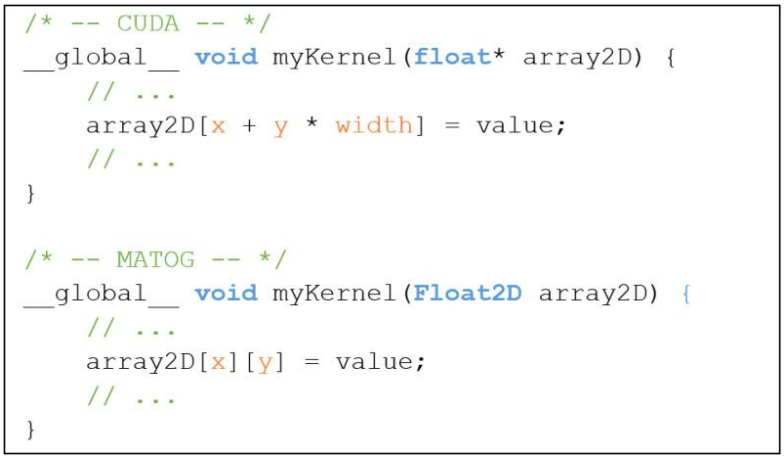
MATOG: Auto-Tuning On GPUs



```
/* -- CUDA -- */
__global__ void myKernel(float* array2D) {
    // ...
    array2D[x + y * width] = value;
    // ...
}

/* -- MATOG -- */
__global__ void myKernel(Float2D array2D) {
    // ...
    array2D[x][y] = value;
    // ...
}
```

Researchers
Nicolas Weber

Principal Investigator
Prof. Dr.-Ing. Michael Goesele

Project Term
2015 - 2015

Project Areas
Computer Science

Clusters
Lichtenberg Cluster Darmstadt

Institute
Fachgebiet Graphics, Capture and Massively Parallel Computing

University
Technische Universität Darmstadt

Introduction

In the last years, Graphics Processing Units (GPUs) have been increasingly used in many scientific applications. The main reason for this is their massive compute power for parallel applications, which can be used in many research fields. Programming interfaces such as CUDA or OpenCL allow even programmers with only little experience in programming to write basic GPU applications. Unfortunately it is still quite challenging to leverage the full potential of a GPU since this requires a deep understanding of the underlying hardware itself and a lot of experience in programming GPUs efficiently.

Methods

There are a lot of aspects which have to be optimized starting by choosing the correct algorithms, mapping the algorithms to the hardware resources and providing the data in the correct format, so that the complex memory hierarchy is optimally used. Memory access is especially difficult to optimize, as there are many degrees of freedom available, e.g. memory layouts, memory types, and caches. Furthermore with each new GPU generation the constraints for the memory typically change quite significantly which requires to re-optimize the entire memory access. Additionally even in the same GPU generation there are many differences in memory types used, memory interface width, number of processing units on the GPU, etc. which create a unique memory access behavior for each model even in the same GPU generation. That is why we created MATOG which aims to take the burden of optimizing the memory access in CUDA applications from the programmer. MATOG tries to interfere as little as possible with the actual programming and does not use a different compiler. Instead it generates C++ code which can be used platform independently and mimics the behavior of a Java-like memory access (Figure 1) as this is much less error prone as the classic C++ memory access for

multidimensional arrays and gives us the necessary degrees of freedom for optimizing.

Results

In our first paper [1] we showed the potential of such an automated optimization as it even outperformed hand optimized code but it also showed where we can continue to optimize MATOG. Currently we are working on removing framework specific limitations so that the integration of MATOG into existing GPU applications can be done with ease in any kind of GPU applications. Further we are working on techniques to reduce the number of benchmark runs required to find the best configuration for an application on a given GPU. For this we use the Lichtenberg's accelerator section with NVIDIA GPUs which allows us to run many different optimization algorithms on multiple evaluation applications in parallel.

Discussion

As one of the goals of MATOG is to optimize real world applications which typically have long execution times, it would take much more time if we only could execute these evaluations on a single machine in our institute. Figure 1 shows all evaluated configurations for a GPU based parallel sorting algorithm where MATOG is able to speed up the execution more than 2 times compared to a naïve implementation. This is a very limited evaluation example with only 27 possible configurations. Our more complex evaluation applications have up to 140.000 possible configurations which cannot be fully evaluated in a feasible time.

Figures

```
/* -- CUDA -- */
__global__ void myKernel(float* array2D) {
    // ...
    array2D[x + y * width] = value;
    // ...
}

/* -- MATOG -- */
__global__ void myKernel(Float2D array2D) {
    // ...
    array2D[x][y] = value;
    // ...
}
```

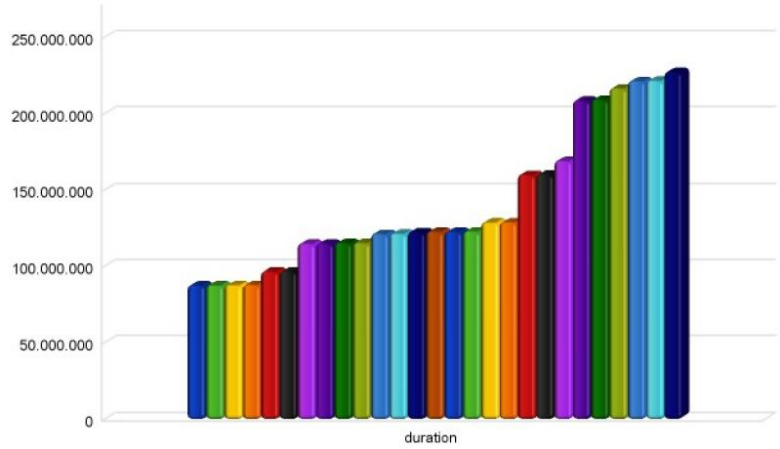


Fig. 1: This figure shows the quite significant difference of execution time (in nanoseconds) by simply changing the input data format for a GPU based parallel sorting algorithm. For more information: www.gris.tu-darmstadt.de/projects/matog

Reference

[1] N. Weber and M. Goesele (2014), Auto-Tuning Complex Array Layouts on GPUs, In Proc. Eurographics Symposium on Parallel Graphics and Visualization. http://dx.doi.org/10.2312/pgv.20141085

Last Update: 2022-07-14 23:21