$
```

DOT PRODUCT EXAMPLE - NATIVE ESDK

DEVICE PROGRAM - E_TASK.C

```
#include <stdio.h>
#include <stdlib.h>
#include "e-lib.h"
#include "common.h"

int main(void)
{
  unsigned *a, *b, *c, *d;
  int i;

  a    = (unsigned *) 0x2000;//Address of a matrix
(transfered here by host)
  b    = (unsigned *) 0x4000;//Address of b matrix
(transfered here by host)
  c    = (unsigned *) 0x6000;//Result
  d    = (unsigned *) 0x7000;//Done

  //Clear Sum
  (*(c))=0x0;

  //Sum of product calculation
  for (i=0; i<N/CORES; i++){
    (*(c)) += a[i] * b[i];
  }

  //Raising "done" flag
  (*(d)) = 0x00000001;

  //Put core in idle state
  __asm__ __volatile__("idle");
}
```

HOST PROGRAM - MAIN.C

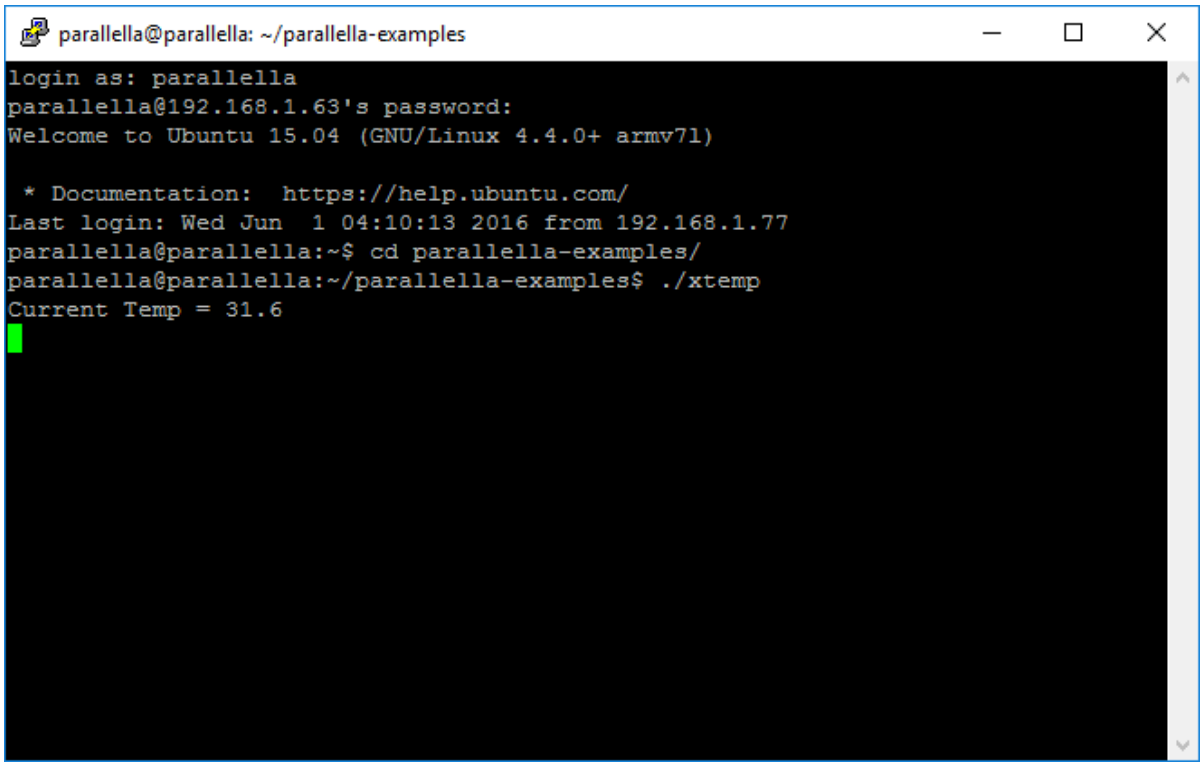
```
#include <stdlib.h>
#include <stdio.h>
#include <e-hal.h>
#include "common.h"

#define RESULT 85344 //recognize /Sum_{i=0}^{n-1} i^2 =
\frac{N(N-1)(2N-1)}{6}

int main(int argc, char *argv[]){
  e_platform_t platform;
  e_epiphany_t dev;

  int a[N], b[N], c[CORES];
```

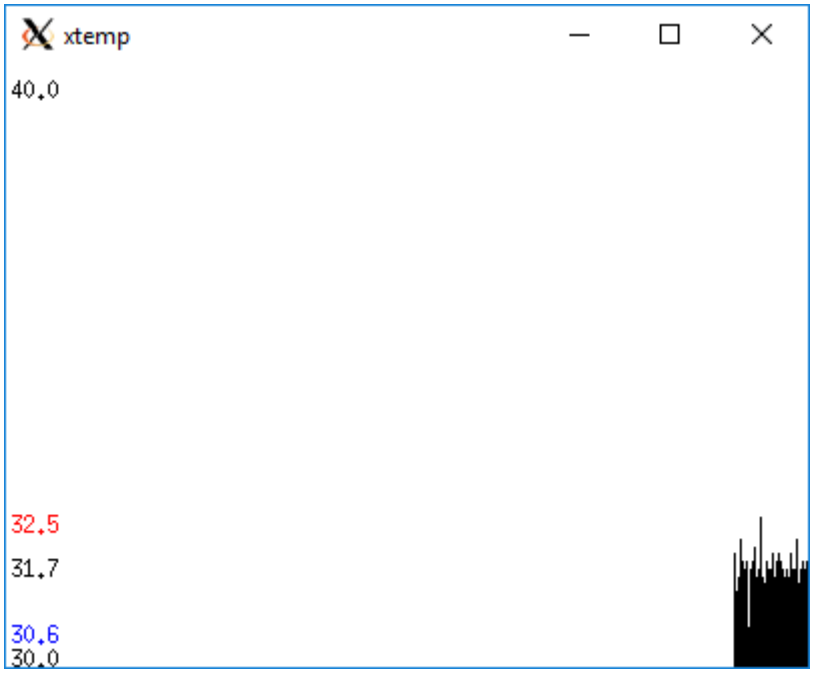
```
  int done[CORES],all_done;
  int sop;
  int i,j;
  int sections = N/CORES; //assumes N % CORES = 0
  unsigned clr = 0;

