Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 309 – 318

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Regularity versus Load-Balancing on GPU for treefix computations

David Defour[a], Manuel Marin[a]

[a]*Univ. Perpignan Via Domitia, DALI F-66860, Perpignan, France*
Univ. Montpellier II, LIRMM, UMR 5506, F-34095, Montpellier, France
CNRS, LIRMM, UMR 5506, F-34095, Montpellier, France

Abstract

The use of GPUs has enabled us to achieve substantial acceleration in highly regular data parallel applications. The trend is now to look at irregular applications, as it requires advanced load balancing technics. However, it is well known that the use of regular computation is preferable and more suitable when working with these architectures. An alternative to the use of load balancing is to rely on *scan* and other GPU friendly parallel primitives to build the desired result; however implying in return, the involvement of extra memory storage and computation.

This article discusses of both solutions for treefix operations, which consist of applying a certain operation while performing a tree traversal. They can be performed by traversing the tree from top to bottom or from bottom to top, applying the proper operation at each vertex. It can be accelerated using either load balancing which maintains a pool of tasks while performing only the necessary amount of computation or using a vector friendly representation that will involve twice the amount of computation than the first solution. We will explore these two approaches and compare them in terms of performance and accuracy. We will show that the vectorial approach is always faster for any category of trees, but it raises accuracy issues when working with floating-point data.

Keywords: GPU computing ; regular versus irregular algorithms ; numerical quality ;

1. Introduction

In recent years, processors such as IBM cell SPUs, FPGAs, GPUs, and ASICs were successfully considered to provide speedup on numerous classes of applications. Of these, GPUs stand out as they are produced as commodity processors and exhibiting a number of processing cores doubling every year, revealing the current architectural trend. GPUs were used to improve the performance of regular computations such as those described in [1]. On such highly regular computations, GPUs can outperform a single core CPU by a large factor on average, that could be higher than 400 in some cases [2]. These large speedups are only possible for highly regular and computationally intensive classes of application. More recently, irregular computations on graphs such as list ranking [3] and connected components [4] were also considered. However, in these cases, the observed speedup compared to single core performance is of the order of 5 or less.

Treefix operations were first introduced by Leiserson and Maggs [5] as intermediate steps in a number of higher-level graph analysis algorithms. They defined two basic operations, *Rootfix* and *Leaffix*. Rootfix returns to

*Corresponding author. Tel.: +33-4-30-19-23-06 ; fax: +33-4-68-66-22-87.
E-mail address: david.defour@univ-perp.fr.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.194

each vertex of the tree the result of applying a certain operation over all its ancestors; Leaffix returns to each vertex the result of applying an operation over all its descendants. Rootfix and Leaffix have application for example in the Backward-forward sweep algorithm for electrical network analysis [6] or to evaluate the parsimony score of phylogenetic trees [7, 8]. In this article, we explore the available alternatives to accelerate these computations using GPUs.

The usual implementation of Rootfix and Leaffix is based on traversing the tree, from top to bottom or from bottom to top. The vertices are updated as visited, allowing to effectively propagate the accumulated result of the operation through the whole tree as the traversal progresses. The order of visit is relevant. Starting from the root, Depth-first or Breadth-first traversals are both valid alternatives. Ultimately, Rootfix and Leaffix can be viewed as performing a complete Breadth-first or Depth-first search over a tree, updating the vertices' weights as they are visited.

Successful implementations of parallel Breadth-first search over a general graph on GPU can be found in [9, 10, 11, 12, 13]. All of them rely on *level-synchronization*, i.e. processing every level of the graph in parallel, in order of depth. This is often implemented as an iterative process that performs one iteration per level. Some versions [11, 12, 13] examine every vertex of the graph at every iteration: if the predecessor was visited during the last iteration, then the vertex is visited. These methods perform a quadratic amount of work, as the graph can have, in the worst case, as many levels as vertices. A work efficient versions [9, 10] focus on producing, at each iteration, a vertex or edge *frontier*, including only those elements to be visited or traversed during that iteration. The main advantage of these methods is to exhibit a work efficient scheme, but have to deal with the irregularity of the graph data structure, which involves load imbalance and potential underutilization of SIMD lanes. Different load balancing strategies are applied to improve the performance achieved by these methods.

An alternative for performing Rootfix and Leaffix on a GPU, is to use a parallel-friendly representation of the tree consisting of three arrays based on the Euler-tour ordering. A series of highly regular parallel operations performed over these arrays, such as *scan*, allow to compute the result of Rootfix and Leaffix for a tree with n vertices in $O(\lg n)$ parallel steps, independently of the tree topology. However this methods relies on array of size $2.n$ with two times more computations than load balancing implementations.

The purpose of this article is to determine the best solution between a work efficient scheme thanks to irregular computation or a solution with regular computation with double the amount of operation to solve the treefix problem on GPUs. It makes the following contributions in the area of parallel computing:

- **Regular vs irregular algorithm comparison**. We present two different approaches that make use of data-parallelism to perform a distinctive operation over trees. One of them leads to an application that is highly regular, the other to one that is highly irregular and compares them in terms of performance.
- **Numerical quality analysis**. We compare the numerical accuracy of both methods when dealing with floating-point data as the amount and the order of operation is different.
- **Rootfix and Leaffix OpenCL implementation**. We provide a vectorial implementation of +Rootfix and +Leaffix in OpenCL. Even if there has been some work on implementing Rootfix and Leaffix in different languages [14, 15, 16], this is, to our knowledge, the first parallel implementation that could run on a GPU.

2. Presentation of Rootfix and Leaffix

Leiserson and Maggs [5] formally defined Rootfix and Leaffix as follows: given a weighted tree and a binary operator $\oplus$, Rootfix assigns to each vertex the result of applying $\oplus$ to all of the vertex's ancestors; Leaffix assigns to each vertex the result of applying $\oplus$ to all of the vertex's descendants.

From there, we can define the *+Rootfix* and *+Leaffix* operations, where $\oplus$ is the addition, as assigning to each vertex the sum of its ancestors and the sum of its descendants, respectively. Figure 1 shows an example. In particular, if all the vertices of the tree have weight 1, +Rootfix returns the depth of each vertex, and +Leaffix returns the size of the sub-tree rooted on every vertex.

2.1. Parallel algorithm

Regarding the type of trees considered, there are two easy cases of parallelization: balanced binary tree and linked list. For the balanced binary tree, Leiserson and Maggs [5] proposed a randomized algorithm that performs