UNIVERSITY OF
Southampton
Faculty of Physical Sciences and Engineering
School of Electronics and Computer Science

Acceleration of High-Performance Computing with OpenCL on FPGA

by

Yeqiu Tang
(yt2n16)

September 2017

Project supervisor: **Professor Mark Zwolinski**

Second examiner: **Dr Nick R Harris**

A dissertation submitted in partial fulfilment of the degree of
MSc Embedded Systems

Abstract

As the clock speed of CPU is static for more than 10 years, the demand for high-performance computing is increasing. Currently, FPGA-based accelerators have evolved to be a better solution than traditional accelerators in modern computing systems.

This project aims to develop a proper OpenCL application to demonstrate and evaluate the performance of the accelerated application to see if the FPGA-based accelerators are suitable for high-performance computing. For comparison, both of the applications in this project have a C++ version and an OpenCL version.

Firstly, based on the brute-force string searching algorithm, a trail application, String Search was developed in order to get the necessary developing experience of OpenCL. The whole development process benefits the further research on OpenCL for FPGA.

Secondly, a fully functional application, Local DNA Sequence Alignment which is based on Smith-Waterman Algorithm is developed. This is the main part of this project that is used to analyse the performance between after and before acceleration. In addition, because only a part of this program is accelerated, the definition of the proportion of parallel part of a program is discussed.

Finally, by analysing and comparing the speedup of the programs on FPGA platform, the prospect of the FPGA-based accelerator is also discussed. What is more, both of the algorithms above are researched specifically. Moreover, a series of experiments are taken and the results that prove the performance improvement on FPGA are well shown in this thesis.

Keywords: CPU, FPGA, Accelerator, C++, OpenCL, Brute-force, String Search, DNA Sequences Alignment, Smith-Waterman Algorithm

Acknowledgement

When the final thesis is about to complete, my heart is filled with gratitude.

First of all, I would like to thank my supervisor, Professor Mark Zwolinski, who not only raised the idea of this project but also gave me a lot of guidance and valuable advice in the project developing and thesis writing, so that the project and report can proceed smoothly.

Also, I would like to thank all my tutors who taught me a lot about embedded systems. Their teaching and help gave me the chance to have enough relative knowledge to handle this project.

Then I would like to thank my seniors, Nan Cao and Li Sun. Their kind assistances helped me to overcome the difficulties I faced in this project and even in life.

I would like to thank my parents, who are significant to me. Their support and care, trust and encouragement, supervision and comfort help me go through all the honor and frustration, make me never lose myself in prosperity nor easily give up in trouble.

Last but not the least, I would like to thank all my friends. When I was under the high pressure, they were always able to make me feel relaxed.

Finally, my achievement was never without the help from everyone mentioned above and I feel so grateful for all of them.

Contents

List of Figures

List of Tables

1. Introduction

1.1Background

When the clock speed of CPU is nearly static for more than 10 years, as shown in Figure 1 [1], the demand for high-performance computing is increasing. It is obviously the traditional processor (CPU) as the core of the computing system is facing unprecedented challenges.

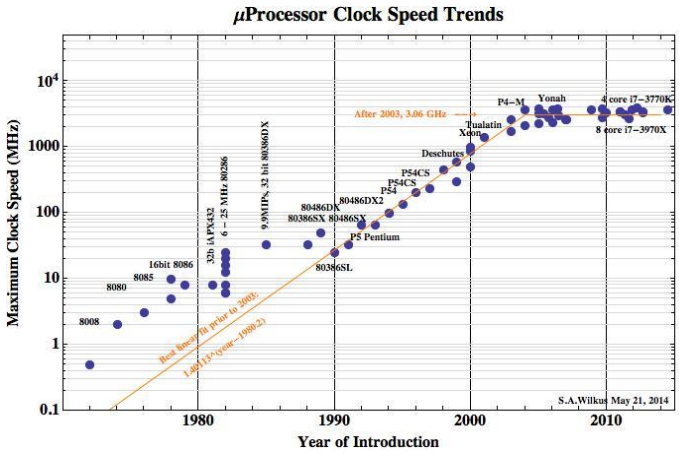


Figure 1 The trends of CPU clock speeds

Currently, FPGA-based accelerators have evolved to be a better solution than traditional accelerators such as GPU in modern computing systems. Also, they are able to supply high computing capabilities, consume low power and reduce the size of device area. And high-level synthesis (HLS) can be used to program an FPGA directly using high-level languages (HLL) such as C, C++, SystemC, and OpenCL.

Therefore, this project focuses on evaluating the performance of heterogeneous computing systems with FPGA-based accelerator by developing and testing some applications with OpenCL on Intel FPGA.

Additionally, in this project, the DNA Sequence Alignment based on Smith-Waterman Algorithm was eventually chosen to be one of the applications. Before that, in order to get a good knowledge of Intel FPGA SDK for OpenCL, a relatively simpler application, String Searching was developed for researching.

What is more, the research and development platform contains three Hewlett Packard Enterprise's BladeSystem Blade Servers equipped with three OpenCL FPGA Cards of Nallatech. Moreover, the operating system is Red Hat Enterprise Linux Server release 6.9 (Santiago) and an authorized Intel FPGA SDK for OpenCL is included.

1.2Aim and Objective

The main aim of this project is to develop the applications based on FPGA in order to validate if they could obtain a better performance and save resources than traditional applications based on CPU. In this project, the OpenCL will be used to develop the applications and deploy it to FPGA.

The secondary aim is to find other possible uses of the FPGA-based accelerators. It is definitely worth to find out more functions of FPGA accelerators if time permits.

In terms of the aims above, the objectives are listed below:

- Evaluation of the technologies

The main connector technologies of the FPGA accelerator cards are PCIe and CAPI. There are many FPGA accelerator cards are able to be used but the processing speed depends not only on the data bus but also on the storage media. Since the Hewlett Packard Enterprise's BladeSystem Blade Servers and OpenCL FPGA Cards of Nallatech have been supplied by Professor Mark Zwolinski, this project focuses more on the OpenCL applications development and performance analysis.

- Research on the algorithm

To implement sequence alignment, the Smith-Waterman Algorithm is widely used all over the world. A particular research on this algorithm makes the development procedure much easier.

- Development of a simple application

It is significant to build a trial application for testing in early development period to start. But the applications in official examples may not be helpful to understanding the use of OpenCL in this project. Therefore developing a simple string searching application is a great way to learn and be familiar with OpenCL.

- Development of a fully functional application

When the evaluation and validation of the trial application are finished, the fully functional application, sequence alignment with Smith-Waterman Algorithm is about to implement. This application is also used to examine the performance of the accelerator system.

- Optimization of the whole system

It is necessary to optimize the prototype system in order to obtain an extreme performance.

- Evaluation of the performance

The evaluation of the performance is important to determine whether the FPGA accelerator cards are able to accelerate computing process compared with traditional computing systems. In this project, the running time of applications is used as the standard of performance.

- Exploration of extra functions

Because of the time limit, there must be other types of useful functions that can be implemented by FPGA accelerators. Thus it is necessary to discover more.

2. Literature Review and Related Research

2.1Heterogeneous Computing

Heterogeneous computing mainly refers to the computing system composed of computing units using different types of instruction set and architecture. Common computing unit categories include CPU and other coprocessors like DSP, ASIC, FPGA and so on. The normal architecture of a heterogeneous computing system [2] is shown below.

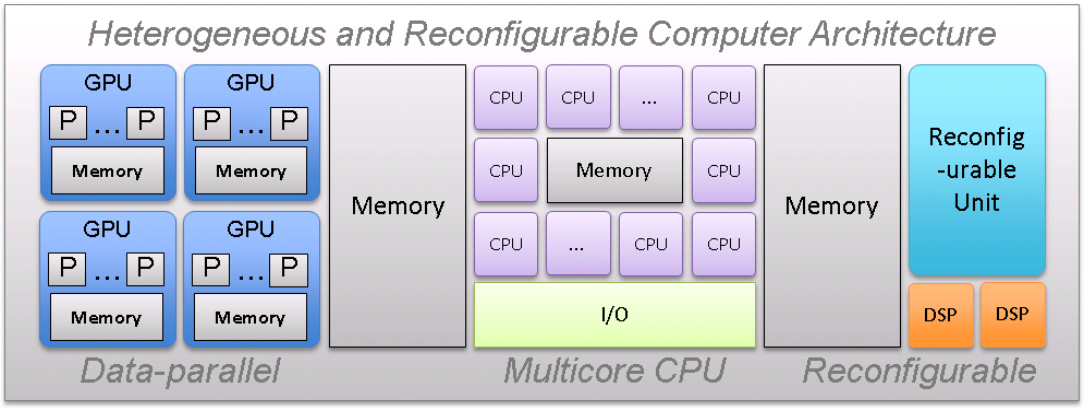


Figure 2 The architecture of a heterogeneous computing system

In recent years, heterogeneous computing has received more attention, mainly because of the traditional way of increasing computing power by increasing the CPU clock frequency and the number of cores, encountered bottlenecks of heat dissipation and energy consumption. At the same time, GPU and other special computing units whose working frequency is relatively low, with more kernel number and parallel computing ability, the overall performance of chip/area and performance/power ratios are very high, but far from being fully utilized.

In general, there are various different levels of heterogeneous computing platform, in addition to the hardware layer, internet instruction set, memory hierarchy, software layer application binary interface and API language, the different characteristics of the underlying implementation, for applications and services that are heterogeneous.

From a point of view on implementation, heterogeneous computing is the development of a series of software and hardware standards that allow different types of computing devices to share the process and results of computing. Meanwhile, the process of optimizing and speeding up the computation will make it more efficient.

2.2OpenCL on FPGA

2.2.1 OpenCL

OpenCL (Open Computing Language) is the first open and free standard for parallel programing of heterogeneous computing systems. It is also a unified programing environment for software developers to work on high-performance computing servers, desktop computing systems (CPU), graphic processors (GPU), Cell type architectures and digital signal processors (DSP) and other parallel processors [3]. It has broad prospects for development.in the game, entertainment, scientific research, medical and other fields.

The OpenCL application contains two parts (host and kernel). OpenCL main program is a pure software routine, written in standard C/C ++, can run on any type of microprocessor. For example, such a processor can be an embedded soft-core processor in an FPGA, a hard-core ARM processor, or an external x86 processor. At some point during the execution of this master software routine, a function may require a large amount of computation, which can take advantage of the parallel acceleration of parallel devices such as CPU, GPU, FPGA, and so on. The function to be accelerated is called the OpenCL kernel. These cores are written using standard C (C99).

Developing FPGA design using the OpenCL description has many advantages over traditional methods based on HDL design. The process of developing software programmable devices generally involves conceiving, programing algorithms in high-level languages such as C or C++, and then using an automatic compiler to build instruction streams. Intel FPGA SDK for OpenCL provides a design environment that makes it easy to implement OpenCL applications on FPGA, as shown below [4].