  //Calculation being done
  printf("Calculating sum of products of two integer vectors
of length %d initalized from (0..%d) using %d Cores.\n",N,N-
1,CORES);
  printf("........\n");

  //Initalize Epiphany device
  e_init(NULL);
  e_reset_system();
//reset Epiphany
  e_get_platform_info(&platform);
  e_open(&dev, 0, 0, platform.rows, platform.cols); //open
all cores

  //Initialize a/b input vectors on host side
  for (i=0; i<N; i++){
    a[i] = i;
    b[i] = i;
  }

  //Load program to cores
  e_load_group("e_task.elf", &dev, 0, 0, platform.rows,
platform.cols, E_FALSE);

  //1. Copy data (N/CORE points) from host to Epiphany local
memory
  //2. Clear the "done" flag for every core
  for (i=0; i<platform.rows; i++){
    for (j=0; j<platform.cols;j++){
      e_write(&dev, i, j, 0x2000,
&a[(i*platform.cols+j)*sections], sections*sizeof(int));
      e_write(&dev, i, j, 0x4000,
&b[(i*platform.cols+j)*sections], sections*sizeof(int));
      e_write(&dev, i, j, 0x7000, &clr, sizeof(clr));
    }
  }

  // start cores
  e_start_group(&dev);

  //Check if all cores are done
  while(1){
    all_done=0;
    for (i=0; i<platform.rows; i++){
      for (j=0; j<platform.cols;j++){
```

```
    e_read(&dev, i, j, 0x7000, &done[i*platform.cols+j],
sizeof(int));
    all_done+=done[i*platform.cols+j];
      }
    }
    if(all_done==CORES){
      break;
    }
  }

  //Copy all Epiphany results to host memory space
  for (i=0; i<platform.rows; i++){
      for (j=0; j<platform.cols;j++){
    e_read(&dev, i, j, 0x6000, &c[i*platform.cols+j],
sizeof(int));
      }
  }

  //Calculates final sum-of-product using Epiphany results as
inputs
  sop=0;
  for (i=0; i<CORES; i++){
    sop += c[i];
  }

  //Print out result
  printf("Sum of Product Is %d!\n",sop);
  //Close down Epiphany device
  e_close(&dev);
  e_finalize();

  if(sop==RESULT){
    return EXIT_SUCCESS;
  }
  else{
    return EXIT_FAILURE;
  }
}
```

xTemp

The xTemp utility is a program under the Parallella Utility package, where the temperature of the board can be visualised. With SSH access, X11 forwarding is needed to see the graphical output on the remote connection.





EPIPHANY BSP - HELLO WORLD - ESBP LIBRARY

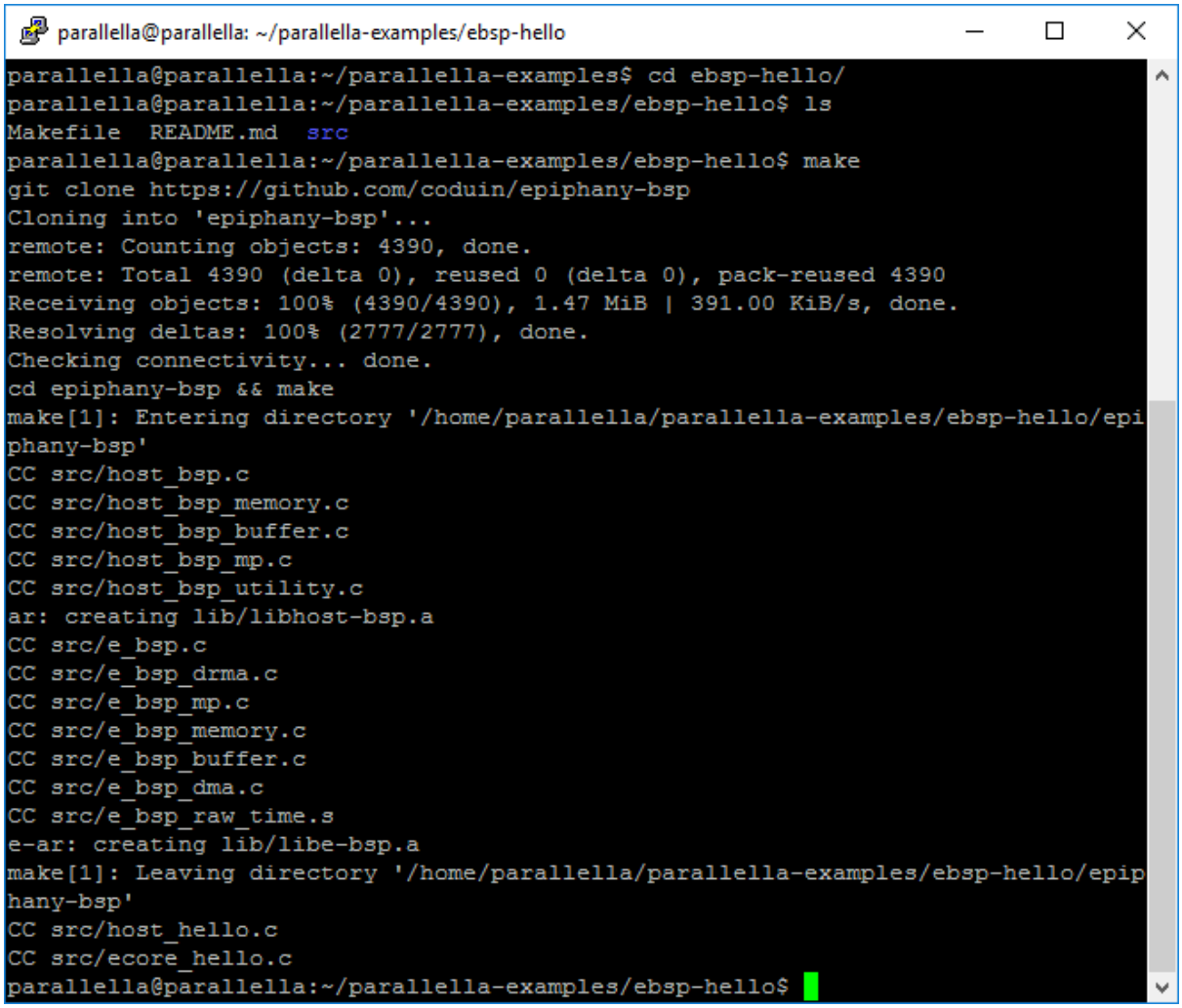
DEVICE PROGRAM - E_CORE_HELLO.C

