A Redundancy-based greedy heuristic for the Data Layout Problem on Cache Oblivious Mesh Layouts

Zachary DeStefano
University of California, Irvine

May 16, 2014

Abstract

In Computer Graphics, the Data Layout Problem involves figuring out how to lay out the data units for a cache oblivious mesh layout in such a way that minimizes seek time required. Finding a deterministic solution to the problem is NP-hard so various heuristics have been proposed. In this paper, we present a solution to the problem that involves redundancy. We will describe the algorithm in detail as well as its running time and storage requirements. We will then show that in many cases we will get a better seek time than the best one without redundancy while not using much extra space. We will also show that our algorithm obtains a better seek time experimentally than one that does not use redundancy.

Introduction

The Data Layout Problem can be formulated as follows. The input is a linear sequence of data units. Each of the data units is assigned at least one color and many of them have multiple colors assigned. The length of a color is the distance from its first data unit to its last data unit. The data units can be rearranged as desired. The output we would like is the sequence of data units that will minimize the total length of all the colors.

In Yoon's paper **INSERT CITATION**, the Data Layout Problem is described as well as the metric that is motivating the above definition. Since the problem ends up being NP-hard, they proposed a heuristic that computes locally optimal solutions. In the paper, they described the access patterns as data units that are likely to be accessed together. This could end up leading to data units far apart from other ones with the same access pattern. Let's say that we have a data unit A which has two patterns attached to it. The two patterns have no data units in common besides A. It stands to reason that if we copy A and give the copy to one of the patterns, then both patterns have A and we have greatly improved seek time while only adding one extra unit. Therefore, we can save a lot of seek time without too much redundancy.

Algorithm Description

Min cut approaches

Algorithm Pseudo-code

Run-time and Storage Analysis

Benefits over Existing Algorithms

Experimental Results

Conclusions and Future Work

Acknowledgments

References