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 the steps required to distribute a verified implementation of the edit distance problem on a GPU over a GPU-cluster using MPI and OpenCL. The new implementation will be verified too and 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

1. INTRODUCTION

As the amount of data increases the need for parallel computation does so too, but this is no small feat. Multiple ways exist to process data in parallel, but one of the most efficient ways to do this is to use the Graphical Processing Unit (GPU). A GPU enables the parallel execution of a single operation on multiple variables, unlike the Common Processing Unit (CPU) which only allows for the execution of a single operation on a single value. Even if the CPU has multiple cores the amount of data processed in parallel is generally still inferior to that of a GPU.

To decrease the processing time even further a logical step is to increase the number of GPUs [2]. One could add more GPUs to their computer, but this is not a scalable solution since most motherboards only support a limited amount of GPUs. Another solution would be to make multiple devices work together, each containing at least one GPU. This is called a GPU-cluster.

Various algorithms have already been implemented on a single GPU device and were verified [1]. The verification of a program is important since it can guarantee the result of an algorithm is always correct without fail. If one wants to distribute a verified implementation over multiple nodes some steps have to be taken to ensure the implementation is still mathematically correct. Those steps will be explored while distributing a verified implementation of the edit distance problem.

The edit distance problem is used in various fields of research [5]. Fields such as Computational Biology, Signal Processing, and Text Retrieval. It is used to compare two strings or sequences of data, such as genome sequences.

The existing implementation of the edit distance problem uses a dynamic programming algorithm, which is well-suited for general-purpose computing on GPUs (GPGPU). The implementation was written in C++ using OpenCL, which can run on most GPUs [3]. An alternative for OpenCL 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) [7]. 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 [2], 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 exceeds that of a single unit [2]. The goal of this research is to implement the edit distance problem on a GPU-cluster using MPI and verify this implementation. This goal gives us the research question mentioned in the following section.

2. RESEARCH QUESTIONS

The research question of this proposal is:

What are the steps required to distribute a verified implementation of an algorithm on a GPU-cluster?

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. How can the verification of the implementation be guaranteed?
4. What is the optimal number of GPUs when considering cost, efficiency, and the amount of data compared?

3. BACKGROUND

3.1 OpenCL

GPGPU programming is the use of GPUs to handle computation which traditionally is done by CPUs. A CPU consists of one or more cores allowing single instructions streams and a single data stream (SISD). A GPU on the other hand has a Single Instruction stream and Multiple Data streams (SIMD). The number of cores on a GPU is generally much higher than a CPU has, so a GPU can process more data in parallel using its SIMD architecture.

One programming language allowing the developer to run programs on a GPU is OpenCL. OpenCL 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 [6].

In the OpenCL architecture one CPU-based program called the "Host" controls multiple GPUs and CPUs called "Compute Devices". Each of those compute devices consist of one or more "Compute Units", which each contain one or more "Processing Elements". These processing elements execute the OpenCL kernels provided by the host program. After such kernel has finished running the results are returned to the host program. The kernel on every processing element is the same, so the only way to change the outcome of the program is to modify the input of the program [4].

The memory hierarchy used in OpenCL is not equal to that of the physical memory configuration on GPUs. This is to prevent having to take into account every type of architecture, which would be tedious work as the amount of types is quite large. Each of the architectural devices discussed above have their own memory, which is inaccessible to components of the same type. Every processing element can access its own private memory, the memory of its compute unit, the memory of its compute device. The host memory can be accessed, but it is generally slower than the on-board memory [4].

The architecture and memory hierarchy already enforce the division of the algorithm if one wants to use every component of a GPU.

3.2 MPI

MPI is a standard specification in communication between computers which enables parallel computing. An implementation of this 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 [7]. These features could help dividing the workload over all nodes.

3.3 Edit distance

The edit distance problem is way of measuring how much two strings differ from each other [5]. The distance 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. The operations costs are C_i (insertion), C_d (deletion), and C_r (replacement).

An example input is:

$$S_1 = \text{Saturday}$$
$$S_2 = \text{Sunday}$$
$$C_i = 2$$
$$C_d = 3$$
$$C_r = 4$$

A possible shortest path is:

Step	**Cost**
Saturday	
Sturday	3
Surday	3
Sunday	4
Total:	10

With this given input the output of the program should be 10.

4. METHOD

The research question is best answered by distributing an verified implementation over a GPU-cluster. By answering every subquestion the steps required to answer the main research question should be described.

The first subquestion can be answered by proving mathematically that such a division of the algorithm is possible. An implementation could further proof the correct division of the algorithm.

The next subquestion can be answered by searching for examples of such implementations. After that the implementation answering the previous subquestion could be expanded using MPI.

The subquestion after that is best answered by describing the steps required to keep the implementation verified during the previous two subquestions. After this question has been answered the main question could be considered to be answered, but an GPU-cluster-based implementation is no good if the performance is inferior to an single GPU based implementation. Answering the next subquestion could show the performance difference of both implementations.

The last subquestions can be answered by testing. One possible way to measure the difference in time required 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. EXPECTED RESULTS

As only the answer of subquestion 4 will consist of tests only the expected results of these tests will be discussed. When comparing different cluster sizes, and lengths of data, the time required to compute the edit distance can be used to define most efficient setup. For each cluster size and data length the computation time can be divided in four 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.

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 to have some setups to be within this range.

The last range is the most improbable. It ranges from n times as fast to beyond, 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.

To compare the efficiency an implementation resolving the edit distance problem has to be made. This will be done according to the schedule mentioned in table 1.

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 [1]. 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.

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	Verifying the implementation
Dec 14	Optimizing and debugging
Dec 18	Getting a test environment
Jan 11	Writing the paper
Jan 18	Writing a presentation

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 [2]. 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] S. de Heus. A Case Study for GPGPU Program Verification. `http://referaat.cs.utwente.nl/conference/20/paper/7428/a-case-study-for-gpgpu-program-verification.pdf`.

[2] 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.

[3] Kronos Group. Conformant products. `https://www.khronos.org/conformance/adopters/conformant-products#opencl`.

[4] A. Munshi. *The OpenCL Specification*. Khronos OpenCL Working Group, November 2012.

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

[6] NVIDIA Corporation. CUDA GPUs. `https://developer.nvidia.com/cuda-gpus`, October 2015.

[7] University of Tennessee, Knoxville, Tennessee. Mpi : A message-passing interface standard. `http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`.