```
#include <e_bsp.h>

int main()
{
    bsp_begin();

    int n = bsp_nprocs();
    int p = bsp_pid();

    ebsp_message("Hello world from core %d/%d", p, n);

    bsp_end();

    return 0;
}
```

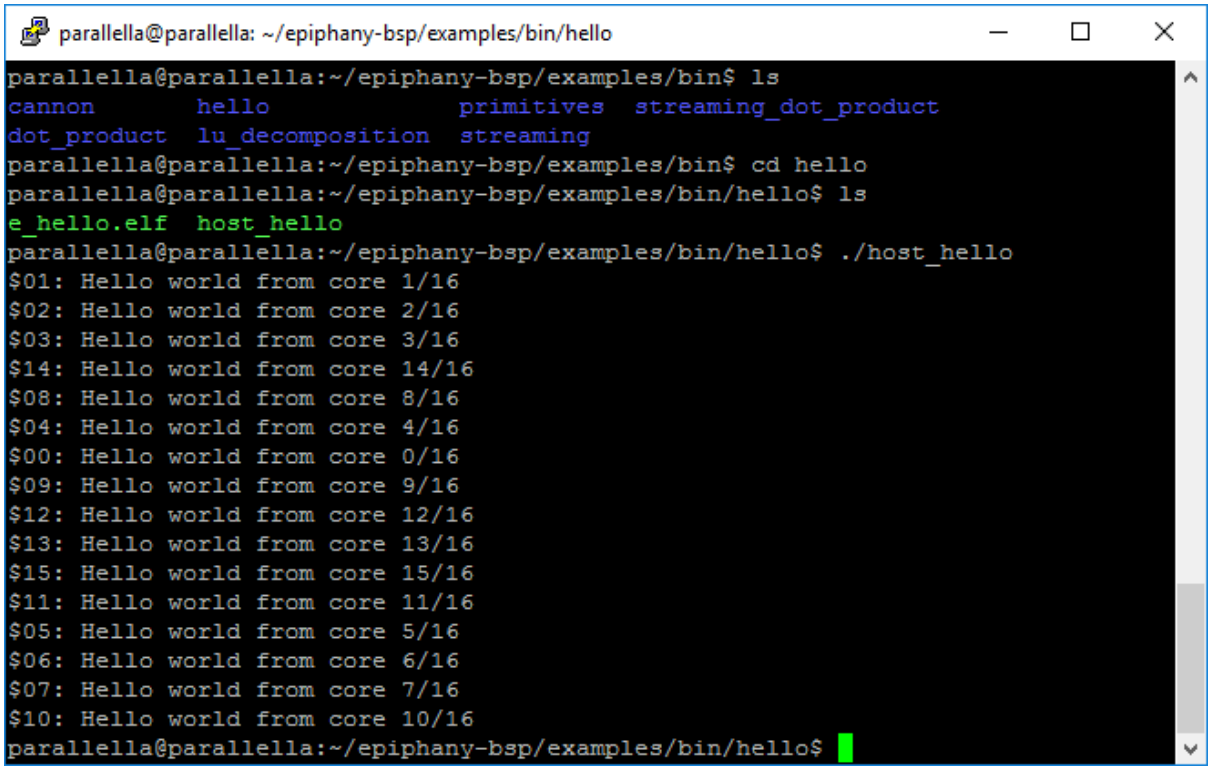
HOST PROGRAM - HOST_HELLO.C

```
#include <host_bsp.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    bsp_init("ecore_hello.srec", argc, argv);

    bsp_begin(bsp_nprocs());

    ebsp_spmd();

    bsp_end();

    return 0;
}
```

RUNNING MAKEFILE



PROGRAM OUTPUT



PARA-PARA EXAMPLE - OPENCL/MPI

OPENCL REQUIREMENTS

```
###Libelf prerequisite
wget www.mr511.de/software/libelf-0.8.13.tar.gz
tar -zxvf libelf-0.8.13.tar.gz
cd libelf-0.8.13
./configure
sudo make install
cd ../