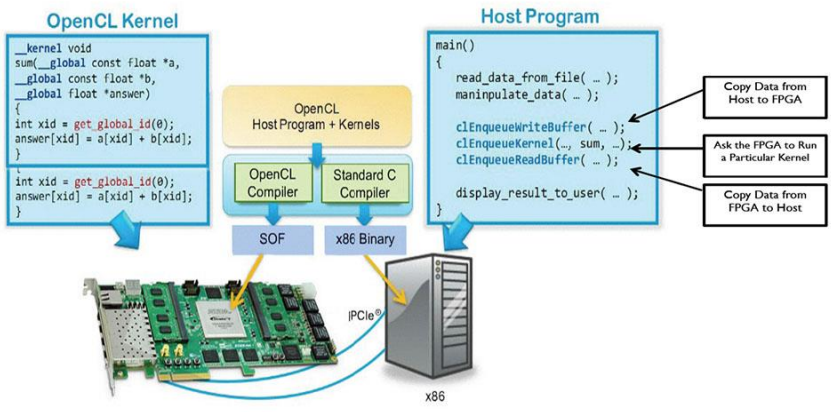


Figure 3 The OpenCL implementation on FPGA

2.2.2 OpenCL Architecture

The above is a general introduction of what OpenCL is, following the architecture of OpenCL is summed up [5]:

a. Hardware Layer

Figure 4 is the abstraction of the OpenCL hardware layer:

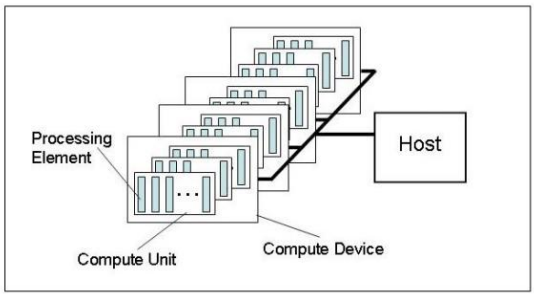


Figure 4 OpenCL Hardware Layer

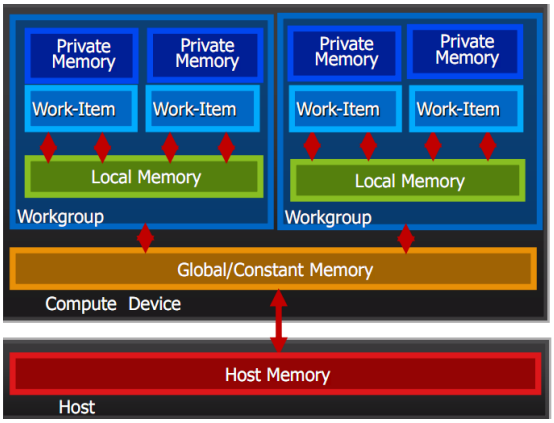


Figure 5 OpenCL Memory Layer

This is a Host (control processing unit, usually by a CPU) and a Compute Device (computing processing unit, usually by FPGA, GPU or other supported chips). The Compute Device is cut into many independent participations, Processing Elements, to process data. Some Processing Element can be formed into a group as a Compute Unit, where shared memory can conveniently move between the elements. In addition, only the elements in one unit can achieve synchronous operation.

b. Memory Layer

As shown in Figure 5 above, in the memory layer, the Host has its own Host Memory, while the Compute Device is more complicated. First of all, there is a Global/Constant memory, which can be used by all, the access is usually the fastest but most scarce. Then each element has its own memory, this is the private memory.

What is more, a group of elements shares a local memory. In the detailed analysis, this is an efficient and elegant memory organization. Data can flow along with channels of Host → Global → Local → Private (which may cross many hardwares).

c. Software Layer

The software layer contains the corresponding data type in the Intel FPGA SDK for OpenCL. The following diagram illustrates the programing framework of OpenCL [6].

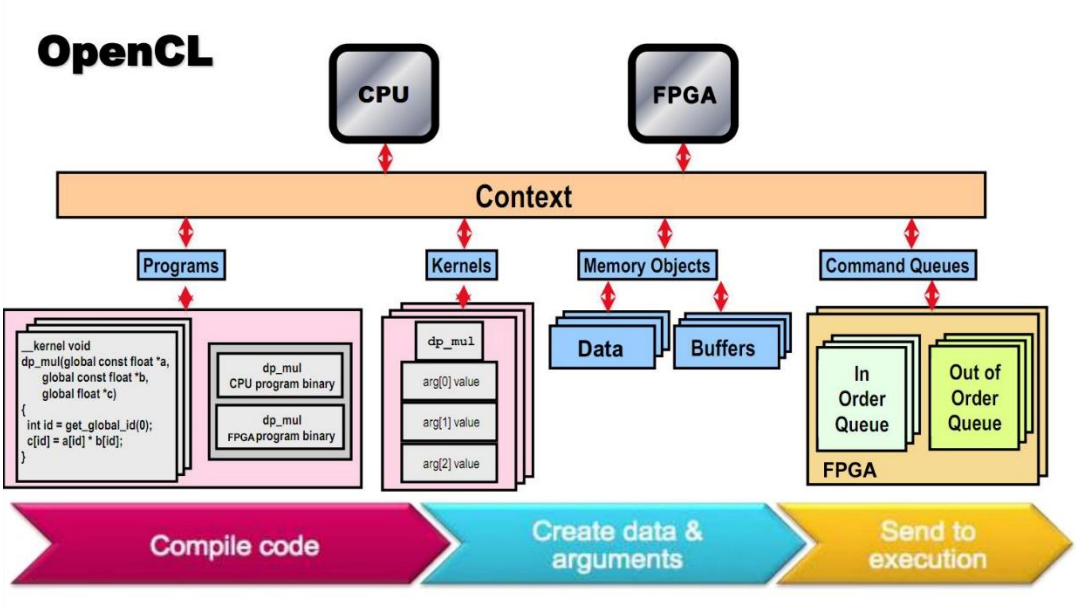


Figure 6 The programing framework of OpenCL

As can be seen from the diagram above (refer to the OpenCL Programing Guide):

- Heterogeneous computing devices can be CPU or FPGA.
- OpenCL API is linked by Context (environment context).
- The program runs in the steps of Compile → Create arguments → Execution.

The concept of some technical terms:

- **Platform**: this is a combination of the host and a number of devices managed by the OpenCL framework that forms the platform. Through it the application can share resources with the device and execute the kernel on the device. In practice, a vendor basically corresponds to a Platform, such as Intel and AMD.
- **Device**: The official explanation of Device is the collection of Compute Units. For example, FPGA is a typical device. The multi-core CPU of Intel or AMD also provides the OpenCL interface, so it can also be used as Device.
- **Context**: a Context includes several devices (FPGA), which is a link between devices. It is an OpenCL environment for sharing and using resources on Platform, including kernels, devices, memory, objects, command queue, and so on. In use, a Platform usually corresponds to a Context.

- **Command Queue**: This is a sequence of instructions submitted to each Device. In the queue, the instructions can be executed in order or not. A device can correspond to multiple instruction queues.
- **Program**: The OpenCL program consists kernel functions, other functions, and declarations. OpenCL is a dynamic compiler, the code is compiled to generate an intermediate file (which can be implemented as a virtual machine code, assembly code or hardware description code), linking to send the program into processors when in use.
- **Kernel**: This is a group of functions and their parameters running in an element. The function can be called from the host side and run on the device side.
- **Work-item**: This is a Processing Element on the hardware, the main computing unit.
- **Work-group**: The existence of work-group is to allow work-item communication and collaboration between It reflects the organization of work-item (work-group is organized in the form of N dimensional grids, N=1, 2 or 3).
- **Events**: In such a distributed computing environment, synchronization between different units is a big problem, and the event is used for synchronization.
- **Memory Object**: The objects that pass data between the host and the device are mapped to global memory in the OpenCL program. There are two specific types: Buffer, Object (cache object) and Image Object (image object).
- **NDRange**: The N-dimensional range where N=1, 2 or 3 and is the main interface of the host side running the device side kernel function. In fact, there are others, but NDRange is very common, used in packet computing. The structure of NDRange [7] is shown below:

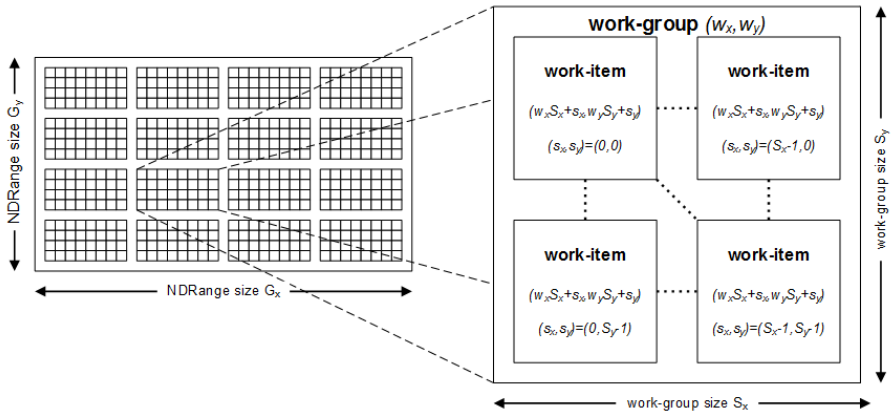


Figure 7 The structure of NDRange

2.2.3 OpenCL Programing

The hardware platform of this project is the Hewlett Packard Enterprise's BladeSystem Blade Servers equipped with a Nallatech 385A FPGA Accelerator Card. And the software environment is Intel FPGA SDK for OpenCL which is installed in Red Hat Enterprise Linux Server release 6.9 (Santiago).

The specification of hardware and software are included in the Appendix A. Besides, the development environment is already established so the development will be the emphasis in this project.

a. Host Design

From the Host side, the main execution process for OpenCL is like this:

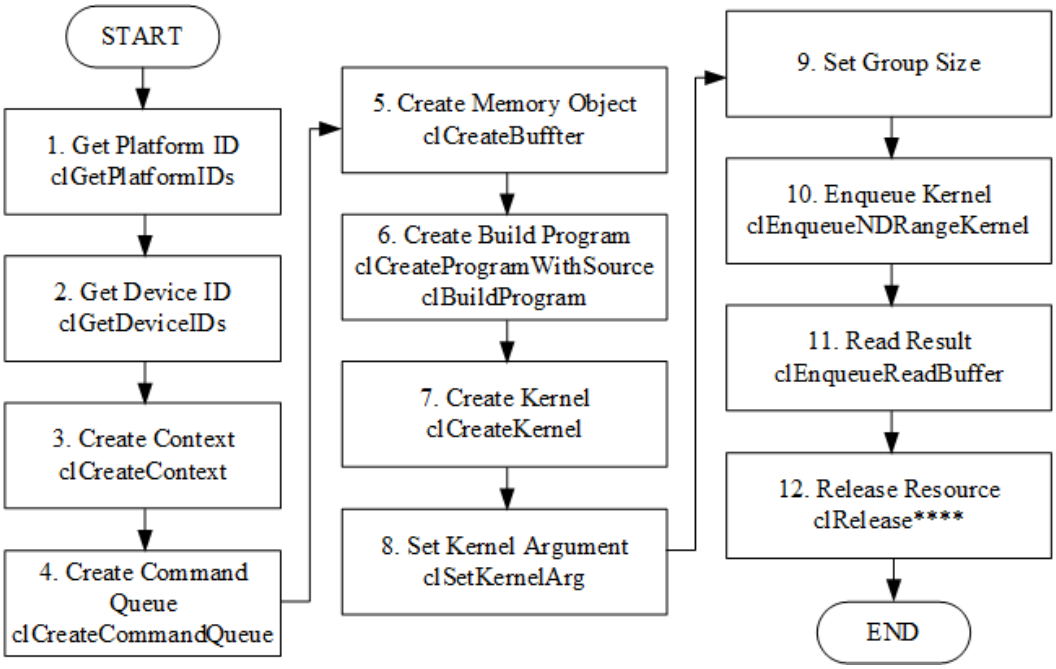


Figure 8 Execution process for OpenCL

Firstly, in particular, Platform is a complete hardware system of Device(s) and Host, the OpenCL management framework which can run the OpenCL program. So the first step is to choose one of the available OpenCL platforms. One machine can have more than one such platform, and a platform can have more than one device. After creating the OpenCL context selection platform, to use the selection operation of the equipment. Also, to create a command queue, the command queue in the definition of the equipment to complete the operation, and the operation order of each operation.

