Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 749 – 758

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

High-Level Programming for Medical Imaging on Multi-GPU Systems using the SkelCL Library

Michel Steuwer*, Sergei Gorlatch

Department of Mathematics and Computer Science, University of Muenster, Einsteinstrasse 62, 48149 Muenster, Germany

Abstract

Application development for modern high-performance systems with Graphics Processing Units (GPUs) relies on low-level programming approaches like CUDA and OpenCL, which leads to complex, lengthy and error-prone programs.

In this paper, we present SkelCL – a high-level programming model for systems with multiple GPUs and its implementation as a library on top of OpenCL. SkelCL provides three main enhancements to the OpenCL standard: 1) computations are conveniently expressed using *parallel patterns (skeletons)*; 2) memory management is simplified using *parallel container data types*; 3) an automatic *data (re)distribution* mechanism allows for scalability when using multi-GPU systems.

We use a real-world example from the field of medical imaging to motivate the design of our programming model and we show how application development using SkelCL is simplified without sacrificing performance: we were able to reduce the code size in our imaging example application by 50% while introducing only a moderate runtime overhead of less than 5%.

Keywords: SkelCL, Multi-GPU Computing, Algorithmic Skeletons, LM OSEM Algorithm, Image Reconstruction

1. Introduction

Modern high-performance computer systems increasingly employ *Graphics Processing Units* (GPUs) and other accelerators. The state-of-the-art application development for systems with GPUs is cumbersome and error-prone, because GPUs are programmed using relatively low-level models like CUDA [1] or OpenCL [2]. These programming approaches require the programmer to explicitly manage GPU's memory (including memory (de)allocations, and data transfers to/from the system's main memory), and to explicitly specify parallelism in the computation. This leads to lengthy, low-level, complex and thus error-prone code. For multi-GPU systems, programming with CUDA and OpenCL is even more complex, as both approaches require an explicit implementation of data exchange between GPUs, as well as separate management of each GPU, including low-level pointer arithmetics and offset calculations.

In this paper, we describe the SkelCL (Skeleton Computing Language) – a high-level programming model for parallel systems with multiple GPUs. The model is based on the OpenCL standard and extends it with three novel, high-level mechanisms:

*Corresponding author. Tel.: +49 251 8332744; fax: +49 251 8332742.
E-mail address: michel.steuwer@uni-muenster.de.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.239

1) *parallel skeletons*: pre-implemented high-level patterns of parallel computation and communication which can be customized and combined to express application-specific parallelism;
2) *parallel container data types*: collections of data (e. g., vectors and matrices) that are managed automatically across all GPUs in the system;
3) *data (re)distributions*: an automatic mechanism for describing data distributions and re-distributions among the GPUs of the target system.

The paper describes how the SkelCL model is used for programming a sample real-world application from the area of medical imaging, and how the model is implemented as the SkelCL programming library, using C++. Our focus is on programming methodology; therefore, we motivate our work using one typical imaging application and then study it in great detail throughout the paper.

The remainder of the paper is organized as follows. In Section 2, we introduce the application example used throughout the paper – the LM OSEM algorithm for medical image reconstruction. The application is used to identify requirements on a high-level programming model. Section 3 introduces the SkelCL programming model and its C++ implementation in the SkelCL library. In Section 4, we present experimental results for the LM OSEM algorithm using SkelCL, before we compare our contributions with related work and conclude in Section 5.

2. Iterative PET Image Reconstruction and the LM OSEM Algorithm

Our running application example in this paper is the LM OSEM algorithm [3, 4] for image reconstruction used in Positron Emission Tomography (PET). In PET, a radioactive substance is injected into a human or animal body, which is then placed inside a PET scanner that contains several arrays of detectors. As the particles of the applied substance decay, positrons are emitted (hence the name PET) and annihilate with nearby electrons, such that two photons are emitted in the opposite directions (see Fig. 1). These "decay events" are registered by two opposite detectors of the scanner which records these events in a list. Data collected by the PET scanner are then processed by a reconstruction algorithm to obtain a resulting image.

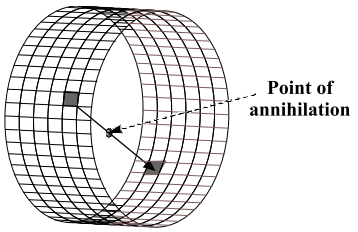

Fig. 1. Two detectors register an event in a PET-scanner

2.1. The LM OSEM Algorithm

List-Mode Ordered Subset Expectation Maximization [3] (called LM OSEM in the sequel) is a block-iterative algorithm for 3D image reconstruction. LM OSEM takes a set of events from a PET scanner and splits them into s equally sized subsets. Then, for each subset $S_l, l \in 0, \ldots, s-1$, the following computation is performed:

$$f_{l+1} = f_l c_l; \quad c_l = \frac{1}{A_N^T \mathbf{1}} \sum_{i \in S_l} (A_i)^T \frac{1}{A_i f_l}. \tag{1}$$

Here $f \in \mathbb{R}^n$ is a 3D image in vector form with dimensions $n = (X \times Y \times Z)$, A – the so called system matrix, element a_{ik} of row A_i is the length of intersection of the line between the two detectors of event i with voxel k of the reconstruction image, computed with Siddon's algorithm [5]. $\frac{1}{A_N^T \mathbf{1}}$ is the so-called normalization vector; since it can be precomputed, we will omit it in the following. The multiplication $f_l c_l$ is performed element-wise. Each subset's computation takes its predecessor's output image as input and produces a new, more precise image.