###Libevent prerequisite
wget github.com/downloads/libevent/libevent/libevent-2.0.18-
stable.tar.gz
tar -zxvf libevent-2.0.18-stable.tar.gz
cd libevent-2.0.18-stable
./configure
sudo make install
cd ../

###Libconfig prerequisite
wget www.hyperrealm.com/libconfig/libconfig-1.4.8.tar.gz
tar -zxvf libconfig-1.4.8.tar.gz
cd libconfig-1.4.8
./configure
sudo make install
cd ../

###Install parallella opencl package
wget http://www.browndeertechnology.com/code/coprthr-1.6.0-
parallella.tgz
tar -zxvf coprthr-1.6.0-parallella.tgz
sudo ./browndeer/scripts/install_coprthr_parallella.sh

### Add paths to .bashrc
echo 'export PATH=/usr/local/browndeer/bin:$PATH' >>
~/.bashrc
echo 'export
LD_LIBRARY_PATH=/usr/local/browndeer/lib:/usr/local/lib:$LD_L
IBRARY_PATH' >> ~/.bashrc

### Add paths to root .bashrc
sudo su
echo 'export PATH=/usr/local/browndeer/bin:$PATH' >>
~/.bashrc
echo 'export
LD_LIBRARY_PATH=/usr/local/browndeer/lib:/usr/local/lib:$LD_L
IBRARY_PATH' >> ~/.bashrc
```

```
### Add paths to .cshrc
echo 'setenv PATH /usr/local/bin:$PATH' >> ~/.cshrc
echo 'setenv LD_LIBRARY_PATH
/usr/local/browndeer/lib:/usr/local/lib:$LD_LIBRARY_PATH' >>
~/.cshrc
```

MPI REQUIREMENTS

```
wget http://www.open-
mpi.org/software/ompi/v1.8/downloads/openmpi-1.8.1.tar.gz
tar -zxvf openmpi-1.8.1.tar.gz
cd openmpi-1.8.1
./configure --prefix=/usr/local \
            --enable-mpirun-prefix-by-default \
            --enable-static
make all
sudo make install
```

OPENCL IMPLEMENTATION

```
#define DEVICE_TYPE CL_DEVICE_TYPE_ACCELERATOR


#include <stdlib.h>
#include <stdio.h>
#include <CL/cl.h>

int main()
{
   int i,j;
   int err;
   char buffer[256];

   unsigned int n = 1024;

   cl_uint nplatforms;
   cl_platform_id* platforms;
   cl_platform_id platform;
   //--------------------------------------------------------
-
   //Discover and initialize the platform
   //--------------------------------------------------------
-
   clGetPlatformIDs( 0,0,&nplatforms);
   platforms =
(cl_platform_id*)malloc(nplatforms*sizeof(cl_platform_id));
   clGetPlatformIDs( nplatforms, platforms, 0);

   for(i=0; i<nplatforms; i++) {
      platform = platforms[i];
```

```
clGetPlatformInfo(platforms[i],CL_PLATFORM_NAME,256,buffer,0)
;
      if (!strcmp(buffer,"coprthr")) break;
   }

   if (i<nplatforms) platform = platforms[i];
   else exit(1);
   //---------------------------------------------------------
-
   //Discover and initialize the devices
   //---------------------------------------------------------
-
   cl_uint ndevices;
   cl_device_id* devices;
   cl_device_id dev;

   clGetDeviceIDs(platform,DEVICE_TYPE,0,0,&ndevices);
   devices =
(cl_device_id*)malloc(ndevices*sizeof(cl_device_id));
   clGetDeviceIDs(platform, DEVICE_TYPE,ndevices,devices,0);

   if (ndevices) dev = devices[0];
   else exit(1);


   //---------------------------------------------------------
-
   //Create a context
   //---------------------------------------------------------
-
   cl_context_properties ctxprop[3] = {
      (cl_context_properties)CL_CONTEXT_PLATFORM,
      (cl_context_properties)platform,
      (cl_context_properties)0
   };
   cl_context ctx = clCreateContext(ctxprop,1,&dev,0,0,&err);

   //---------------------------------------------------------
-
   //Create a command queue
   //---------------------------------------------------------
-
   cl_command_queue cmdq =
clCreateCommandQueue(ctx,dev,0,&err);

   //---------------------------------------------------------
-
   //Allocate dynamic memory on the host
   //---------------------------------------------------------
-
```

```
   size_t a_sz = n*n*sizeof(float);
   size_t b_sz = n*sizeof(float);
   size_t c_sz = n*sizeof(float);

   float* a = (float*)malloc(n*n*sizeof(float));
   float* b = (float*)malloc(n*sizeof(float));
   float* c = (float*)malloc(n*sizeof(float));
   for(i=0;i<n;i++) for(j=0;j<n;j++) a[i*n+j] = 1.1f*i*j;
   for(i=0;i<n;i++) b[i] = 2.2f*i;
   for(i=0;i<n;i++) c[i] = 0.0f;

   //------------------------------------------------------------
-
   //Copy data to device buffer
   //------------------------------------------------------------
-
   cl_mem a_buf =
clCreateBuffer(ctx,CL_MEM_USE_HOST_PTR,a_sz,a,&err);
   cl_mem b_buf =
clCreateBuffer(ctx,CL_MEM_USE_HOST_PTR,b_sz,b,&err);
   cl_mem c_buf =
clCreateBuffer(ctx,CL_MEM_USE_HOST_PTR,c_sz,c,&err);

   //------------------------------------------------------------
-
   //The kernel
   //------------------------------------------------------------
-
   const char kernel_code[] =
      "__kernel void matvecmult_kern(\n"
      "   uint n,__global float* a,__global float* b,__global
float* c )\n"
      "{\n"
      "   int i = get_global_id(0);\n"
      "   int j;\n"
      "   float tmp = 0.0f;\n"
      "   for(j=0;j<n;j++) tmp += a[i*n+j] * b[j];\n"
      "   c[i] = a[i*n+i];\n"
      "}\n";


