Master Thesis

Dynamic OpenCL - Distributed Computing on Cloud Scale

Dynamic OpenCL - Verteiltes Rechnen für Rechenzentren mit angebundenen Clouds

Florian Rösler

Hasso-Plattner-Institut an der Universität Potsdam

January 4, 2017

Master Thesis

Dynamic OpenCL - Distributed Computing on Cloud Scale

Dynamic OpenCL - Verteiltes Rechnen für Rechenzentren mit angebundenen Clouds

Florian Rösler

Betreuung
Prof. Dr. Andreas Polze, Max Plauth
Fachgebiet für Betriebssysteme und Middleware

Hasso-Plattner-Institut an der Universität Potsdam

January 4, 2017

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

1. Introduction

1.1. Motivation

As of today, there are various available approaches for programmers to distribute workloads across multiple devices within a single machine or even among multiple machines belonging to a cluster. Enhancing single threaded code to a multithreaded program that runs on multiple machines in parallel adds several layers of complexity to a project, which may introduce errors and increased maintenance efforts.

Frameworks like Open Multi-Processing (abbrv. OpenMP), CUDA or Open Computing Language (abbrv. OpenCL) allow for the parallel utilization of resources within a machine but in return require programmers to learn extensive APIs or write code in low level languages like C. At the same time cluster distribution approaches like Hadoop MapReduce require extended configuration management of the participating nodes. MapReduce as well as other techniques like the Message Passing Interface (abbrv. MPI) also add a layer of complexity to the written program code that is entirely focused on synchronizing the distributed parts. Ultimately, the combination of both levels of parallelization may lead to additional problems, which distract programmers from their original intentions.

While the creation of distributed algorithms implicates various problems on its own, running these programs in a cost efficient manner respresents another elaborate task. Clusters dedicated to running computational jobs in a shared environment should have as little underutilization as possible as idle resources still draw significant power and contribute to the total cost of ownership. Therefore the dynamic adaption of resources to the actual needs is a challenge with significant monetary benefits. As a result, cloud services have seen a steady increase in revenue due to their flexible billing options that allow booking resources for short amounts of time. Still, a pure cloud approach may introduce several disadvantages like higher costs or reduced performance due to geographical distances between the user and the utilized resources. Accordingly, a hybrid approach appears benefitial, providing a steady baseline of performance with local hardware. In the case of a computational peaks additional resources might be booked from an external provider, which allows for lower costs and better quality of service during times with high resource demands.

The previously mentioned issues have either been tackled in previous research or solutions are already incorporated in available libraries. Still, a framework that combines all available solutions is not freely available yet. Therefore the goal of this research is to create such framework that includes available solutions and provides additional functionality.

1.2. Goals

The main goal of this research is to provide a framework that assists programmers to distribute their programs within a hybrid cluster. Thus, the following requirements are defined in order to streamline the development:

Heterogeneity
The framework should support modern CPUs and GPUs. Thus, devices by the following vendors should be included: AMD, ARM, Intel and NVIDIA. Therefore various workloads and their specific hardware demands can be supported. Additionally, heterogenous hardware allows for a more granular employment of the required resources. For example a more cost saving ARM CPU could be utilized in low demand situations instead of a more potent but also energy demanding Intel CPU.

Resource Scalability
The general purpose of the future system is the management of a cluster with static resources. Still, it is desirable to have the ability to dynamically increase the computational power by utilizing external cloud services like Microsoft Azure, Amazon EC2 or others.

Scalable Speed
It is desirable that the user is able to scale the execution speed of their respective task. Therefore it must be possible to add or remove resources to or from a workload without interrupting the overall execution. With that ability the framework would be enabled to handle burst scenarios as well as save resources in the case of continuous low computational demand.

Ease of Programming
The introduced framework should provide programmers with an easy to use distribution mechanism without adding noticeable overhead to their development process. Preferably the resulting code should be created in a modern high level language on a high abstraction level.