Secondly, the program object is used to store the compiled executable code of the device associated with the context, and also to load and compile the kernel source code. Besides, to execute the compiled kernel operations in the program object, it needs to create the kernel in memory and assign parameters to the kernel function, define the memory object on the device, and allocate the storage space. After creating the kernel and memory object, to set the kernel data, and the kernel queue will be implemented.

Finally, once the kernel has been completed, it needs to copy the data from the device to the CPU for further processing by the host, and all the OpenCL resources need to be released after the read and/or write work.

Basically, in most of the programs, the initialization process of OpenCL in Host are similar, but depending on the program, the process may vary in order. Usually, the host is used to perform a global control on the whole program, while the kernels are used to handle the intensive computing part of the program.

When using Intel FPGA SDK for OpenCL, the API (Application Programing Interface) has been supplied, thus it is much easier to set the initialization for OpenCL in Host. Plus, because sometimes the Host may also take part in computing, only initialization part of the host will be shown in this section.

The pseudo code below is the common initialization of OpenCL, where the details can be found in *Intel® FPGA SDK for OpenCL Programing Guide.*

```
int main(int argc, char **argv) {

/**Step 1: Getting platforms and choose an available one or more.*/
  platform = findPlatform("Intel(R) FPGA");
  if(platform == NULL) {
    printf("ERROR: Unable to find Intel(R) FPGA OpenCL platform.\n");
    return false;
  }

/**Step 2: Query the platform and choose the first device if has one.*/
  cl_device_id devices;
  cl_uint num_devices;
  devices.reset([Parameters]);
  // Just use the first device.
  device = devices[0];
  // Display some device information.
  display_device_info(device);

/**Step 3: Create context.*/
  context = clCreateContext([Parameters]);
  checkError(status, "Failed to create context");
```

```
/**Step 4: Creating command queue associate with the context.*/
  queue = clCreateCommandQueue([Parameters]);

/**Step 5: Create program object */
  std::string binary_file = getBoardBinaryFile([Parameters]);
  printf("Using AOCX: %s\n", binary_file.c_str());
  program = createProgramFromBinary([Parameters]);

/**Step 6: Build program. */
  status = clBuildProgram([Parameters]);
  checkError(status, "Failed to build program");

/**Step 7: Initial input,output for the host and create memory objects for the kernel*/
  cl_mem [Buffername] = clCreateBuffer([Parameters]);
  checkError(status, "Failed to create buffer [Buffer name]");
  //Num of Buffter may be many, depended on the application
  cl_event [event name];
  status = clEnqueueWriteBuffer([Parameters]);
  checkError(status, "Failed to operate [Buffername]");
  //Num of Event may be many, depended on the buffers

/**Step 8: Create kernel object */
  const char *kernel_name = "[Kernel name]";  // Kernel name, as defined in the CL file
  kernel = clCreateKernel([Parameters]);
  checkError(status, "Failed to create kernel");

/**Step 9: Sets Kernel arguments.*/
  status = clSetKernelArg([Parameters]);
  //Num of Arg may be many, depended on the application

/**Step 10: Running the kernel.*/
  status = clEnqueueNDRangeKernel([Parameters]);
  checkError(status, "Failed to clEnqueueNDRangeKernel");
  clReleaseEvent([Parameters]);
  clWaitForEvents([Parameters]);
/**Step 11: Read the output back to host memory.*/
  status = clEnqueueReadBuffer([Parameters]);
  checkError(status, "Failed to clEnqueueReadBuffer");

/**Step 12: Clean the resources.*/
  status = clReleaseKernel([Kernel name]);//*Release kernel.
  status = clReleaseProgram(Program);    //Release the program object.
  status = clReleaseMemObject([Buffer name])Release mem object.
  status = clReleaseCommandQueue([Queue name]);//Release  Command queue.
  status = clReleaseContext(context);//Release context.

  return 0;
}
```

The pseudo code above only contains the OpenCL device setting and I/O buffers setting. In practice, many applications which are more complex will contain more functions. This one is just used as an example of how to use the OpenCL by supplied C++ API.

b. Kernel Design

From the Kernel side, use a simple example, vector addition kernel to show how a simple kernel looks like.

```
// AOCL Kernel for adding two input vectors
__kernel void vector_add(__global const float *x,
                         __global const float *y,
                         __global float *restrict z)
{
   // get index of the work-items
   int index = get_global_id(0);

   // add the vector elements
   z[index] = x[index] + y[index];
}
```

The code of example kernel is shown above, where "__global" is a qualifier, indicating that use external storage (such as DDR or HDD) to store, the grammar is the same as the standard C (C99). The most important part of the code is "get_global_id", which is commonly used in multi-work-item mode, through the id to determine the work-items and then operate, and all the items are the same. Thus the function inside the "vector_add" does not contain "for()" function.

The kernel can also be customized, including setting compute unit, SIMD mode, etc., by this way to control the parallelism of the program. However, sometimes the greater parallel performance consumes more resources. Therefore, to get a balance between serial and parallel performance must be considered by the program designers.

Moreover, to explain by the memory layer of the kernel: the data is created and transferred by "clCreateBuffer" and "clSetKernelArg" in the global memory and constant memory. The register variables of kernel function are in the private memory. The internal variables and caches are in Local memory. It can be seen from the memory layer (illustrated by Figure 5 in 2.2.2) that Device does not directly access global memory, but through the Cache to access. It can be imagined that when the work-items running at the same time, the use of memory in the same cache, then the memory throughput will be at the highest efficiency. Corresponding to the Work-group, that is, use the work-itemss in one Work-group to operate one piece of memory to improve the memory access efficiency.

2.2.4 Prospect of OpenCL on FPGA

When using OpenCL to develop FPGA applications, the designer's main job is to describe the hardware in accordance with each cycle, for the implementation of its algorithm. The traditional process involves the establishment of the data path, designing the state machine to control the data path, using system level tools (e.g. SOPCBuilder, PlatformStudio) to connect to the bottom of the IP kernel, the external interface must satisfy the constraints. As a result, it needs to handle the timing convergence problem.

In addition, the purpose of the OpenCL compiler is to help designers automatically accomplish all of these steps so that they can concentrate on defining algorithms rather than focusing on tedious hardware design. By designing in this way, the designer is very easy to transplant applications to the new FPGA with better performance, this is because the OpenCL compiler will convert to the same level description line to take the advantage of the new FPGA devices.

Therefore, using the OpenCL standard on FPGA, compared with the current hardware architectures (CPU, GPU, etc.), the performance can be greatly improved while the power consumption is reduced. What is more, compared with traditional FPGA hardware description languages (HDL) such as Verilog or VHDL, the use of OpenCL standard and FPGA-based hybrid system (CPU+FPGA) has obvious advantages on the time consumption of product coming into the market.

2.3Speedup Model

2.3.1 Amdahl's Law and its evolution

G. M. Amdahl proposed Amdahl's law in 1967, and gave a model for parallel processing of scalability, pointing out that the speed of using parallel processing is determined by the parallel part of the problem [8], formulated below:

$$speedup_{Amdahl} = \frac{T_{before\ improved}}{T_{after\ improved}} = \frac{1}{(1-p) + {}^{p}/_{s}}$$

where:

- $speedup_{Amdahl}$ is the speedup of the whole task
- s is the speedup of the accelerated part of the task
- p is the proportion of task which can be accelerated

Amdahl's law shows that increasing the number of processors does not significantly accelerate the problem-solving time when the problem is not parallel, shown below:

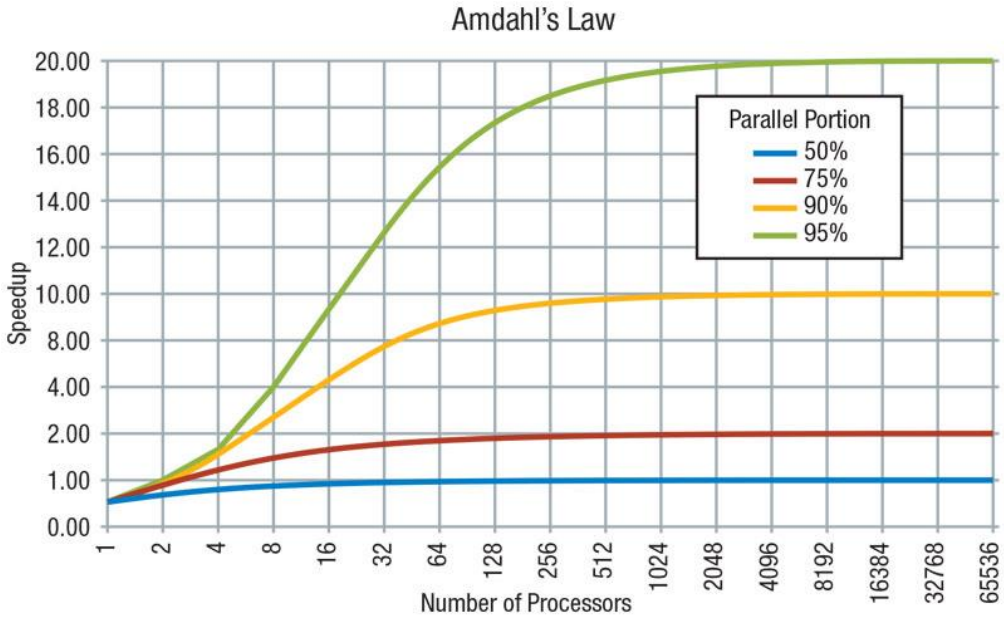


Figure 9 The illustration of Amdahl's Law

But Amdahl's law ignores some important facts. Amdahl's law is a fixed-size model, that is, the size of the problem to be solved is fixed so the ratio of parallelism is fixed. However, in practice, to deal with a small problem will not need 1000 processors. When the computing ability increases, we can and should solve bigger problems. When the problem is bigger, this problem is more likely to be divided into small problems what can be parallelized (or multiple independent problems can be solved), which means a greater p (close to 1) and getting more speedup.