   //------------------------------------------------------------
-
   //Compiling the kernel
   //------------------------------------------------------------
-
   const char* src[1] = { kernel_code };
   size_t src_sz = sizeof(kernel_code);

   cl_program prg = clCreateProgramWithSource(ctx,1,(const
char**)&src,
```

```
          &src_sz,&err);

   clBuildProgram(prg,1,&dev,0,0,0);

   cl_kernel krn =
clCreateKernel(prg,"matvecmult_kern",&err);

   //-----------------------------------------------------
-
   //Set kernel arguments
   //-----------------------------------------------------
-
   clSetKernelArg(krn,0,sizeof(cl_uint),&n);
   clSetKernelArg(krn,1,sizeof(cl_mem),&a_buf);
   clSetKernelArg(krn,2,sizeof(cl_mem),&b_buf);
   clSetKernelArg(krn,3,sizeof(cl_mem),&c_buf);

   //-----------------------------------------------------
-
   //Queue up kernel for execution
   //-----------------------------------------------------
-
   size_t gtdsz[] = { n };
   size_t ltdsz[] = { 16 };
   cl_event ev[10];

clEnqueueNDRangeKernel(cmdq,krn,1,0,gtdsz,ltdsz,0,0,&ev[0]);


   //-----------------------------------------------------
-
   //Readb back result data
   //-----------------------------------------------------
-

clEnqueueReadBuffer(cmdq,c_buf,CL_TRUE,0,c_sz,c,0,0,&ev[1]);
   err = clWaitForEvents(2,ev);

   //-----------------------------------------------------
-
   //Print result
   //-----------------------------------------------------
-
   for(i=0;i<n;i++) printf("c[%d] %f\n",i,c[i]);

   //-----------------------------------------------------
-
   //Release OpenCL resources
   //-----------------------------------------------------
-
   clReleaseEvent(ev[1]);
```

```
    clReleaseEvent(ev[0]);
    clReleaseKernel(krn);
    clReleaseProgram(prg);
    clReleaseMemObject(a_buf);
    clReleaseMemObject(b_buf);
    clReleaseMemObject(c_buf);
    clReleaseCommandQueue(cmdq);
    clReleaseContext(ctx);

    //-------------------------------------------------------
-
    //Free host resources
    //-------------------------------------------------------
-
    free(a);
    free(b);
    free(c);

}
```

MPI IMPLEMENTATION

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(processor_name, &namelen);

  printf("Hello World from MPI Process %d on machine %s\n",
rank, processor_name);

  MPI_Finalize();
}
```

[8]

Using this example, I am able to see how the device handles OpenCL and MPI differently.

PARALLELA EPIPHANY WORKSPACE CREATION

The following commands in the Terminal or PuTTY SSH connection will allow the workspace creation. This will allow for easier programming on the Epiphany.

```
cd ~/Downloads
wget
ftp://ftp.parallella.org/esdk/old/esdk.5.13.07.10_linux_x86_64_armv7l.tgz
sudo mkdir -p /opt/adapteva
sudo mv esdk.5.13.07.10_linux_x86_64_armv7l.tgz /opt/adapteva
cd /opt/adapteva
sudo tar xvf esdk.5.13.07.10_linux_x86_64_armv7l.tgz
sudo ln -sTf esdk.5.13.07.10 /opt/adapteva/esdk
sudo apt-get install libmpfr-dev libgmp3-dev libmpc-dev
openjdk-6-jre tcsh csh g++ -y
sudo nano /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games:/usr/local/games:/opt/adapteva/esdk/tools/e-
gnu/bin"
EPIPHANY_HOME="/opt/adapteva/esdk"
LD_LIBRARY_PATH="/usr/lib:/usr/lib/x86_64-linux-gnu"
cd /usr/lib/x86_64-linux-gnu
sudo cp libmpc.so libmpc.so.2
sudo ldconfig
sudo cp libmpfr.so libmpfr.so.1
sudo cp libgmp.so libgmp.so.3
sudo nano /opt/adapteva/esdk/tools/host/bin/echo-process
Save empty file
sudo chmod 777 /opt/adapteva/esdk/tools/host/bin/echo-process
e-eclipse
```

Create a workspace and name the project. Since the Epiphany is 16 core in this case, we must have the settings of:

Number of rows = 4
Number of columns = 4
Row number in first core = 32
Column number of first core = 8

This creates a master project for all projects that will be created.

To program the Epiphany, we now change the host name to the Epiphany IP address or host name in our network. The 'stop at main' checkbox should be unticked and both 'Resume' and 'Verbose mode' should be ticked.

The final step is to right click the first core project and complete the following:

C/C++ Build → Settings → Epiphany Linker → Linker Description File

Change Select LDF to: ${EPIPHANY_HOME}/bsps/current/legacy.ldf

In the Epiphany Linker, add e-lib to the libraries, then apply these settings to all of the projects. A dialog should pop up showing success messages if completed correctly.

EPIPHANY PROGRAM EXECUTION

Type e-server in the Terminal.
The Epiphany listens on the port 51000 by default.

MEMORY & PERFORMANCE

To run tests on the Epiphany, we must first understand how the device works. The Epiphany has 16 cores, and each of them has a separate DMA Engine. This stands for Direct Memory Access, and is responsible for transferring data between the Epiphany cores. Two DMA channels are present; this means that two pairs of addresses (source and destination) can be set while the CPU continues to work on other tasks. The addresses are also not limited to their own internal memory; it can be any other core within the Epiphany.

