Edit distance on GPU clusters using MPI

Antoine Veenstra
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.j.veenstra@student.utwente.nl

ABSTRACT

In this paper, we describe how the edit distance problem can be distributed over a GPU-cluster using MPI and OpenCL. The performance of the cluster will then be compared to that of a single device.

Keywords

OpenCL, Edit distance problem, GPU-cluster, C++, case study, GPGPU program, MPI, Message Passing Interface, case study

1. INTRODUCTION

As the amount of data increases the need for parallel computation does so too. The edit distance problem is used in various fields of research[5]. Fields such as Computational Biology, Signal Processing, and Text Retrieval. This algorithm has already been implemented on a single Graphics Processing Unit (GPU)[2], but to decrease the processing time even further a logical step is to increase the number of GPUs[3]. These GPUs allow the processing of larger amounts of data in parallel than is possible on a CPU.

The existing implementation of the edit distance problem uses a dynamic programming algorithm, which is well-suited for general-purpose computing on graphics processing units (GPGPU). The implementation was written in C++ using OpenCL which can run on both NVIDIA and AMD GPUs. An alternative would have been CUDA, which has been developed by NVIDIA and runs exclusively on NVIDIA GPUs. OpenCL has been and will be chosen over CUDA in order to ensure compatibility with most GPUs.

To allow interaction between devices in a cluster a protocol is required. One standard which has been around for years is the Message Passing Interface (MPI)[6]. This interface will be used to distribute the algorithm on multiple (multi-)GPU nodes. A successful program using MPI and OpenCL has already been made[3], so it might be possible to make a MPI-OpenCL implementation of the edit distance problem.

By implementing the edit distance problem on a GPU-cluster instead of a single GPU the processing time could be reduced as the performance of a cluster could exceeds that of a single unit[3]. The goal of the research described in this paper is to implement the edit distance problem on a GPU-cluster using MPI. This goal gives us the research question mentioned in the following section.

24th Twente Student Conference on IT January , 2016, Enschede, The Netherlands.

2. RESEARCH QUESTIONS

The research question of this proposal is:

How much can the processing time needed to calculate the edit distance problem be reduced using a GPU cluster which uses MPI?

A possible division in subquestions is:

1. How can the algorithm be divided in separate processes?
2. How can the algorithm be run on multiple devices using MPI?
3. What is the optimal number of GPUs when considering cost and efficiency?

3. BACKGROUND

3.1 OpenCL

OpenCL is a programming language which allows a developer to run a kernel on a GPU or CPU[4]. It is a low level programming language which can run on most GPUs and CPUs and allows general purpose parallel programming across both CPUs and GPUs. The traditional CPU-based programming models do not allow the same complex vector operations on GPUs as OpenCL offers without the need to translate their algorithms to a 3D graphics API such as OpenGL. As mentioned before OpenCL is preferred over CUDA since the support of CUDA for GPUs and CPUs is limited to NVIDIA GPUs[1].

3.2 GPGPU programming

GPGPU programming is the use of GPUs to handle computation which traditionally is done by CPUs. The number of cores on a GPU is generally much higher than a CPU has, so a GPU can process more data in parallel. OpenCL loads a kernel on every OpenCL device and a host program manages the execution of the kernels. After a kernel has finished running the results are returned to the host program. The kernel on every core is the same, so the only way to change the outcome of the program is to modify the input of the program.

3.3 MPI

MPI is a standard specification in communication between computers which enables parallel computing[6]. An implementation of specification is freely available and will be used in this project. The implementation allows the

transmission of multiple datatypes and messages between nodes. It also provides a way to identify all the connected processes and assign an identifier to each process. These features could help dividing the workload over all nodes.

3.4 Edit distance

The edit distance problem is way of measuring how much two strings differ from each other[5]. The distance between two strings is measured by operations like inserting, removing, replacing, and rearranging characters. The complexity of the algorithm depends on what operations are allowed and the cost of these operations in the implementation. For this project only inserting, removing, and replacing are considered. These operations costs are C_i (insertion), C_d (deletion), and C_r (replacement) for each operation is variable.