Until 1988, Gustafson proposed a fixed-time model, or Gustafson's Law, which indicates the future in massively parallel computing [9]. This is a model addressing the shortcoming of Amdahl's law. It can be expressed as the following formula:

$$speedup_{fixed-time} = (1 - p) + sp$$

In this model, the total speedup grows by s, which means more cores make more speedup. Thus, the scale of the problem can be expanded (scaled). From this formula it is obviously when p is fixed, speedup grows linearly by s.

In 1990, Sun and Ni proposed a memory-bounded model that is Sun and Ni's law [10], formed as follows:

$$speedup_{Sun-Ni} = \frac{(1 - p) + pG(s)}{(1 - p) + pG(s)/s}$$

In this model, improved part of task grows in the pattern of memory G(s). Speedup also grows linearly with the part of the improved task and is more optimistic than the Gustafson's Law.

In fact, these models are actually consistent in nature, and there is no conflict between them, the difference of speedups is due to the different utilization of the processor. When the problem is scaled instead of fixed, the added processors will have something to do to maintain a relatively high utilization ratio. That is, these computing capacity are not wasted. If not wasted, the computing capacity will do more work in a certain amount of time, and the time to solve the problem has been shortened. As a result, speedup becomes bigger.

Thus, the conclusion is that using more processors are correct, but there is a need for good scalability system support to improve processor utilization.

2.3.2 Use of Speedup Model

In this project, the Amdahl's Law and its model are used to theoretically calculate the speedup of the applications. By using the Amdahl's law, the performance gain can be calculated by improving a part of the computer.

Amdahl's law defines the speedup by obtained using a particular function. The speedup shows how much faster a task can be done on an upgraded computer compared to the original computer.

The basic formula $speedup_{Amdahl} = \frac{T_{before\ improved}}{T_{after\ improved}} = \frac{1}{(1-p)+{}^{p}/_{s}}$ is used to define the total speedup of acceleration.

The speedup depends on the following two factors:

- The proportion of the upgraded portion of the original computation time

For example, the total execution time of a program is 60 seconds, and if the execution time of 20 seconds can be upgraded, the ratio is 20/60. Then this value is the acceleration proportion p, which is always less than or equal to 1.

- The improvement by upgrading the execution mode

That is, how much faster the task will run when it is used by the entire program. This value is equal to the execution time of the original mode divided by the execution time of the upgrade mode. If the upgrade mode for some part of the program takes 2 seconds, while in the original mode it takes 5 seconds, and the boost ratio is 5/2. This is the acceleration speedup s, which is always greater than 1.

In order to gain a better understanding of speedup model. Here is an example: Assuming to upgrade a processor that provides Web services. The new processor executes Web services which is 10 times faster than the original processor. Suppose the original processor has 40% of the time to be busy with computation, and 60% of the time is waiting for I/O. Then the total speedup after this upgrade should be

$$speedup_{total} = \frac{1}{(1-0.4)+{}^{0.4}/_{10}} = \frac{1}{0.64} \approx 1.56$$

2.4Brute-force String Search

The string searching algorithm is a searching algorithm designed to find out whether a string contains target pattern string. There are many string searching algorithms. The Brute-force search may not be the most efficient one but must be the simplest one. Thus it is chosen to be used to implement the string searching of this project.

So called “Brute-force Algorithm" is the most frequently used method in algorithm analysis. This is a method, also an algorithm thinking, that is, without considering the space and time complexity, the simplest solution of the problem.

String Searching problem can also be considered the simplest string algorithm. It's the way most people think about such a matching problem, assuming that there are text string $T[1 \ldots n]$ and pattern string $P[1 \ldots m]$, where $m \leq n$. In general, consider the alphabets of the text and the pattern are same. If an integer s exists, $0 \leq s \leq N - M$ and $T[s + 1 \ldots s + m] = P[1 \ldots m]$, then say that the pattern P shifts in the text T by s. The string problem is to find all the legal shifts.

Because this algorithm is just used to develop a “Hello World” program for learning string operation in OpenCL, this algorithm is described by pseudo code, shown below:

```
BruteForceStringSearch(T[1…n], P[1…m]):
for s  = 1 to n – m + 1
      equal = true
      i = 1
      while equal and i <= m
            if T[s + i - 1] <> P[i]
                  equal = false
            else
                  i = i +1
      if equal = true
            return s
return 0
```

The idea is to compare the initial letters of the pattern with the text. If they are same, compare the next letter of the pattern. If the result has not been found in the N-M loops, it indicates a mismatch and ends the loop. The N-M loops here is because M is the length of the pattern, and N is the length of the text thus if the number of loops exceeds N-M the remaining string length must be less than M, which definitely does not match [11].

2.5Local DNA Sequence Alignment

2.5.1 DNA Sequence

The DNA sequence (or gene sequence) is a primary structure of a DNA molecule that carries real or hypothetical genetic information with a string of letters. A primary structure of a DNA molecule expressing a true or hypothetical genetic information with a string of letters in part of a DNA sequence [12].

The DNA sequences may have letters only A, C, G, and T, representing the four nucleotides that make up DNA - adenine, cytosine, guanine, and thymine. Each letter represents a base, and two bases form a base pair. The pairing of the base pairs is fixed, that is, A-T, C-G. Typically, they are arranged together without intervals, such as sequences of AAAGTCTGAC. A string of nucleotides of any length greater than 4 is called a sequence. As for its biological function, it depends on the context sequence, a sequence that may be read and read backward, including encoding or no encoding. The DNA sequence may also contain "junk DNA".

It is important to note that in this project, the DNA sequence samples are all generated randomly by Python, the code can be found in the ZIP file.

There are two reasons explaining why to use simulated data. On one hand, the scale of these sequences can be easily changed, which is able to bring convenience to the tests of the program. On the other hand, simulated information is free to obtain without any legal problems.

Of course, technically the simulated data cannot clearly show the natural patterns in real DNA sequences. However, this project focuses more on the acceleration of computing rather than the accuracy of the algorithm. Besides, the parameters of this sequence alignment application can be tuned by the biology professional users manually to improve the accuracy of this alignment algorithm. Therefore, the simulated data is totally enough in this project.

2.5.2 Sequence Alignment

Sequence alignment is the basic component and important basis of bioinformatics. The basic idea of sequence alignment, biological sequence determines structure based on the common law, structure determines function, the sequence of nucleic acid sequence and protein structure as a string consisting of the basic character of similarity detection between sequences, find the function, structure and evolutionary information in biological sequences [13].

In biological information processing, finding some similarity relation between two sequences is the pairwise sequence alignment. The character differences between the two sequences are usually used to determine the similarity between sequences. If the corresponding position of the two sequences of characters if the difference is large, then the sequence similarity is low, conversely, sequence similarity is high. There are two methods to perform, one is a global alignment that looks at the entire sequences, and another is a local alignment that only checks the similar regions of two sequences. The diagram showing the relationship between these two methods is shown below.

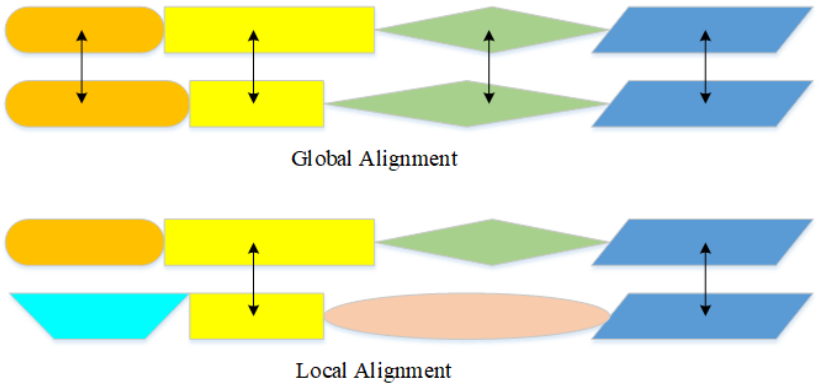


Figure 10 Two methods of alignment

The analysis of sequences of biological macromolecules is a very basic task. From sequence fragment determination, splicing, gene expression analysis, structural and functional prediction of RNA and protein, phylogenetic tree construction of species requires a comparison of molecular sequence similarity. In the evolution process of the genetic material in the long term, originally the same DNA sequence because one sequence was missing several pieces, or adding a few pieces, changed with the change of position or a subsequence, so they have differences and the two sequences do not necessarily match for accurate, but they have certain similarity. In this case, biologists have proposed a method for evaluating sequence similarity, called the scoring function [14].

2.5.3 Smith-Waterman Algorithm

In 1981, by F. Smith and M. Waterman firstly proposed a local alignment algorithm, using the dynamic programing to identify matching sub-sequences with fewer changes and to ignore mismatches and vacancies before or after the matching area [15]. In local alignment, the location of the table where less than zero is replaced by zero. This algorithm is used to examine specific fragments of two sequences. In this project, the local alignment between two DNA sequences is discussed.

a. Formula

Smith-Waterman Algorithm first calculates all possible similarity comparison values of two sequences by an iterative method and then searches for optimal similarity comparison by dynamic programing method.

The specific description of the Smith Waterman algorithm is as follows:

For the two sequences A and B with elements $\mathrm{A}[i]$ and $\mathrm{B}[j]$, where $0 < \mathrm{i} < |A|, 0 < \mathrm{j} < |B|$, belong to a set of characters Ω. For any element and null in Ω, there is a score between each pair of them, represented by a scoring function $\sigma(x, y)$, then use $\mathrm{H}(i, j)$ to indicate the score of the optimal similarity between the prefixes $\mathrm{A}[1]\mathrm{A}[2] \ldots \mathrm{A}[i-1]A[i]$ of sequence A and the prefixes $\mathrm{B}[1]\mathrm{B}[2] \ldots \mathrm{B}[j-1]B[j]$ of sequence B. Then the formula is:

$$\mathrm{H(i,j)} = \max \begin{cases} 0 \ (i = 0 \ or \ j = 0) \\ \mathrm{H(i-1,j-1)} + \sigma(A[i], B[j]) \\ \mathrm{H(i-1,j)} + \sigma(-, j) \\ \mathrm{H(i,j-1)} + \sigma(i, -) \end{cases}$$

In the formula above, the equation $\sigma(A[i], B[j])$ is the match/mismatch score. The equations $\sigma(-, j)$ and $\sigma(i, -)$ are the gap penalties. A "gap" is added to compensate for insertions or deletions in sequences due to the mutations in the process of gene evolution, but not too many gaps can be used in sequence alignment, otherwise the sequences would become unrecognizable. In this project, the gap opening score and gap extension score are set as constant number to simplify the calculation. Normally the gap opening score is higher than the gap extension score.

From the formula above, a scoring matrix can be derived. Then apply the dynamic programing method to trace back the path. A key point to note here is that the traceback process should start from the highest score and end at zero.

b. Example

In order to show the whole process more intuitively, here assume two DNA sequences:

A: G G T T G A C T A

B: T G T T A C G G

Firstly, the parameters are set below:

$\sigma(A[i], B[j]) = \begin{cases} 5 & match \\ -5 & mismatch \end{cases}$

Both of the gap penalty $\sigma(-, j)\ and\ \sigma(i, -)$ can be set as -3.