Message passing between the cores are different in the Epiphany compared to standard processors. The Epiphany utilises a message queue. This eliminates the need of registering variables. The Epiphany cores have very little local memory but they are fast. Bigger external memory is present but they come at a speed cost.

- local memory: 32 KB for each core
- external memory: 32 MB shared for all cores [10]

A couple of memory transfer tests were done to benchmark the Epiphany memory speed, here are the results:

- Host to Internal Memory
  - Write speed - 14.12 MBps
  - Read speed - 16.95 MBps
- Host to External Memory
  - Write speed - 99.52 MBps
  - Read speed - 120.13 MBps
- Using memcpy
  - Core to Internal Memory
    - Write speed - 487.59 MBps
    - Read speed - 114.85 MBps
  - Core to External Memory
    - Write speed - 139.02 MBps
    - Read speed - 4.15 MBps
- Using DMA:
  - Core to Internal Memory
    - Write speed - 1938.79 MBps
    - Read speed - 478.52 MBps
  - Core to External Memory
    - Write speed - 469.25 MBps
    - Read speed - 151.54 MBps

The transfers to internal memory utilises a single cycle for read and write, however, if the destination and source are on different Epiphany cores, latency increases as the data hops from core to core in the mesh network. The external memory latency is affected by a multitude of variables, including the mesh latency, speed of the RAM as well as the speed of the interface between the FPGA chip on board of the Parallella and the Epiphany.

COPRTHR

Running a vector multiplication tool on the Epiphany is done by allocating each of the vectors to the co-processor memory. Each co-processor then computes the vector that it has been assigned and multiplies each element with the corresponding element in another vector. The result is then set to another vector.

Using the COPRTHR SDK and comparing the performance between the cores of the Epiphany and the main ARM processor on board, the result is that the Epiphany cores working together, handled the tool about three times slower than the ARM processor.

As the COPRTHR SDK is based on the OpenCL framework, there seems to be an unnatural fit between the framework and the hardware, resulting in this poor performance.

EBSP

BSP stands for Bulk Synchronous Parallel, while EBSP is a specialised version developed specifically for the Epiphany, hence the name Epiphany BSP. It is a model where the algorithms use computations that are non-blocking and then a synchronisation even occurs at the end to ensure that all data communications execute in the correct way.

The company that developed EBSP is CODUIN, and they are based in the Netherlands which focuses on software libraries development for multicore embedded systems.

The BSP model was developed in the 1980s, and thee important requirements had to be followed:

- It has n processors capable of computation and communication, i.e. it allows for local memory transactions.
- It has a network in place that allows the different processors to send and receive data.

- It has a mechanism that allows for the synchronisation of these processors, e.g. by means of a blocking barrier.

Consider the EBSP Hello World example again, notice the processor id is out of order here as there is no set logic on which core gets and processes the message first. This is a proof that the program is running in parallel instead of serial. Using the write and read methods of EBSP, we can get better control over how we want to address the memory of each core.

CROSS COMPILATION ENVIRONMENT

To make developing and running applications on the Parallella Epiphany smoother and easier, a cross compilation environment can be setup. The following packages are installed:

```
build-essential git bison flex libgmp3-dev libncurses-dev
libmpc-dev libmpfr-dev texinfo xzip lzip zip
```

The ARM/Linux cross-toolchain is installed from the below packages:

```
gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf
```

The following packages were then installed in the log order:

```
Commit Log for Wed Jul 20 18:46:28 2016

Installed the following packages:(Not a prerequisite but very
necessary)

gedit (3.10.4-0ubuntu4)gedit-common (3.10.4-0ubuntu4)
gir1.2-gtksource-3.0 (3.10.2-0ubuntu1)
gir1.2-peas-1.0 (1.8.1-2ubuntu2)
gnome-user-guide (3.8.2-1)
libgtksourceview-3.0-1 (3.10.2-0ubuntu1)
libgtksourceview-3.0-common (3.10.2-0ubuntu1)
libpeas-1.0-0 (1.8.1-2ubuntu2)
libpeas-common (1.8.1-2ubuntu2)
libpython3.4 (3.4.3-1ubuntu1~14.04.3)
libyelp0 (3.10.2-0ubuntu1)
libzeitgeist-2.0-0 (0.9.14-0ubuntu4.1)
python-gi-cairo (3.12.0-1ubuntu1)
python-zeitgeist (0.9.14-0ubuntu4.1)
yelp (3.10.2-0ubuntu1)
yelp-xsl (3.10.1-1)
zeitgeist (0.9.14-0ubuntu4.1)
zeitgeist-core (0.9.14-0ubuntu4.1)
zeitgeist-datahub (0.9.14-0ubuntu4.1)

Commit Log for Wed Jul 20 18:49:14 2016

Installed the following packages:

binutils (2.24-5ubuntu14.1)
build-essential (11.6ubuntu6)
dpkg-dev (1.17.5ubuntu5.7)
```

```
fakeroot (1.20-3ubuntu2)
g++ (4:4.8.2-1ubuntu6)
g++-4.8 (4.8.4-2ubuntu1~14.04.3)
gcc (4:4.8.2-1ubuntu6)
gcc-4.8 (4.8.4-2ubuntu1~14.04.3)
libalgorithm-diff-perl (1.19.02-3)
libalgorithm-diff-xs-perl (0.04-2build4)
libalgorithm-merge-perl (0.08-2)
libasan0 (4.8.4-2ubuntu1~14.04.3)
libatomic1 (4.8.4-2ubuntu1~14.04.3)
libc-dev-bin (2.19-0ubuntu6.9)
libc6-dev (2.19-0ubuntu6.9)
libdpkg-perl (1.17.5ubuntu5.7)
libfakeroot (1.20-3ubuntu2)
libfile-fcntllock-perl (0.14-2build1)
libgcc-4.8-dev (4.8.4-2ubuntu1~14.04.3)
libitm1 (4.8.4-2ubuntu1~14.04.3)
libstdc++-4.8-dev (4.8.4-2ubuntu1~14.04.3)
libtsan0 (4.8.4-2ubuntu1~14.04.3)
linux-libc-dev (3.13.0-92.139)
make (3.81-8.2ubuntu3)
manpages-dev (3.54-1ubuntu1)

Commit Log for Wed Jul 20 18:50:05 2016

Installed the following packages:

git (1:1.9.1-1ubuntu0.3)
git-man (1:1.9.1-1ubuntu0.3)
liberror-perl (0.17-1.1)

Commit Log for Wed Jul 20 18:50:43 2016

Installed the following packages:

bison (2:3.0.2.dfsg-2)
libbison-dev (2:3.0.2.dfsg-2)
libsigsegv2 (2.10-2)
m4 (1.4.17-2ubuntu1)

Commit Log for Wed Jul 20 18:51:23 2016

Installed the following packages:

flex (2.5.35-10.1ubuntu2)
libfl-dev (2.5.35-10.1ubuntu2)

Commit Log for Wed Jul 20 18:52:11 2016
```

