3D Knapsack Problem: a parallel approach

Project for the GPU Computing course held by Professor Giuliano Grossi, University of Milan, Month 2022

Alessio La Greca

Abstract

The Knapsack Problem is a known combinatorial optimization problem. In the basic version of the problem, one is asked to fill a knapsack with as many items as possible in order to maximixe the profit of the contained items without "breaking" the knapsack. That is, given a knapsack whose maximum capaciti is c, and an array of items for which every i-th item has a weight w_i and a profit p_i, we want to find maximize the quantity:

$$\sum_{i=1}^{n} p_i x_i$$

where $x_i = \{0,1\}$, which tells us if the i-th item is packed or not, with the constraint:

$$\sum_{i=1}^{n} w_i x_i \leq c$$

The basic knapsack problem has been long studied, and even if its decision form is NP-complete, there exists a pseudo-polynomial time algorithm based on dynamic programming that can solve the optimization problem. A more complex version of the problem is its 3D variant: the 3D knapsack problem. It is more complex since some actual geometrical constraints are given, which must be taken into account in order to obtain a valuable solution for the problem. As of now, no algorithm is known for which, given an arbitrary instance of the problem, the optimal solution is found. However, different heuristics have been proposed, which can (more or less depending on the heuristic) be parallelized up to a certain degree. Our main goal in this study is to try to use the CUDA parallel capabilities in order to see if a good speedup can be obtained for running such algorithms, or (though this is very optimistic) if they can help us obtain better solutions.

Contents

1 Introduction

1.1 Problem formulation and simplifying assumptions

The 3D variant of the Knapsack Problem is more difficult than the ordinary one since, in this version, we are given some geometrical constraints. The main goal is filling a given *container* with as many *boxes* as possible from a given set of boxes, where:

1. Two boxes should never compenetrate in the container.
2. Each box is either fully packed or not packed at all in the container.
3. The width, height and depth (measures of the boxes' length along the x, y and z axis) of the boxes must be taken into account to ensure the two preceding constraints are met.
4. To also be realistic, we should allow the boxes to freely rotate *at least* in each of their six orthogonal configurations: WHD, WDH, HWD, HDW, DWH and DHW, where W, H and D represent respectively the initial width, height and depth of a box.

We also do some realistic assumptions that can help us simplify *just a little bit* the problem:

1. The container is a parallelepiped.
2. The boxes can only be parallelepipeds.
3. The boxes cannot freely rotate in all the possible ways along every axis, but can only rotate orthogonally around the main x, y and z axis (so that each of their faces either touches a wall of the container, a face of another box or nothing at all).
4. The value of a box is its volume. This makes sense since, for example, in cargo shipping, one would like to fill a container with as many goods as possible in order to minimize the number of travels required.
5. All the boxes can be fully described, for what is of our concern, by their width, height, depth and their position inside of the container, which is described by their *Left-Bottom-Behind* point (from now on simply *LBB* point). This means that, given a box, the x coordinate of its right wall is given by LBB.x + width; the y coordinate of its upper wall is given by LBB.y + height; the z coordinate of its front wall is given by LBB.z + depth.

1.2 Exaustive search and its intractability

Since this is an optimization problem, literature teaches us that we have four main approaches with which we can tackle the problem[1]:

- Analytical Solution.
- Complete/Exhaustive Exploration
- (Blind) Random Search
- Guided (Random) Search

Unfortunately, no Analytical Solution is known for solving the 3D Knapsack Problem (*otherwise we wouldn't be here*), so we have to exclude such approach.
One could think to tackle the problem with an Exhaustive Exploration then. After all, if we try all the possible configurations of the boxes and then we select the one with maximum occupancy of the container, we have found the optimal solution. Let's try to study the complexity of such an approach, with all the simplifying assumption made before.
Say we have n boxes to store in a container, ordered as we prefer. What are all the possible positions the very first box can occupy? Only the LBB corner of the container, so 1. But in how many ways can it be placed? 6, since it can be rotated in six different ways.
Placed the first box, we are given the second. Where can we place it? Well, we can place it on the right of the previous one, on top of it or in front of it, in the points (box1.x + box1.W, box1.y, box1.z), (box1.x, box1.y + box1.H, box1.z) or (box1.x, box1.y, box1.z + box1.D). In total, we have 3 choices. As before, the box can be placed in 6 different ways given how we can rotate it.
For the third box, it can be placed in 5 different positions. In fact, by placing the second box in the container, we removed one of the three previous possible positions, but added three more: to the right, on top or in front of the second box. Of course, also this has 6 possible rotations.
One could argue that a box can be placed, theoretically, in every position inside the container (given that its faces are parallel the the container's walls). So why prefer only the LBB corner for the first box and the right, top and front of previously placed boxes for the following ones? For two reasons mainly:

1. If we dropped this constraint, we'd have an intractable problem given just the very first box. In fact, an infinite number of points are possible given the floor of the container, so we'd never be able to fully explore the search space since, given only the first box, its possible positions are endless. Even if we decided to find a middle ground, saying for example that we discretize the points of the container in a finite set (for example by saying that all the possible points are given by starting with the LBB corner of the container and summing to it only multiples of a basic unit, for example 1 centimeter), we'd still have a very large search space, that grows bigger the finer is the unit of measurement we choose.