Workload Diversity
While the submitted workloads could differ in size and required resources they might also have different hardware requirements. For instance, several workloads within the system could be optimized for CPUs and others for GPUs. The framework should support the execution of these workload types and additionally optimize the distribution across the hardware from a global system view.

Location Optimized Scheduling
Aside from various hardware that workloads can be optimized for, running programs on a hybrid cloud adds the issue of networking distances between the local cluster and the external cloud. Thus, certain tasks may see performance improvements when run locally instead of being initially sent to external resources. It is desirable to take these factors into account and consider network distances and bottlenecks during scheduling descisions.

2. Basics

2.1. Distribution Methods

2.1.1. OpenMP

OpenMP represents a compiler extension that allows programmers to harness parallel computational power on a machine. It adds functions and flags to the respective language (C, C++ and Fortran are currently supported), which indicate the execution environment how to parallelize the program.

2.1.2. MPI

Although OpenMP is a very powerful tool for parallelization on a single machine, in order to write larger scale software, communications across multiple machines are necessary. One standard tool to achieve this is MPI. MPI offers bindings to many languages and extends such by functions to identify a process and send as well as receive messages.

2.1.3. MapReduce

Due to its simple programming model, which mainly consists of a Map and a Reduce phase (there are many other phases in its implementations) it is a favoured approach for large clusters. Its most prominent implementation is Hadoop MapReduce, which in combination with Hadoop Distributed File System, is especially applicable for data intense jobs like log analysis and more.

2.2. CUDA

2.3. OpenCL

2.4. Aparapi

3. Related Work

There exist many research papers that cover partial problems of the goals that the planned framework aims to solve. Thus it is mandatory to showcase the particular specialties of these contributions and explain their general methodology in this chapter.

3.1. OpenCL Single Machine Distribution

3.1.1. Achieving a Single Compute Device Image in OpenCL for Multiple GPUs[7]

Framework that distributes workloads across multiple GPUs on a single machine. The memory of all included GPUs is consolidated in a single virtual memory space. Instead of launching the original OpenCL kernels, the kernels translated to CUDA and launched from the CUDA platform. In order to achieve an improved distribution, the OpenCL code is first translated to C to sample the memory access patterns.

Its distribution method is based on the memory consistency model of OpenCL, which states that "an update to a global memory location by a work-group does not need to be visible to other work-groups in the same kernel index space during the kernel execution" and a general independence of the execution of work groups (except memory barriers).

In order to reduce data transfers, not all data is transferred to every device but only the minimal necessary data for correct execution of every distributed work group. This is achieved by sampling the memory access and thus receiving a lower and upper memory bound of used memory addresses.

3.1.2. SOCL[5]

SOCL is a framework offering the following key points:

- Schedule and load balance kernels across multiple devices
- Manage memory transfers and coherency
- Dynamically adapt granularity of kernels

SOCL acts as a middleware with its own OpenCL ICD utilizing other installed ICDs on a machine. Thus, it is able to implement functionalities such as shared command queues for multiple devices by hiding the still remaining split command queues for each device. Through this method, load balancing techniques can also be applied.

In order to allow dynamic granularity adaptions, programmers have to supply the framework with a function that indicates how to divide the kernel with its respective data.

3.1.3. Static multi-device load balancing for OpenCL[9]

In this paper, an approach for executing a single kernel on multiple devices as well as a load balancing mechanism based on each devices's computing power is presented.

Instead of splitting the data (input buffers) into smaller parts, it replicates input buffers across the utilized devices and merges back the resulting output buffers.

As their future work they announce the development of dynamically adaptable load balancing algorithms.

3.1.4. A multi-GPU Programming Library for Real-Time Applications[13]

The proposed framework called MGPU promises to support CUDA as well as OpenCL to allow execution of a Kernel on multiple GPUs within a single machine. It's targeted at real time applications like MRI scan data, which would not inherently benefit from cluster setups due to latency penalties. In its current version MGPU focuses on CUDA due to its richer feature set.