Installed the following packages:

libgmp-dev (2:5.1.3+dfsg-1ubuntu1)
libgmp3-dev (2:5.1.3+dfsg-1ubuntu1)
libgmpxx4ldbl (2:5.1.3+dfsg-1ubuntu1)

Commit Log for Wed Jul 20 18:53:26 2016

Installed the following packages:

libncurses5-dev (5.9+20140118-1ubuntu1)
libtinfo-dev (5.9+20140118-1ubuntu1)

Commit Log for Wed Jul 20 18:54:55 2016

Installed the following packages:

libmpc-dev (1.0.1-1ubuntu1)
libmpfr-dev (3.1.2-1)

Commit Log for Wed Jul 20 18:55:40 2016

Installed the following packages:

libintl-perl (1.23-1build1)
libtext-unidecode-perl (0.04-2)
libxml-libxml-perl (2.0108+dfsg-1ubuntu0.1)
libxml-namespacesupport-perl (1.11-1)
libxml-sax-base-perl (1.07-1)
libxml-sax-expat-perl (0.40-2)
libxml-sax-perl (0.99+dfsg-2ubuntu1)
texinfo (5.2.0.dfsg.1-2)

Commit Log for Wed Jul 20 18:58:00 2016

Installed the following packages:

xzip (1:1.8.2-3)

Commit Log for Wed Jul 20 18:58:29 2016

Installed the following packages:

lzip (1.14-2)

Commit Log for Wed Jul 20 18:59:59 2016

Installed the following packages:

binutils-arm-linux-gnueabihf (2.24-5ubuntu13cross1.98.1)
cpp-4.8-arm-linux-gnueabihf (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
cpp-arm-linux-gnueabihf (4:4.8.2-1)
gcc-4.8-arm-linux-gnueabihf (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
gcc-4.8-arm-linux-gnueabihf-base (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
gcc-4.8-multilib-arm-linux-gnueabihf (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
gcc-arm-linux-gnueabihf (4:4.8.2-1)
libasan0-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libatomic1-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libc6-armel-armhf-cross (2.19-0ubuntu2cross1.104)
libc6-armel-cross (2.19-0ubuntu2cross1.104)
libc6-armhf-cross (2.19-0ubuntu2cross1.104)
libc6-dev-armel-armhf-cross (2.19-0ubuntu2cross1.104)
libc6-dev-armel-cross (2.19-0ubuntu2cross1.104)
libc6-dev-armhf-cross (2.19-0ubuntu2cross1.104)
libgcc-4.8-dev-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libgcc1-armhf-cross (1:4.8.4-2ubuntu1~14.04.1cross0.11.2)
libgomp1-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libsfasan0-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libsfatomic1-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
libsfgcc-4.8-dev-armhf-cross (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
libsfgcc1-armhf-cross (1:4.8.4-2ubuntu1~14.04.1cross0.11.2)
libsfgomp1-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)
linux-libc-dev-armel-cross (3.13.0-12.32cross1.104)
linux-libc-dev-armhf-cross (3.13.0-12.32cross1.104)

Commit Log for Wed Jul 20 19:01:01 2016

Installed the following packages:

g++-4.8-arm-linux-gnueabihf (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
g++-4.8-multilib-arm-linux-gnueabihf (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
g++-arm-linux-gnueabihf (4:4.8.2-1)
libsfstdc++-4.8-dev-armhf-cross (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
libsfstdc++6-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)

```
libstdc++-4.8-dev-armhf-cross (4.8.4-
2ubuntu1~14.04.1cross0.11.2)
libstdc++6-armhf-cross (4.8.4-2ubuntu1~14.04.1cross0.11.2)

Commit Log for Thu Jul 21 15:18:09 2016

Installed the following packages:

gnome-system-monitor (3.8.2.1-2ubuntu1)
libatkmm-1.6-1 (2.22.7-2ubuntu1)
libcairomm-1.0-1 (1.10.0-1ubuntu3)
libglibmm-2.4-1c2a (2.39.93-0ubuntu1)
libgtkmm-3.0-1 (3.10.1-0ubuntu2)
libpangomm-1.4-1 (2.34.0-1ubuntu1)
libsigc++-2.0-0c2a (2.2.10-0.2ubuntu2)

Commit Log for Thu Jul 28 18:18:54 2016

Installed the following packages:

guile-1.8 (1.8.8+1-8ubuntu3)
guile-1.8-libs (1.8.8+1-8ubuntu3)

Commit Log for Wed Aug  3 16:13:14 2016

Installed the following packages:

autoconf (2.69-6)
automake (1:1.14.1-2ubuntu1)
autotools-dev (20130810.1)

Commit Log for Fri Aug  5 12:58:53 2016

Installed the following packages:

libltdl-dev (2.4.2-1.7ubuntu1)
libtool (2.4.2-1.7ubuntu1)
```