An example input is:

$$S_1 = \text{Saturday}$$
$$S_2 = \text{Sunday}$$
$$C_i = 2$$
$$C_d = 3$$
$$C_r = 6$$

With this given input the output of the program should be 11.

4. METHOD

The research question is best answered with a comparison in time needed to compute the edit distance. One possible way to measure the difference in time required is to calculate the edit distance of two large strings, possibly DNA sequences, is to calculate it on every node of the cluster individually and then as a cluster working together. The average time every individual node requires can then be compared to the time taken by the cluster. This test should be repeated with different sizes of clusters and different types of nodes to obtain reliable and realistic results.

One thing to consider is how to distribute the two strings, especially if they are quite large. A few distribution methods are:

1. Saving the strings on the nodes before execution.
2. Choosing a master node which distributes the string to its slave nodes.
3. Hosting the strings on a separate server which does not take part in calculation of the edit distance.

Options two and three are the most realistic options, as the distribution of the data would be part of the processing time. Option one on the other hand reduces the bottleneck create by the reading speed of the storage device, which allows comparison of the speed without taking into account the hardware storing the strings.

Option three will probably be used, since a device uploading the strings to a cluster could be a real live situation.

5. CONCLUSIONS

The possible answers to the research question can be divided in different ranges. The first range is a negative time difference, which indicates the use of MPI, dividing the algorithm, or the use of a cluster in general is inefficient.

Table 1. Research schedule

Deadline	Task/Deliverable
Oct 12	Peer reviews
Oct 21	Final proposal
Oct 26	(No) Go
Nov 2	Research
Nov 16	Dividing the algorithm and implementing
Nov 23	Researching MPI
Nov 30	Implementing MPI
Dec 7	Optimizing and debugging
Dec 14	Getting a test environment
Jan 11	Writing the paper
Jan 18	Writing a presentation

The next range is a difference almost equal to zero, which indicates that dividing the calculation gives too much overhead, denying any reduction of the processing time required.

The third range is a difference up to n times as fast as the single node implementation, where n is the number of nodes in the cluster. This would mean that dividing the computation over a cluster is more efficient than calculating it on a single node, but that the overhead caused by the distribution of the algorithm impacts the performance. The distribution of algorithms over a GPU-cluster have already shown improved performances, so I suspect the result to be within this range.

The last range is the most improbable. It ranges from n times as fast to infinity times as fast, where n is the number of nodes. This would mean that the algorithm is more efficient as it is divided on multiple computers. If, for example, the number of nodes is doubled the speed would more than double. This would be strange since the processing power does not double.

The easiest way to find out in what range the result will be is to implement and test the algorithm on a GPU-cluster using MPI. This will be done according to the schedule mentioned below.

6. RELATED WORK

6.1 Edit distance problem on GPU

As mentioned before the edit distance problem has already been implemented on a single GPU[2]. This implementation also uses the operations insert, delete, and replace. This implementation will be improved if necessary and will probably be the base of the future implementation. The result of this project could be compared to this single node implementation to calculate the difference in time required to compute the edit distance of two strings.

6.2 Benchmark on a GPU-cluster

This is an MPI-OpenCL implementation of the LINPACK benchmark which was run on a cluster containing 49 nodes, each node containing two eight-core CPUs and four GPUs[3]. The implementation achieves 93.69 Tflops, which is 46 percent of the theoretical peak. It shows a successful implementation of MPI in combination with OpenCl, which is required in the future implementation of the edit distance algorithm.

7. RESEARCH SCHEDULE

The research schedule for this paper is given in Table 1.

8. REFERENCES

[1] Cuda gpus, October 2015.

[2] S. de Heus. A case study for gpgpu program verification.

[3] G. Jo, J. Nah, J. Lee, J. Kim, and J. Lee. Accelerating linpack with mpi-opencl on clusters of multi-gpu nodes. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):1814–1825, 2014.

[4] A. Munshi. *The OpenCL Specification*. Khronos OpenCL Working Group.

[5] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, Mar. 2001.

[6] University of Tennessee, Knoxville, Tennessee. *MPI : A Message-Passing Interface Standard*.