Secondly, the scoring matrix can be formed:

		G	G	T	T	G	A	C	T	A
	0	0	0	0	0	0	0	0	0	0
T	0	0	0	5	5	2	0	0	5	2
G	0	5	5	2	2	10	7	4	2	0
T	0	2	2	10	7	7	5	2	9	6
T	0	0	0	7	15	12	9	6	7	4
A	0	0	0	4	12	10	17	14	11	12
C	0	0	0	1	9	7	14	22	18	15
G	0	5	5	2	6	14	11	18	17	14
G	0	5	10	7	4	11	9	15	14	12

Table 1 Scoring Matrix of the Example

Thirdly, find the biggest element (yellow highlight) from the matrix above and traceback from it to the final element (green highlight). Just follow the red arrow above.

Finally, take the highlighted part and the alignment is finished. The alignment result is:

G T T G A C

| | | | | |

G T T -- A C

Then put them back to the original sequences to check:

A: G G T T G A C T A

B: T G T T A C G G

From the result above it is clearly shown that these two sequences have similar parts which are highlighted. In other words, the best-matched sequences of two DNA sequences are derived. Thus, the local alignment is implemented successfully in this example.

3. Methodology

3.1Intel FPGA SDK for OpenCL

The specific user manual of Intel FPGA SDK for OpenCL can be found in *Intel FPGA SDK for OpenCL Getting Started Guide* and *Intel FPGA SDK for OpenCL Programing Guide*. Most of the operations are the same as the standard OpenCL development process, but Intel provides some of its own API in the SDK for developers in order to perform a better development for high-level synthesis on the FPGA platform.

The device used is Nallatech p385a_sch_ax115 and the information of the platform and device is shown below:

```
Querying platform for info:
==========================
CL_PLATFORM_NAME                         = Intel(R) FPGA SDK for OpenCL(TM)
CL_PLATFORM_VENDOR                       = Altera Corporation
CL_PLATFORM_VERSION                      = OpenCL 1.0 Intel(R) FPGA SDK for Open
CL(TM), Version 16.1.2

Querying device for info:
========================
CL_DEVICE_NAME                           = p385a_sch_ax115 : nalla_pcie (aclnall
a_pcie0)
CL_DEVICE_VENDOR                         = Nallatech ltd
CL_DEVICE_VENDOR_ID                      = 4466
CL_DEVICE_VERSION                        = OpenCL 1.0 Intel(R) FPGA SDK for Open
CL(TM), Version 16.1.2
CL_DRIVER_VERSION                        = 16.1
CL_DEVICE_ADDRESS_BITS                   = 64
CL_DEVICE_AVAILABLE                      = true
CL_DEVICE_ENDIAN_LITTLE                  = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE          = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE      = 0
CL_DEVICE_GLOBAL_MEM_SIZE                = 8589934592
CL_DEVICE_IMAGE_SUPPORT                  = true
CL_DEVICE_LOCAL_MEM_SIZE                 = 16384
CL_DEVICE_MAX_CLOCK_FREQUENCY            = 1000
CL_DEVICE_MAX_COMPUTE_UNITS              = 1
CL_DEVICE_MAX_CONSTANT_ARGS              = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE       = 2147483648
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS       = 3
CL_DEVICE_MEM_BASE_ADDR_ALIGN            = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE       = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR    = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT   = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT     = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG    = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT   = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE  = 0
```

Figure 11 Information of the platform and device

3.2Program Design

There are two applications designed in this project, one is String Searching and another is Local DNA Sequence Alignment.

When designing the programs, in order to make it convenient to compare the performances between different implement methods (pure C++ and OpenCL), both of these two applications have two versions, C++ and OpenCL. By this way, the speedup of OpenCL implementation can be derived. The analysis of each application will also consider the theoretical speedup by using speedup model which has been mentioned in 2.3.

Besides, there is another way to design the programs, to combine the C++ version of the application into the OpenCL version. The reason of this way is that the OpenCL use C++ to code the host program, thus technically it is reasonable to put a C++ program into the host. However, this way may let the time analysis very difficult because it cannot give an accurate time report (in g++ the "time" command will report system time and user time).

Moreover, the time consumption is used to be the standard to measure the performance, by using the same machine, the running time of the programs will indicate different performances.

Therefore, the separated two implementations of each application will be designed and debugged while the running time will be used to as a measure of performance. Also, by this method, the complexity of porting a C++ program to the OpenCL platform can be simply defined. The full versions of all of the code will be attached in the ZIP file.

3.2.1 String Search

String search is an important operation in the field of information security and information filtering, especially for the real-time processing of a very large text. Besides, this application was chosen to be a "Hello World" of this project.

Although there are many other "Hello World" programs such as a kernel printing example and a vector addition example supplied by the Intel FPGA, there are still two reasons to take string searching as the start, one is that the string searching uses string

operation and which is also used in local sequence alignment, and another is that the brute-force string search is one of the simplest algorithms in string algorithms. Thus, it is suitable for a start of the development part in this project.

a. C++ Implementation

The implementation of this application in C++ is quite simple. The function of string search can be coded following the pseudo code in 2.4, shown below:

```
int BruteForceStringSearch(string Text, string Pattern)
{
   int lenT = Text.length();
   int lenP = Pattern.length();

   int s,i;
   for (s = 0; s <= lenT-lenP; s++)
   {
      i = 0;
      bool bEqual = true;
      while (bEqual && (i < lenP))
      {
         if (Text[s+i] == Pattern[i])
         {
            i++;
         }
         else
         {
            bEqual = false;
         }
      }

      if (bEqual)
      {
         return s;
      }
   }

   return 0;
}
```

By using the function above, once a matched pattern was found the function needs to be re-run after the end of the current pattern. Thus, there is a “while” loop in the main function to make sure that all of the text has been searched. What is more, in the main function, by including a header file “fstream.h” it is able to read the string from a “.txt” file, which can make it convenient to set the input of this program. The text file is also a randomly generated string. In this program, the generated random DNA sequences can be used directly.

b. OpenCL Implementation

In OpenCL implementation, it will be more difficult. Firstly, to design a kernel to implement the string search function in order to accelerate this algorithm.

The idea of the basic design is shown below:

- Store a small amount of constant data, such as pattern string length, text length, and so on, in the private memory.
- The pattern string is preserved in the local memory of FPGA to accelerate the access of threads to the pattern string.
- Save the text in the global memory, and use the work-items as many as possible to access the global memory to reduce the average memory access time.
- Work-items in each Work-group manipulate a section of text, and multiple work-group handles the whole large text in parallel.

It should also be concerned that the synchronization is necessary for these Work-groups to ensure each part of the text can meet the end at the same time before next operation. Thus the "barrier" function should be added into the kernel, here the memory fence is used.

The Kernel is shown below:

```
int compare(__global const uchar* text, __local const uchar* pattern, uint length){
   for(uint l=0; l<length; ++l){
      if (text[l] != pattern[l])
      return 0;
   }
   return 1;
}

__kernel void StringSearch (
     __global uchar* text,              //Input Text
     const uint textLength,             //Length of the text
     __global const uchar* pattern,     //Pattern string
     const uint patternLength,          //Pattern length
     const uint maxSearchLength,        //Maximum search positions for each work-group
     __global int* resultCount,         //Result counts (global)
     __global int* resultBuffer,        //Save the match result
     __local uchar* localPattern)}      //local buffer for the search pattern

   int localIdx = get_local_id(0);
   int localSize = get_local_size(0);
   int groupIdx = get_group_id(0);

   uint lastSearchIdx = textLength - patternLength + 1;
   uint beginSearchIdx = groupIdx * maxSearchLength;
   uint endSearchIdx = beginSearchIdx + maxSearchLength;
   if(beginSearchIdx > lastSearchIdx)
      return;
   if(endSearchIdx > lastSearchIdx)
```

```
        endSearchIdx = lastSearchIdx;

    #pragma unroll 2
    for(int idx = localIdx; idx < patternLength; idx+=localSize)
        localPattern[idx] = pattern[idx];
    barrier(CLK_LOCAL_MEM_FENCE);

    #pragma unroll 2
    for(uint stringPos=beginSearchIdx+localIdx; stringPos<endSearchIdx; stringPos+=localSize){
    if (compare(text+stringPos, localPattern, patternLength) == 1){
            int count = atomic_inc(resultCount);
            resultBuffer[count] = stringPos;
        //printf("%d ",stringPos);
        }
    }
}
```

In a kernel, the loop operation in C++ is replaced by work-items in OpenCL, and all of the work-items run at the same time which is purely parallel so the running speed is faster than the loop in C++. Moreover, in the kernel code above, there are two factors, "#pragma unroll 2". This is the operation which can unroll the loop.

The Intel FPGA SDK for OpenCL offline Complier can unroll a loop of the kernel to optimize the data processing efficiency, which is a useful strategy, and there are more can be found in *Intel FPGA SDK for OpenCL Programing Guide*. Besides, all of the kernels must be saved as ".cl" and where must contain only OpenCL code.

In addition, the "atomic_inc()" is one of the atomic operations which is used to ensure each result from the work-times can be written into the right location of global memory. This operation will affect the efficiency of the whole kernel but it is necessary especially for a text with a very large string inside. It should also be concerned that not every OpenCL device support atomic operation and if the device doesn't support it, simply remove this function and the kernel can be proper to use.

After the kernel design, the host is designed to manage the data and run the kernel on the device (FPGA).

The method to initialize the OpenCL platform has been already introduced in 2.2.3. Because the code of host in OpenCL is very long even if it is a very simple program, here use pseudo code to show how to design the host for this application, String Search.

```
#include <Necessary Headers>
using namespace aocl_utils;
int main(int argc, char **argv) {
```

```
//OpenCL init
findPlatform()
getDevices()
clCreateContext()
clCreateCommandQueue()
createProgramFromBinary()
clBuildProgram()

//Input the text and pattern
input text
input pattern

//Set Buffers and Write data
clCreateBuffer(write text)
clCreateBuffer(write pattern)
clCreateBuffer(result)
clEnqueueWriteBuffer(write text)
clEnqueueWriteBuffer(write pattern)

//Create kernel
clCreateKernel()

//Set kernel argument
clSetKernelArg()

//Running the kernel
clEnqueueNDRangeKernel()

//Release write event
clReleaseEvent(write_event)

//Wait for kernel finished
clWaitForEvents()

//Read the output
clEnqueueReadBuffer(result)

//Free the resources
clReleaseKernel(kernel);//*Release kernel.
clReleaseProgram(program);    //Release the program object.
clReleaseMemObject(buffer);//Release mem object.
clReleaseCommandQueue(queue);//Release  Command queue.
clReleaseContext(context);//Release context.
}
```

From the code above it is clearly shown that the host code of OpenCL is very complex even in pseudo code. The most of the host of this application is to setup the OpenCL device and parameters to ensure the kernel can work properly. The memory allocation is the most significant part, in order to prevent the data overflow, each operation of memory allocations must be concerned about.

Besides, according to the whole program, there is only the input of text and pattern and output of the result coded in the host so the whole string searching algorithm is actually

handled in the kernel. As a result, the I/O and OpenCL initialization of the host is serial which may cost some time while the kernel runs in parallel which should have a better efficiency.

c. Comment

To summarize the development of String Search. In the C++ implementation, it is quite simple due to the simple algorithm and familiar development environment. And the result of it should be easily verified. The only problem might be the efficiency of the program.

