Statically Optimal Binary Search Tree Computation Using Non-Serial Polyadic Dynamic Programming on GPU’s

Mohsin Altaf Wani, University of Kashmir, Srinagar, India
Manzoor Ahmad, University of Kashmir, Srinagar, India

ABSTRACT

Modern GPUs perform computation at a very high rate when compared to CPUs; as a result, they are increasingly used for general purpose parallel computation. Determining if a statically optimal binary search tree is an optimization problem to find the optimal arrangement of nodes in a binary search tree so that average search time is minimized. Knuth’s modification to the dynamic programming algorithm improves the time complexity to O(n²). We develop a multiple GPU-based implementation of this algorithm using different approaches. Using suitable GPU implementation for a given workload provides a speedup of up to four times over other GPU based implementations. We are able to achieve a speedup factor of 409 on older GTX 570 and a speedup factor of 745 is achieved on a more modern GTX 1060 when compared to a conventional single threaded CPU based implementation.

KEYWORDS

CUDA, GPU, Granularity, Knuth’s Modification, Non-Serial Polyadic Dynamic Programming, Optimal Binary Search Tree, Speed-Up

1. INTRODUCTION

Graphics Processing Units are the mainstay of many-core parallel computing platforms. Today GPU’s are fully programmable parallel processing units having massive data processing capability. This data processing capability drives the use of GPU’s for general purpose programming GPGPU. GPGPU’s are particularly suited for data parallel tasks since they are based on SIMD (single instruction multiple data) and SPMD (single program multiple data) paradigms (Nvidia, Corp., 2011a). GPU’s are even used for algorithms which have highly irregular data access patterns like the branch and bound algorithms, with large gains in performance (Chakroun & Melab, 2012; Gmys, Mezma, Melab, & Tuyttens, 2016; Chakroun & Melab, 2013).

CUDA (compute unified device architecture) is a parallel computing architecture with a parallel programming model and instruction set architecture that is used to harness the parallel compute engine in NVidia. We are using two NVidia GPU’s for purpose of evaluation, one using an older FERMI based GTX 570 and another is recently launched GTX 1060.

Dynamic programming is an algorithm design technique used to find an optimal solution of a problem over an exponential number of candidate solutions (Neapolitan & Naimipour, 2003). Several problems like optimal polygon triangulation, chained matrix multiplication can be solved using dynamic programming (Cormen, Lieserson, & Rivest, 1990; Neapolitan & Naimipour, 2003).
The primary contribution of this paper is optimized parallelization of the dynamic programming algorithm so that maximum possible speedup is achieved. To that end we develop three parallel GPU based implementations and a CPU based implementation.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to Dynamic programming algorithm as applied to computing optimal binary search tree. Section 3 presents GPU architecture. Section 4 presents related work. Section 5 presents sequential algorithm for the given problem. Section 6 discusses the parallel implementations at length. Section 7 presents the results of our experiments. Section 8 gives concluding remark.

2. NON-SERIAL POLYADIC DYNAMIC PROGRAMMING ALGORITHM FOR CONSTRUCTING A STATICALLY OPTIMAL BINARY SEARCH TREE

Ordinarily, a binary search tree contains records that are retrieved according to the values of the keys. Our goal is to organize the keys in a binary search tree so that the average time it takes to locate a key is minimized. A tree that is organized in this fashion is called optimal. A tree in which all the keys have an equal probability of being the search key is said to be balanced if the difference between the height of left and right sub-trees is kept to a minimum e.g. 1 or 2. However, we are concerned with the case where the keys do not have the same probability.

The number of comparisons done by the searching procedure to locate a key is called the search time. We will determine a tree for which the average search time is minimal. Let Key1, Key2, . . . Keyn be the n keys in order, and let Pk be the probability that Keyk is the search key. If Ci is the number of comparisons needed to find Keyi in a given tree, the average search time for that tree is. Once the tree is constructed it cannot be modified hence the term statically optimal binary search tree.

Average Search Time = \sum (Ci.Pi) where 1 ≤ i ≤ n (a)

This value is to be minimized. We can generalize this for i ≤ k ≤ j instead of only for 1 ≤ i ≤ n.

The Dynamic programming algorithm for the solution of this problem proceeds as follows. Let Ck be the number of comparisons needed to locate Keyk in the tree. We will call such a tree optimal for those keys and denote the optimal value by A[i][j]. Because it takes one comparison to locate a key in a tree which contains a single key hence; A [i][i] = Pk.

Let tree 1 be an optimal tree given the restriction that Key1 is at the root, tree 2 be an optimal tree given the restriction that Key2 is at the root… tree n be an optimal tree given the restriction that Keyn is at the root. For 1 ≤ k ≤ n, the sub-trees of tree k must be optimal, and therefore, the average search times in these sub-trees are as depicted in Figure 1. This figure also shows that for each m ≠ k it takes exactly one more comparison (the one at the root) to locate Keym in tree k than it does to locate that key in the sub-tree that contains it. This one comparison adds 1 x pm (where pm is probability of Keym being the search key) to the average search time for Keym in tree k. the average search time for tree k is given by the equation below.

Equation 1 can be rewritten as

Average Search Time = A[1][k-1] + A[k+1][n] + \sum pm where 1 ≤ m ≤ n

Because one of the k trees must be optimal, the average search time for the optimal tree is given by

Average Search Time = \text{minimum} (A[1][k-1] + A[k+1][n]) + \sum pm where 1 ≤ m ≤ n 1 ≤ k ≤ n

This can be generalized for i ≤ k ≤ j.
Related Content

A Genetic Fuzzy Semantic Web Search Agent Using Granular Semantic Trees for Ambiguous Queries
www.igi-global.com/chapter/genetic-fuzzy-semantic-web-search/44714?camid=4v1

A Credible Cloud Service Model based on Behavior Graphs and Tripartite Decision-Making Mechanism
www.igi-global.com/article/a-credible-cloud-service-model-based-on-behavior-graphs-and-tripartite-decision-making-mechanism/165091?camid=4v1
www.igi-global.com/chapter/design-soa-based-framework-collaborative/64492?camid=4v1a

Discovering Perceptually Near Granules
www.igi-global.com/chapter/discovering-perceptually-near-granules/44710?camid=4v1a