The following is then executed with the following settings and environment

Environment settings:

- ESDK_BUILDROOT=/home/username/epiphany-sdk
- ESDK_DESTDIR=/home/username/epiphany-sdk/esdk.2016.3.1/

Build settings:

- eSDK install directory: /home/username/epiphany-sdk/esdk.2016.3.1/
- eSDK prefix directory: /opt/adapteva/esdk.2016.3.1

- epiphany-libs host prefix: arm-linux-gnueabihf
- Build version: 2016.3.1
- Build from branch or tag: 2016.3

```
export EPIPHANY_BUILD_HOME=$HOME/epiphany-sdk
cd $EPIPHANY_BUILD_HOME
sdk/build-epiphany-sdk.sh
```

Using this environment means that the building and running of programs on the Epiphany no longer gets bottlenecked by the device, and instead, relies on the performance of the host machine this is installed on. This makes testing much easier on the Epiphany.

DISCUSSION

When working with the Parallella Epiphany, I found that most of the documentation were either out-of-date of incorrect in some cases, even when they were provided officially by the company or third parties developing specifically for the device. The lack of popularity and "well-knowness" of the hardware contributes to this phenomenon and that help and assistance were hard to find scouring the web. Most of the tasks completed were done with the guidance of the in-house Engineer and he mentioned the issues of binaries being out-of-date as well as instructions provided incorrectly by the developers and that he had spent a long time debugging to fix some of the problems.

The goal of the Epiphany was mainly to provide high performance at a low wattage. As people already know, the Epiphany is best suited for performing parallel tasks, using the device the same way as an Intel or AMD CPU would be a complete disaster. While the device excels at some parallel tasks, if a processor that draws around the same amount of power performs better, or even just as well, then the entire point of using the Epiphany is lost. Using the COPRTHR SDK, performance was disappointing and this was because of the natural unfit of the OpenCL framework with the Epiphany.

People ask that "What is it that is holding the Epiphany back from being the 'new' supercomputer?" and this is a valid question, if the device utilises such low wattage and outputs a respectable amount of power, why is it not being used in popular fashion? There is no clear answer to the question, as it turns out, the technology looks to be immature at this stage and the Epiphany seems to only satisfy a certain niche. There are numerous problems with the Parallella Epiphany and Adapteva must set out to fix them before mass adoption takes place in the commercial area.

Programming the Epiphany requires the correct setup of environment. Any details done incorrectly and the device will throw an error at you and most of the time it is not at all obvious how to debug it. In the times that I have met obstacles in the research and test phases, I usually just try to do the entire setup from scratch, hoping that a user error was made during my steps instead of a fundamental program problem. Sometimes, even accessing the wrong partitions or memory segments, cause the Epiphany to freeze, and a manual reboot is required. Since there is no GUI in the build I was using, it meant that pulling the power cord and re-plugging was the only solution.

CONCLUSION

The project involved a brand-new computing hardware device developed by a small company based in the USA. It focused on parallel computing and its advantages as well as its disadvantages. The Parallella Epiphany has proven itself to be an extremely tricky device to work with, from not having a GUI if choosing to use the latest features with the latest updates, to incomplete documentation provided officially. The goal was to verify its ability stated by the Epiphany manufacturer, Adapteva, and while for the most part, including the power usage, the individual core performance, were certainly impressive, the advantages over other embedded systems, I am still unable to be fully confident in determining if the Epiphany has got the factor to beat every other system. In future work, running the Parallella in clusters may be extremely interesting in figuring out the potential uses for the device as it could potentially increase performance by a large scale.

REFERENCES

1. Compucon.co.nz. (2009). Compucon Computers NZ - Quality Servers and Workstations - Company Profile. [online] Available at: http://www.compucon.co.nz/content/view/27/242/.
2. Adapteva, (2016). Epiphany Datasheet [online] Available at: http://adapteva.com/docs/e16g301_datasheet.pdf.
3. Graham E Fagg. (2006). CS594 Lecture Slides http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2006/Lect07-extra.pdf
4. Brown Deer Technology. (2013). CO-PRocessing THReads (COPRTHR) Software. http://www.browndeertechnology.com/coprthr.htm
5. Brown Deer Technology, (2013). COPRTHR API Reference. [online] Available at: http://www.browndeertechnology.com/docs/coprthr_api_ref.pdf.
6. Suzannejmatthews.github.io. (2016). Technical Musings : Parallella Setup Tutorial. [online] Available at: http://suzannejmatthews.github.io/2015/05/29/setting-up-your-parallella/.
7. Adapteva, (2011). Epiphany Architecture Reference. [online] Available at: http://www.adapteva.com/docs/epiphany_arch_ref.pdf.
8. Parallella, Parallella Examples, (2016). https://github.com/parallella/parallella-examples/tree/master/para-para
9. Jan-Willem Buurlage, Tom Bannink, Abe Wits, (25 Aug 2016). Bulk-synchronous pseudo-streaming algorithms for many core accelerators. https://arxiv.org/pdf/1608.07200v1.pdf
10. CODUIN, (2016). Benchmarking the Parallella. http://blog.codu.in/parallella/epiphany/ebsp/2016/03/02/benchmarking-the-parallella.html