However, in OpenCL implementation, it is obviously more difficult and time-consuming but the efficiency of the program has been improved. According to the record, it took fifteen days to implement String Search by OpenCL while the development time of its C++ version took only three days. Therefore, the thinking about if it worth to re-code the C++ programs into OpenCL ones seems necessary. And this thinking will be discussed later.

Anyway, the development of String Search is done and the knowledge about OpenCL programing is also partly gained here, the next section is about the Local DNA Sequence Alignment, which is the main research object in this project.

3.2.2 Local DNA Sequence Alignment

The concept of this alignment algorithm has already been discussed in 2.5. This application was chosen to be the main research object in this project because many of the biological algorithms such as DNA sequence alignment usually take days or even weeks when using traditional way to compute.

Therefore, if the FPGA accelerator can improve the performance of such applications, this will not only demonstrate the enormous potential of FPGA-based accelerators but also contribute to biological and other scientific research because it will accelerate their research process to obtain better efficiency.

a. C++ Implementation

To implement Smith-Waterman algorithm in C++ is not very complicated. In the beginning, a flow chart is necessary to design in order to indicate the whole structure of this algorithm, shown below:

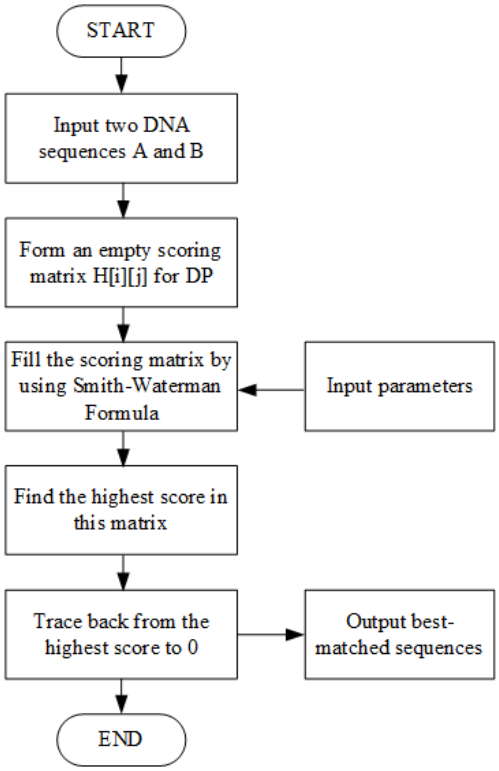


Figure 12 Flowchart of SW algorithm

Here use the pseudo code to show how to design a C++ code for local alignment.

```
//Read simulated DNA sequences from text file
ifstream read A from "test_data0.txt"
ifstream read B from "test_data1.txt"

//Store the sequences in the buffers
bufferA << readA.rdbuf()
bufferB << readB.rdbuf()

//Input the sequences
string A(bufferA.str())
string B(bufferB.str())
printf original A and B

//Input parameters
set parameters: match/mismatch score and gap penalty

//Alignment
score (A, B, parameters)
traceback

//Result
Printf best-matched A and B
```

It is similar to the design of String Search, the data is stored in a text file where the sequences are randomly generated. The reason why use the simulated data has been explained in 2.5.1. So the first thing of this program to do is to read the text file by using "fstream".

Firstly, to discuss the scoring matrix. According to the formula in 2.5.3, assume the length of A and B are m and n, the matrix should be in the form of (m+1) × (n+1) while both of the first row and column are zero. In this design, the matrix H is formatted by "vector<double>" because the size or the number of elements in a vector are allowed to be changed anytime. In this case, by using a vector, there is no need to change the size of arrays when handling sequences in different length. As a result, the vector is more suitable for the dynamic programing than traditional arrays. The matrix H can be initialized by the following code:

```
vector <vector <double> > H;

for(long k = 0; k <= m; k++)
      H[k][0] = 0; //First row

for(long l = 0; l <= n; l++)
      H[0][l] = 0;  //First column
```

Secondly, when scoring the matrix, the value of current element is the maximum value derived by comparing the three values calculated by the elements from left, top left and top. The code below is shown how to implement this function:

```
for(long i = 1; i <= m; i++)
{     for(long j =1; j <= n; j++)
      {
            double c1=0, c2=0, c3=0;

            c1 = H[i-1][j-1] + S(A[i-1],B[j-1]); //top left //S(A, B) returns the score of match/mis
            c2 = H[i-1][j] + gap; //left
            c3 = H[i][j-1] + gap; //top

            H[i][j]=max(c1, c2, c3);
      }
}
```

The code above is the simple C++ implementation of the scoring process using linear gap penalty which has the same scores for opening and extending a gap. This is exactly like the method used in the example of 2.5.3. However, to obtain a better alignment, a more complex function is used to optimize the algorithm, the modified code is shown below:

```
maxc=maxr=0-opengap;
for(long i=1;i<=n;i++)
{   for(long j=1;j<=m;j++)
    {
        double c1=0,c2=0,c3=0;
        c1=H[i-1][j-1]+S(A[i-1],B[j-1]);
        if(maxc-(H[i-1][j]-opengap)>0 )
            maxc=maxc-extgap;
        else
            maxc=H[i-1][j]-opengap-extgap;
        if(maxr-(H[i][j-1]-opengap)>0 )
            maxr=maxr-extgap;
        else
            maxr=H[i][j-1]-opengap-extgap;

        H[i][j]=max(max(c1, maxc),max(maxr, (double)0.0));
    }
}
```

In this way, it considers more on the alignment itself and the result will be more accrete while the computational time will be longer. As a result, the best-matched part of two sequences will be less. Thus, there is a trade-off between performance and accuracy. Concerned the smith-waterman algorithm is not the one which focuses on performance but the accuracy, the optimized scoring method is usually adopted.

The next part and also the final part of this program is the “traceback”, the most significant process. It starts with the highest score in the matrix and end at zero. The code is shown below:

```
//find the value and location of the highest mark
for(i = 0; i <= A.length(); i++)
    for(j = 0; j <= B.length(); j++)
        if(H[i][j] > max)
        {
            maxrow = i;
            maxcol = j;
            max = H[i][j];
         }

//traceback according to the comparison
      i=maxrow;
      j=maxcol;
      while((i > 0) && (j > 0) && H[i][j] > 0)
      {
          if(fabs(H[i][j]-(H[i-1][j-1]+S(A[i-1],B[j-1])))<=1e-6) //current element vs top left
          {
              A1.insert(0, 1, A[i-1]);//A1 and B1 are the best-matched sequences of A and B
              B1.insert(0, 1, B[j-1]);
              i--;j--;
          }
          else
          {
              long  k1, k2;
              k1 = 1; k2 = 1;
```

```
            long flag=0;
            while(i - k1 >= 0 || j-k2 >= 0)
            {
                    if(i-k1>=0&&fabs(H[i][j]-(H[i-k1][j]-W(k1)))<=1e-6) //current vs left
                    {
                            flag=0; //flag indicates A or B need to add a gap "-"
                            break;
                    }
                    else
                            k1++;

                    if(j-k2>=0&&fabs(H[i][j]-(H[i][j-k2]-W(k2)))<=1e-6) ) //current vs top
                    {
                            flag=1;
                            break;
                    }
                    else
                            k2++;
             }

             if(flag==0) //according to flag to insert gap in A1 or B1
             {
                    for(long m = 1; m <= k1; m++)
                    {
                            A1.insert(0, 1, A[i-m]);
                            B1.insert(0, 1, '-');
                     }
                     i=i-k1;
              }
              else
              {
                    for(long m = 1;m <= k2; m++)
                    {
                            A1.insert(0, 1, '-');
                            B1.insert(0, 1, B[j-m]);
                     }
                     j = j-k2;
               }
        }
}
```

The function "W(k)" here is to calculate the gap penalty where the result is opengap+extgap×k and k is the number of the count of traceback steps. Then the result A1 and B1 are finally derived. By simply printing these two sequences out the program is finished. The factors that influence the outcome depend mainly on the parameters setting. Due to the use of simulated DNA sequences, there is no common pattern that natural DNA has, the tuning of parameters for simulation and nature may be different.

The next section is to discuss the OpenCL implementation of Local DNA Sequence Alignment.

b. OpenCL Implementation

In OpenCL implementation, the scoring part is re-coded into the kernel while the traceback part remains in the host. The reason why to form the matrix in the kernel is that it is the most computational intense part in the whole algorithm. The code of this kernel is shown below:

```
enum direction {
  CURRENT, TOP, TOP_LEFT, LEFT//define the location of three elements
};

long match(char ai, char bj) { //the match function
  const long MATCH = 2;
  const long MISS_MATCH = -1;
  if (ai == bj)
    return MATCH;
  else
    return MISS_MATCH;
}

// Smith Waterman OpenCL kernel
__kernel void SW (
  __global long *matrix,//scores
  __global long *memory,//path
  __global long *tmpdim,
  __global long *subIndexes,
  __global long *sub,
  __global char *A,
  __global char *B)
{
  long x = get_global_id(0);
  long dim = *tmpdim;
  long startI = subIndexes[2*x+0];
  long startJ = subIndexes[2*x+1];

  long ii, jj;
  const long GAP = -1;//use linear gap penalty
  for (ii = startI; ii < *sub + startI; ++ii)
  {
    long i = ii + 1;
    for (jj = startJ; jj < *sub + startJ; ++jj)
    {
      long j = jj + 1;

      long max = 0;
      long index = j + I * dim;

      long tl = matrix[(j-1)+(i-1)*dim] + match(A[jj], B[ii]);//socre of top-left
      if (tl > max)
      {
        max = tl;
        memory[index] = TOP_LEFT;
      }

      long t = matrix[j+(i-1)*dim] + GAP;//score of top
      if (t > max)
      {
        max = t;
```

```
      memory[index] = TOP;
    }

    long l = matrix[(j-1)+i*dim] + GAP;//score of left
    if (l > max)
    {
      max = l;
      memory[index] = LEFT;
    }

    if (max == 0)
      memory[index] = CURRENT;

    matrix[index] = max; //the max score
   }
  }
}
```

The value of scores is in the "long" type because the FPGA may not be so efficient on floating point calculation. Plus, there is only global memory used here to prevent the data transmitting across many types of memory that reduce the processing speed. In terms of 2.2.2 b, the data cannot cross over memories. As a result, the frequently used data are all stored in the global memory. What is more, the scoring function is the one that used in C++ implementation which one is not modified in order to simplify the kernel function.

There is another method that makes the whole algorithm into the kernel. It is technically possible and more efficient but it requires many kernels and channels to manage each kernel. Moreover, the complex kernel file will consume a long time to compile and it brings a lot of problems when running the program, especially when processing large-scale DNA sequences, which will cause the synchronization problems. Therefore, in order to make a trade-off between performance and stability, just take the scoring part out separately and make it into a relatively simple kernel.

Then the host is mainly designed to implement the traceback of the Smith-Waterman algorithm. In addition, the scoring matrix is also formed in the host, similar to the one in C++ implementation.

It is simple to design the host with the experience of String Search. By adding OpenCL initialization into the C++ version's code and remove the scoring part. It should be concerned that the form of the matrix needs the extra parameters in order to store the location of each element and the path of the traceback.