MGPU employs a container architecture, which abstracts the device memory to a vector in main memory. Communication is done through MPI-based API.

3.1.5. STEPOCL[10]

This paper presents a framework called STEPOCL, which offers programming multi-device applications in their own developed domain specific language. Their key promoted values are linear performance scaling with increasing devices as well as comparable performance to hand written OpenCL applications.

STEPOCL takes an OpenCL Kernel with the following configurations:

- Data layout to split data among devices
- Tiling configuration depending on device type
- Meta control flow of the application (loops etc.)

During compilation, STEPOCL identifies available devices that it can distribute the submitted workloads to. The distribution of splits is done dynamically through a profiling mechanism that takes into account the performance of each device during the previous iteration of a Kernel execution. Therefore it is especially meaningful for iterative algorithms like N-Body or k-Means.

3.1.6. Fluidic Kernels[12]

The paper proposes distributing Kernels among the CPU and GPU of a machine. They showcase two workloads that benefit more from either CPU or GPU. In order to support dynamic executions each Kernel is launched each on the CPU and GPU, with the GPU executing the whole OpenCL range. The CPU starts execution of work groups from the other end of the range in so called subkernels. After each finished subkernel, the CPU reports

to the GPU the finished work group IDs. During execution of each work group, the GPU checks whether the current work group has finished yet and can therefore identify when it has reached work groups that have already been computed by the CPU and thus finish its execution.

3.1.7. Qilin[11]

Qilin tries to solve the problem of distributing a workload among a CPU and GPU by using adaptive mapping for Intel Threading Building Blocks and CUDA. It uses empirical data to predict optimal distribution loads from previous executions of a program.

When it encounters a program for the first time, it splits the work evenly among CPU and GPU, which each split the their parts into smaller subparts. For each subpart it records the execution time that are then used for curve fitting to find the optimal distribution ratio.

3.2. OpenCL Cluster Distribution

3.2.1. VirtualCL[2]

VirtualCL wraps the local OpenCL implementation on a host machine and forwards the API calls to previously defined remote machines, which run a VirtualCL daemon that receives calls from the host node. Thus, VirtualCL creates the impression that the host machine would have all devices in the cluster installed locally.

The authors show that network bandwidth and latency are the main bottlenecks of the approach but that long and compute intense kernels perform very well.

VirtualCL offers an extension, called SuperCL, which is aimed at reducing network transfers by allowing multiple kernels being submitted to a remote node for serial execution. When these kernels depend on each other, the results are not transferred back and forth but stored in temporary buffers. It also supports more complex use cases like alternating iterative kernels.

3.2.2. dOpenCL[6]

Similar to VirtualCL, dOpenCL forwards OpenCL APIs to remote devices to create the image of a single machine, which has a multitude of devices installed, even when these are in fact only available through network.

dOpenCL offers a device manager, which keeps track of utilization of devices within the cluster and schedules submitted kernels accordingly.

3.2.3. SnuCL[8]

SnuCL provides a mechanism to access many heterogenous compute devices as if they were available in a single machine. Besides OpenCL, SnuCL also supports CUDA commands, which are forwarded to remote machines and executed there. Additionally, SnuCL abstracts CPU cores as a compute unit each and transforms OpenCL code to C code, which is then executed on top of a thread of each core.

Furthermore, SnuCL introduces a virtual global memory, in which buffers may be shared among devices. It manages the consistency among shared buffers and attempts to minimize copy operations throughout the execution.

3.2.4. DistCL[3]

DistCL aims at merging multiple GPUs as a single virtual OpenCL device. DistCL creates a single context with one command queue that represents the aggregated device. For the purpose of distribution, DistCL splits Kernels into multiple Kernels with their respective subranges. In order to know, which data is accessed by a subrange, programmers have to supply a meta-function that determines the access pattern. Based on that function DistCL can only transfer relevant data to a device that executes a subrange.