Here use the pseudo code to indicate the host design.

```
Init(OpenCL)

Input sequence A, B from text files

Form the scoring matrix // it has more parameters than the one in C++ implementation

Compute_in_kernel(matrix, memory, dim, sub, A, B, device, kernel)

result = traceback(A, B, memory, matrix)

Output result

Free resources
```

The method that initializes the OpenCL, set buffers and arguments and free resources have already been shown in 3.2.1 b, thus, there is no need to show it again. The simple pseudo code above is clearly shown how it works.

This implementation is different from the String Search, it only takes part of the algorithm to be the kernel. The way to define the proportion of the marking in the whole algorithm is to measure the running time of each part in the C++ version of this program. The measurement will be discussed in the test of this application.

c. Comment

The development of Local DNA Sequence Alignment was not as complicated as imagined thought it took a long time to research the algorithm and parameter tuning. The most difficult part is in the OpenCL version design when choosing if it needs to put the whole algorithm into the kernel.

The first try of implementing the whole algorithm into the kernel (just like how it works in String Search) was failed due to the wrong use of channels. When using channels to manage each kernel, there are many problems about memory allocation and data transmission. As a result, the plan B, to separate the algorithm was selected. With the experience of developing the kernel for String Search, this plan finally succeeded.

It took two weeks to implement the C++ version fully while around one and half a months were used to test different possible implementation and then got a fully functional OpenCL version Local DNA Sequence Alignment application.

3.3Experiment Design

3.3.1 Test for String Search

The randomly generated DNA sequences are used to test this application. In this test, the text is the sequence and the pattern is set as “ATCG”. In other words, to find the string pattern “ATCG” in a DNA sequence. Two DNA sequences are used as the experiment objects, one with the length of 6.4M (6400000) and another with the length of 64M (64000000).

By using two sequences with different length, it is clearer to define the speedup difference between the OpenCL or C++ version of String Search. In order to avoid the system error, each version should be run five times and the average running time will be used as the standard of performance. Thus, if the time consumption is smaller the performance of the program is higher.

3.3.2 Test for Local DNA Sequence Alignment

Similarly, in order to test this application, five pairs of DNA sequences with the length of 1k (1000), 5k, 10k, 50k and 100k are randomly generated. Each pair contains sequence A and B. Also, the pairs of A and B can be with different length. But in order to perform a clear test, the all of the pairs are in the same size, that is, the length of sequence A equals to the length of B.

The sequences are shorter than those used in String Search test because the complexity of Smith-Waterman Algorithm is O(mn), thus, the size of the problem will be 1000000 when A and B are in length of 1k. The size will be very large so the 100k is the longest sequence length used in Local DNA Sequence Alignment test. The alignment of each pair must be run five times to avoid system error. There are five pairs of DNA sequences that can clearly show the variation trend of the speedup.

What is more, because the algorithm is separated into two parts and one of them is accelerated while the other is not, the proportion of the accelerated part must be defined. The method to define the proportion of scoring matrix is that measure the time consumption of scoring and traceback in the C++ version where the program is pure in serial processing. Then the proportion can be derived. By using Amdahl’s law the further analysis can be done.

4. Results and Discussion

4.1String Search

Because the time consumption is used to measure the performance of the program, the table which recorded the running time of each implementation will indicate the speedup.

The table below shows the processing time of both C++ version and OpenCL version of this application, with the speedup ratio derived from the time data.

Sequence Size	Version	No. of Test					Average Time	Time Unit	Speedup Ratio
		1	2	3	4	5			
6.4M	OpenCL	42	40	42	42	42	41.6	ms	5.5x
	C++	235	214	218	222	228	223.4	ms	
64M	OpenCL	1322	1314	1305	1307	1304	1310.4	ms	1.6x
	C++	2229	2137	2037	2033	2112	2109.6	ms	

Table 2 Test results of String Search

From the table above it is clearly shown the speedup of the OpenCL version compared with C++ version. The results indicate that use the FPGA accelerator can gain the positive speedup (5.5x for 6.4M).

However, when processing a lager file (64M), the speedup ratio is much lower (1.6x). Theoretically, the complexity of the Brute-force string search algorithm is O(m+n) (where m is a very big number so it can be considered as O(m)), which means when increasing the length by 10x the time consumption will be increased by 10x. This is shown in C++ version, that it takes 223.4ms to handle the 6.4M sequence while 2109.6ms was taken to handle the 64M sequence. But in OpenCL implementation, the time increased by around 30x.

The kernel compilation report is auto-generated by the SDK and there are lots of information about the kernel. There is a full report in ".html" shown the details about the kernel. Here just shown the resource analysis report while the full report will be attached in the ZIP file. After compiling the kernel of String Search OpenCL, the resource consumption report is shown below:

```
ALUTs: 58060
Registers: 96,844
Logic utilization: 63,735 / 427,200 ( 15 % )
```

```
I/O pins: 335 / 826 ( 41 % )
DSP blocks: 2 / 1,518 ( < 1 % )
Memory bits: 3,823,496 / 55,562,240 ( 7 % )
RAM blocks: 377 / 2,713 ( 14 % )
Actual clock freq: 260.317459666
Kernel fmax: 260.48
1x clock fmax: 260.48
2x clock fmax: 10000
Highest non-global fanout: 15012
```

The report above shows the utilization of the resources on the FPGA accelerator card. When designing the kernel, the usage of each resource must be concerned about, especially for a complex kernel. It can be observed from the resource report that the clock speed of the device is only 260 MHz, this is not a very high frequency compared with CPU, but it still accelerated the program. Thus, there is a selection between parallel processing in a low clock frequency and serial processing in a high clock frequency.

By analysing the implementation of OpenCL, the only serial part is the I/O. As a result, the I/O time takes the most part of time consumption in this program. It is obviously the speedup is limited by the data transmission time. This is because the string search algorithm is too simple that the speedup will not be phenomenal enough when being parallelized in a low processing frequency compared with a high-speed serial processing. Predictably, when the length of the object is very large, the running time of the OpenCL version and C++ version will be similar but anyway, the one implemented by OpenCL will be slightly faster.

To conclude this section, the bottleneck of this FPGA accelerator probably is the limitation of data transmission. The data is transferred by PCIe connector, thus there must be a limitation. Therefore, when handling the simple algorithm with large data such as string search, the speedup of the algorithm itself does not play a decisive role, the emphasis should be put on the data transmission speed. On the contrary, if there is an application that with the complex algorithm and relatively small amounts of the data transmission, which should be more suitable for this FPGA acceleration. The Local DNA Sequence Alignment seems to be the proper one, which will be discussed later.

4.2Local DNA Sequence Alignment

Firstly, the test results of C++ implementation is shown in the table below:

Sequence Size		1k	5k	10k	50k	100k	Unit
Running Time	I/O	6	246	778	13340	63630	ms
	Marking	132	1432	4144	103060	412350	ms
	Traceback	12	158	638	15880	53570	ms
	Total	150	1836	5560	132280	529550	ms
Proportion of Marking		91.7	90.1	86.7	86.7	86.7	%

Table 3 Test results of Local DNA Sequence Alignment in C++

The data of time measurement are all the average value from the results from running the program for each size of experiment objects by five times. From the results, the proportion of marking part of Smith-Waterman Algorithm is around 87%. This value can be used later to analysis the speedup by the kernel on the FPGA device.

Secondly, the results of OpenCL implementation is shown below:

Sequence Size	1k	5k	10k	50k	100k	Unit
Total Running Time	28	525	1784	39231	158188	ms
Total Speedup Ratio	5.4x	3.5x	3.1x	3.4x	3.3x	--

Table 4 Test results of Local DNA Sequence Alignment in OpenCL

The alignment of each pair with different size of sequences is run by 5 times and the value of time above is the average value. Despite the outlier 5.4x, the total speedup of OpenCL program is around 3.4x. It can be derived reasonably that the I/O time does not take the main part of time consumption because the complexity of the algorithm is O(mn) while it of the I/O is O(m+n), where m and n can be a large number thus the effect from data transmission speed is much smaller in this application compared to the String Search application.

Then use Amdahl's law, $speedup_{Amdahl} = \frac{T_{before\ improved}}{T_{after\ improved}} = \frac{1}{(1-p)+{}^{p}/_{s}}$ where known the accelerated part p is 87% and total speedup is 3.4. Re-write the equation into $s = p/\left(\frac{1}{speedup_{total}} - (1-p)\right) = 0.87/\left(\frac{1}{3.3} - (1-0.87)\right) = 5.03$. So the speedup of accelerated part (marking) is around 5x, which is a roughly calculation.

The full analysis generated automatically will be attached in ZIP file and the resource consumption report is shown below:

ALUTs: 49892
Registers: 97,427
Logic utilization: 58,906 / 427,200 (14 %)
I/O pins: 335 / 826 (41 %)
DSP blocks: 4 / 1,518 (< 1 %)
Memory bits: 5,322,248 / 55,562,240 (10 %)
RAM blocks: 470 / 2,713 (17 %)
Actual clock freq: 341.176469736
Kernel fmax: 341.29
1x clock fmax: 341.29
2x clock fmax: 10000
Highest non-global fanout: 15012

Compared with the one of String Search Kernel, this one has a higher kernel clock speed, 341 MHz, it also gets a higher speedup ratio. Because the whole device is basically a black box, the detailed research in the low-level implementation seems not valuable in terms of this SDK is designed for high-level synthesis.

However, it is hard to say if the developers of OpenCL on FPGA need the experience or knowledge about HDL such as Verilog or VHDL. On one hand, the SDK is designed for someone with C++ or OpenCL development experience, it should not require a knowledge about low-level synthesis. On the other hand, because the device itself is FPGA-based, there is an existed demand for the understanding of the FPGA and related information.

Besides, the current experiments do not make the device full loaded. The only limitation that affects the performance should be the resources on the board, it can be predicted that if the logic elements on the FPGA is full, the synthesis has to separate the data to process and which will consume more time. In addition, due to the use of simulated data, the speedup still has much room to increase.

Finally, to conclude this section, the algorithm like Smith-Waterman Algorithm is very suitable to be accelerated by FPGA-based accelerators because it does not require much on data transmission speed but the demand for the speedup on the algorithm itself.

5. Conclusion

Firstly, it depends on the application itself to define if it worth to re-code the program into OpenCL implementation. The re-code process needs more time than developing a program in C/C++ and it takes more if developing the program directly by OpenCL. The OpenCL on FPGA is more suitable for some applications that will be run for a long time. If it is one application that needs to be used for only several times, it may not worth.

Secondly, there will be some bottlenecks when using FPGA-based accelerators. In other words, not every application is suitable for accelerating. Moreover, if the algorithm is too simple that the speedup of it may not so phenomenal, the total speedup will be limited by the data transmission speed. In this project, the String Search application is a good example that shows the data transmission speed lower the total speedup. Conversely, the Smith-Waterman Algorithm gained a good speedup ratio, that is, for an algorithm which has parallel part and which is complex enough that CPU cannot handle it efficiently, the FPGA-based accelerator will be a better choice.

Finally, the developers of OpenCL on FPGA need some basic knowledge about FPGA. It is so important to understand how an FPGA works when developing the host and the kernel because there are many embedded resources can be utilized. Although most of the resources have been set up by the SDK, the basic understanding of an FPGA board is necessary. Besides, there is no need to learn the HDL because the SDK is run on a high-level. This is a good news for many scientific researcher or software programmer who are too busy to learn another programing language or do not want to study an unfamiliar low-level language. The C++ will be enough for OpenCL on FPGA.

By now, this project successfully evaluated the performance of the FPGA-based accelerator and the development on related SDK. The main aim and objectives of this project have been achieved.

6. Future Work

In this project, the DNA sequences used are all randomly generated, the performance of the Local DNA Sequence Alignment may change if using real DNA sequences as the object. The related parameters are all set as adjustable for who may use it to align the natural DNA data.

Moreover, there are many FPGA-based accelerators on the market, the one used in this project is the Nallatech 385A FPGA Accelerator Card. The host interface of it is PCIe Gen3 x8. There are many other types of high-speed interface such as Thunderbolt2 or PCIe x16, if there are accelerators with those connectors, it is worth to test the applications developed in this project to research the difference of the performance on those platforms.

Besides, there are some SoC development boards such as Terasic DE1-SoC Board which contains an ARM core embedded in an FPGA chip. It will be a good trail if port the programs to there to evaluate how those SoC boards run these applications. Of course, it can be predicted that the performance will not be satisfactory due to the low performance of the system. However, it is instructive to learn about heterogeneous computing on those boards, which are much cheaper than the professional FPGA accelerator card.

Reference

[1] Steve Wilkus, “A Critique of Pure Speed,” *5G News*, [Online]. Available: http://5gnews.org/critique-pure-speed/ [Accessed 16/03/2017]

[2] SimTech Exzellenz-Cluster, "Simulation on Reconfigurable Heterogeneous Computer Architectures,"*SimTech*,[Online].Available: https://www.iti.unistuttgart.de/abteilungen/rechnerarc-hitektur/projekte/heterogeneous-computing.html/ [Accessed 15/08/2017]

[3] Aaftab Munshi, “OpenCL Programming Guide,” *Pearson Education*, page 3, Jul. 2011.

[4] Terasic Technologies, "OpenCL for Terasic Board," *Terasic Technologies*, [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=11&No=927/ [Accessed 15/08/2017]

[5] Jonathan Tompson and Kristofer Schlachter, "An Introduction to the OpenCL Programming Model," *NYU: Media Research Lab*, [Accessed 20/08/2017]

[6] Vladimir Starostenkov, "Hands on OpenCL," *SlideShare*, Oct. 2014

[7] Doug Watt, "A Quick Guide to Writing OpenCL Kernels for PowerVR Rogue GPUs," *Imagination Technologies*, [Online]. Available: https://www.embedded-vision.com/industry-analysis/technical-articles/quick-guide-writing-opencl-kernels-powervr-rogue-gpus/ [Accessed 20/08/2017]

[8] Gene M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *AFIPS spring joint computer conference*, 1967

[9] J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, Volume 31, May 1988

[10] Xian-He Sun and Lionel M. Ni, "ANOTHER VEIW ON PARALLEL SPEEDUP," *Department of Computer Science, Michigan State University, East Lansing, MI* 48824-1027

[11] Levitin, "Brute Force Sorting and String Matching," *Levitin*, 2007

[12] Wikipedia, “DNA sequencing,” *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/DNA_sequencing [Accessed 20/08/2017]

[13] Polyanovsky, V. O.; Roytberg, M. A.; Tumanyan, V. G., "Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences," Algorithms for Molecular Biology. 6 (1): 25. PMC 3223492 Freely accessible. PMID 22032267 doi:10.1186/1748-7188-6-25

[14] Didier Gonze, "Macromolecular Sequence Analysis," *Université Libre de Bruxelles*, 18 Sep. 2014

[15] Smith, Temple F. & Waterman, Michael S., "Identification of Common Molecular Subsequences," Journal of Molecular Biology, 147: 195–197. PMID 7265238. doi:10.1016/0022-2836(81)90087-5

Bibliography

1. Intel FPGA, “Intel® FPGA SDK for OpenCL Getting Started Guide,”*Intel*, UG-OCL001 2017.05.08

2. Intel FPGA, “Intel® FPGA SDK for OpenCL Programming Guide,” *Intel*, UG-OCL002 2017.05.08

3. Intel FPGA, “Intel® FPGA SDK for OpenCL Best Practices Guide,” *Intel*, UG-OCL003 2017.05.08

4. Intel FPGA, “Design Examples,” *Intel*, [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/developer-zone.html#design-examples [Accessed 11/06/2017]

5. Aaftab Munshi, “OpenCL Programming Guide,” *Pearson Education*, Jul. 2011.

6. Aaftab Munshi, Lee Howes and Bartosz Sochacki, “The OpenCL C Specification,” *Khronos OpenCL Working Group*, Version: 2.0 Document Revision: 33

7. Wim Vanderbauwhede, Khaled Benkrid, “High-Performance Computing Using FPGAs,” *Springer New York Heidelberg Dordrecht London*, ISBN 978-1-4614-1790-3

Appendix A Hardware and Software Specification

The hardware used in this project is the Hewlett Packard Enterprise's BladeSystem Blade Server equipped with an Nallatech 385A FPGA Accelerator Card. The operating system is Red Hat Enterprise Linux Server release 6.9 (Santiago) and an authorized Intel FPGA SDK for OpenCL is included.

The specification of the Server and OS is shown below:

```
---------------------------------------------------------------------
 System: Red Hat Enterprise Linux Server release 6.9 (Santiago)
 Secdrver: srv01894.soton.ac.uk
   Spec: CPUs: 56, RAM: 252.29 GiB total
Environ: ServiceNow: Development, Puppet: development
Purpose: HPC Test research blade
---------------------------------------------------------------------
```

The specification of the FPGA card and Intel FPGA SDK for OpenCL is shown below:

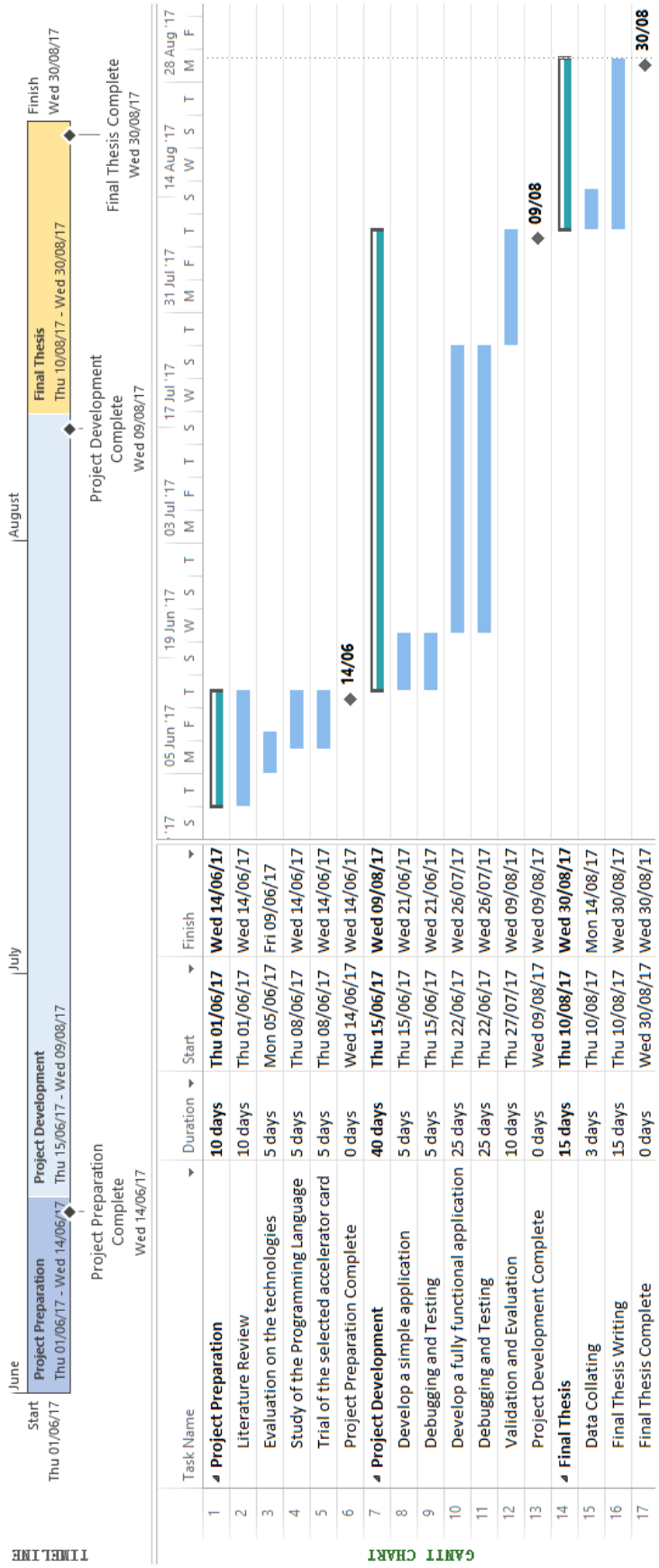
```
Platform info:
==========================
CL_PLATFORM_NAME        = Intel(R) FPGA SDK for OpenCL(TM)
CL_PLATFORM_VENDOR    = Altera Corporation
CL_PLATFORM_VERSION    = OpenCL 1.0 Intel(R) FPGA SDK for OpenCL(TM), Version 16.1.2

Device info:
========================
CL_DEVICE_NAME             = p385a_sch_ax115 : nalla_pcie (aclnalla_pcie0)
CL_DEVICE_VENDOR          = Nallatech ltd
CL_DEVICE_VENDOR_ID     = 4466
CL_DEVICE_VERSION         = OpenCL 1.0 Intel(R) FPGA SDK for OpenCL(TM), Version 16.1.2
CL_DRIVER_VERSION         = 16.1
CL_DEVICE_ADDRESS_BITS               = 64
CL_DEVICE_AVAILABLE                 = true
CL_DEVICE_ENDIAN_LITTLE              = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE        = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE     = 0
CL_DEVICE_GLOBAL_MEM_SIZE             = 8589934592
CL_DEVICE_IMAGE_SUPPORT              = true
CL_DEVICE_LOCAL_MEM_SIZE              = 16384
```

```
CL_DEVICE_MAX_CLOCK_FREQUENCY          = 1000
CL_DEVICE_MAX_COMPUTE_UNITS            = 1
CL_DEVICE_MAX_CONSTANT_ARGS            = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE     = 2147483648
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS     = 3
CL_DEVICE_MEM_BASE_ADDR_ALIGN          = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE     = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR  = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT   = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG  = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE = 0
```

Appendix B Project Management

Planned Gantt chart and Timeline:



Final Gantt chart and Timeline:

Appendix C Submitted Code

DNA_seq_gen----------DNA Sequence Generator

hello_world----------OpenCL Example Code

StringSearch_CPU----------String Search in C++

StringSearch_OCL----------String Search in OpenCL

SW_CPU----------Local Alignment in C++

SW_OCL----------Local Alignment in OpenCL