3.2.5. MultiCL[1]

MultiCL is built on top of SnuCL and promises to schedule command queues approprietly among multiple devices in a cluster. They offer a round robin approach as well as an autofit mechanism. When queueing a computation, a flag can be provided, which labels the assumed bottleneck like computation, memory, I/O or iterative.

In their static scheduling approach they simply profile all available devices and select the best fitting device based on the flagged bottleneck property. In their dynamic scheduling algorithm they apply different mechanisms based on the flag:

Iterative Cache previous Kernel execution times

Compute-Intensive Minikernel Profiling

I/O-intensive Data caching and minimized transfer operations

3.2.6. rCUDA[4]

rCUDA aims to reduce the number of GPU accelerators in a cluster by not installing one in every machine but instead enable sharing the accelerators across the network. It also uses API forwarding to execute CUDA commands on the remote server and retrieve the respective result.

Their main focus when evaluating performance lies on power savings. When reducing the number of accelerators by 90%, they were able to achieve a 20% decrease in power consumption. Still, they are aware that this reduction can lead to a significant performance degradation depending on the running applications within the cluster.

4. Dynamic OpenCL

4.1. Distribution Approach

Choice of distribution -> why not MPI, MapReduce etc.
Choice of Framework -> SnuCL vs VirtualCL vs dOpenCL
Show Benchmarks of dOpenCL

4.2. High-Level Abstraction

Reason why we need the abstraction
Showcase Aparapi
Explain changes made to connect dopencl and aparapi
show benchmark of connection

4.3. Hybrid Cloud

explain the connection of between a local cloud and an external cloud
use amazon as example
also consider security

4.4. Job Design

show own code and what that means for programming workflow
design considerations and limitations of Aparapi

4.5. Scalable Speed

introduce general scaling idea based on the job design
show the necessary changes towards dopencl and aparapi

4.6. Optimized Scheduling

explain general scheduling structure of dynamo
bring out the fancy stuff like performance history based approach
how about simplex?

4.7. Exemplary Use Cases

pure shared local cloud without scaling options
  pure remote cloud with the server running in the cloud itself
  hybrid cloud
  use the framework as a library for a job itself

4.8. Limitations

show bottlenecks (memory of central machine and network in total)
  jobs that make sense
  job design limitations

4.9. Showcase: Dynamo Server

5. Conclusion

5.1. Results

5.2. Advantages and Disadvantages

5.3. Future Work

5.4. Final Thoughts

Bibliography

[1] Ashwin Mandayam Aji, Antonio J. Peña, Pavan Balaji, and Wu-chun Feng. „Automatic Command Queue Scheduling for Task-Parallel Workloads in OpenCL“. In: *Proceedings of the 2015 IEEE International Conference on Cluster Computing*. CLUSTER '15. Washington, DC, USA: IEEE Computer Society, 2015, pages 42–51. ISBN: 978-1-4673-6598-7. DOI: 10.1109/CLUSTER.2015.15. URL: http://dx.doi.org/10.1109/CLUSTER.2015.15.

[2] A. Barak and A. Shiloh. *The VirtualCL (VCL) Cluster Platform*. White Paper. Rachel and Selim Benin School of Computer Science, 2014.

[3] Tahir Diop, Steven Gurfinkel, Jason Anderson, and Natalie Enright Jerger. „DistCL: A Framework for the Distributed Execution of OpenCL Kernels“. In: *Proceedings of the 2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. MASCOTS '13. Washington, DC, USA: IEEE Computer Society, 2013, pages 556–566. ISBN: 978-0-7695-5102-9. DOI: 10.1109/MASCOTS.2013.77. URL: http://dx.doi.org/10.1109/MASCOTS.2013.77.

[4] Jose Duato, Antonio José Peña, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Ort. „RCUDA: Reducing the number of GPU-based accelerators in high performance clusters“. In: *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. Clusters;CUDA;Energy saving;High performance computing;Virtualizations; Caen, France, 2010, pages 224 –231. URL: http://dx.doi.org/10.1109/HPCS.2010.5547126.

[5] Sylvain Henry, Denis Barthou, Alexandre Denis, Raymond Namyst, and Marie-Christine Counilh. „SOCL: An OpenCL Implementation with Automatic Multi-Device Adaptation Support“. PhD thesis. INRIA, 2013.

[6] Philipp Kegel, Michel Steuwer, and Sergei Gorlatch. „dOpenCL: Towards a Uniform Programming Approach for Distributed Heterogeneous Multi-/Many-Core Systems“. In: *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pages 174–186. ISBN: 978-0-7695-4676-6. DOI: 10.1109/IPDPSW.2012.16. URL: http://dx.doi.org/10.1109/IPDPSW.2012.16.

[7] Jungwon Kim, Honggyu Kim, Joo Hwan Lee, and Jaejin Lee. „Achieving a Single Compute Device Image in OpenCL for Multiple GPUs“. In: *SIGPLAN Not.* 46.8 (Feb. 2011), pages 277–288. ISSN: 0362-1340. DOI: 10.1145/2038037.1941591. URL: http://doi.acm.org/10.1145/2038037.1941591.

[8] Jungwon Kim, Sangmin Seo, Jun Lee, Jeongho Nah, Gangwon Jo, and Jaejin Lee. „SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters“. In: *Proceedings of the 26th ACM International Conference on Supercomputing*. ICS '12. San Servolo Island, Venice, Italy: ACM, 2012, pages 341–352. ISBN: 978-1-4503-1316-2. DOI: 10.1145/2304576.2304623. URL: http://doi.acm.org/10.1145/2304576.2304623.

[9] Carlos S. de la Lama, Pablo Toharia, Jose Luis Bosque, and Oscar D. Robles. „Static Multi-device Load Balancing for OpenCL“. In: *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. ISPA '12. Washington, DC, USA: IEEE Computer Society, 2012, pages 675–682. ISBN: 978-0-7695-4701-5. DOI: 10.1109/ISPA.2012.100. URL: http://dx.doi.org/10.1109/ISPA.2012.100.

[10] Pei Li, Elisabeth Brunet, Francois Trahay, Christian Parrot, Gael Thomas, and Raymond Namyst. „Automatic OpenCL Code Generation for Multi-device Heterogeneous Architectures“. In: *Proceedings of the 2015 44th International Conference on Parallel Processing (ICPP)*. ICPP '15. Washington, DC, USA: IEEE Computer Society, 2015, pages 959–968. ISBN: 978-1-4673-7587-0. DOI: 10.1109/ICPP.2015.105. URL: http://dx.doi.org/10.1109/ICPP.2015.105.

[11] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. „Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping“. In: *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: ACM, 2009, pages 45–55. ISBN: 978-1-60558-798-1. DOI: 10.1145/1669112.1669121. URL: http://doi.acm.org/10.1145/1669112.1669121.

[12] Prasanna Pandit and R. Govindarajan. „Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices“. In: *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO '14. Orlando, FL, USA: ACM, 2014, 273:273–273:283. ISBN: 978-1-4503-2670-4. DOI: 10.1145/2544137.2544163. URL: http://doi.acm.org/10.1145/2544137.2544163.

[13] Sebastian Schaetz and Martin Uecker. „A multi-GPU Programming Library for Real-time Applications“. In: *Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*. ICA3PP'12. Fukuoka, Japan: Springer-Verlag, 2012, pages 114–128. ISBN: 978-3-642-33077-3. DOI: 10.1007/978-3-642-33078-0_9. URL: http://dx.doi.org/10.1007/978-3-642-33078-0_9.

Zusammenfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

A. Appendix

A.1. Eins

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

A.2. Zwei

Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

A.3. Drei

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Eidesstattliche Erklärung

Hiermit versichere ich, dass meine Master Thesis „Dynamic OpenCL - Distributed Computing on Cloud Scale“ („Dynamic OpenCL - Verteiltes Rechnen für Rechenzentren mit angebundenen Clouds“) selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, den January 4, 2017,

(Florian Rösler)