2. Thinking of how we humans would fill a container, this approach makes sense. If the container and the boxes are all parallelelipeds, we'd start by placing the first box in a corner, and then the following ones would always touch at least a face of another previously placed box. Intuitively, solutions built this way occupy less

volume than those where we place a box in an arbitrary point of the container without considering the current configuration of the placed boxes. This way of filling a container is known in literature as a *robot packable packing*, that tackles situations in which a robot with artificial hands packs the boxes into the bin[2].

Given this, we can inductively say that, if we have n boxes with a fixed order to place, the number of possible combinations is:

$$\left(\prod_{i=0}^{n-1}(2n-(2i+1))\right)\times 6^n$$

But that's not all: who said that the boxes should have a fixed order? Maybe, with another ordering, the volume occupied would be greater. In how many ways can n boxes be ordered? n factorial, or $n!$. So, the total number of possible configurations of the boxes is:

$$n!\times\left(\prod_{i=0}^{n-1}(2n-(2i+1))\right)\times 6^n$$

Let's be optimistic and assume that, in order to create a configuration (which means deciding the LBB coordinate for each box togheter with its rotation, and computing the volume occupied by the boxes fully contained in the container) we needed 10^{-9} seconds, that is, one nanosecond, and that there is no overhead whatsoever. For a non-trivial problem, where we have, say, 15 boxes (that would anyway still be very few for a real case scenario), we would need more than 681.636.227 years (value computed by omitting the production, so imagine the real number) to explore the whole search space.
We are left only with the Blind Random Search and the Guided Random Search. The Blind Random Search is not a practical solution, since it simply produces random possible solutions in the hope of finding a good one, but is a process based on sheer luck.
We are left with the Guided Random Search, an approach that generates candidate solution not at random, but by following a specific strategy. An important assumption that makes this procedure valuable is that similar candidate solutions should have a similar value (in our problem, a similar container occupancy). Whether this is true or not depends on the representation we give of the candidate solutions, which depends on the heuristic used to guide the search. With this, our problem is reduced to finding a good heuristic to exploit in order to explore the search space in the hope of finding, if not the global optimum, at least a local one.

2 Candidate solutions representation and its associated heuristic

Many representations of a possible solution for the 3D Knapsack Problem have been proposed, but the one that has proven to be the most successful is the Sequence Triple, proposed by Egeblad and Pisinger[3]. Although their Sequence Triple representation is the one of our interest, please note that our application will mainly focus on the work of Hanan Mostaghimi Ghomi[2], that, starting from such representation, proposed a Simulated Annealing approach for tackling the problem. Before delving into our work, it's useful to understand the building blocks we've used for our study and the tweaks we've made.

2.1 The Sequence Triple representation

Given n boxes, we define a Sequence Triple representation (ST representation) of their packing as a triple of three permutations of the numbers $1 \dots n$. These three permutations are called *A-chain*, *B-chain* and *C-chain*. They define the packing of the boxes according to the following rules:

- In the A-chain, if box i appears before box j, then box i is located to the left of, on top of, or in front of box j.
- In the B-chain, if box i appears before box j, then box i is located behind, to the left of, or below box j.
- In the C-chain, if box i appears before box j, then box i is located to the right, under, or in front of box j.

From these rules we can determine the relative placement of the boxes according to the following considerations:

- Box i is located on the left side of box j if it appears before box j in A-chain and B-chain and after box j in C-chain.
- Box i is located below box j if it appears before box j in B-chain and C-chain and after box j in A-chain.
- Box i is placed behind box j if it appears **before** box j in B-chain and **after** it in A-chain and C-chain, or if box i is placed **before** box j in all sequences.[1]

A cool thing to observe is how, in every case, box i appears before box j in every chain. This gives us the starting point for a placement algorithm. First, let's associate to each box an index from 1 to n. Then, we follow the B-chain like this:

1. We place the first box of the B-chain at the origin (the LBB corner of the container)
2. All the following boxes are placed according to their relative position to the already packed boxes. We can compute the coordinate for the LBB point of each to-be-packed box according to these formulas:
   - $x_i = max(0, max_{j \in P_x}(x_j + w_j))$

2.2 Placement Algorithm

2.3 Considerations on Vertical Stability

2.3.1 Hanan Mostaghimi Ghomi's proposed Vertical Stability implementation

2.3.2 Personal variant of the Vertical Stability implementation

References

[1] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Matthias Steinbrecher, and Pascal Held. *Computational Intelligence: A Methodological Introduction.* Springer, 2013.

[1] The bold here is used to mark those words that appear mistaken in the original paper. There in fact the words "before" and "after" are swapped.

[2] Hanan Mostaghimi Ghomi. Three-dimensional knapsack problem with pre-placed boxes and vertical stability. *Electronic Theses and Dissertations.*, page 4, 2013. 4987, https://scholar.uwindsor.ca/etd/4987.

[3] J. Egeblad and D. Pisinger. Heuristic approaches for two- and three- dimensional knapsack packing problem. *Computer & Operations Research, vol. 36*, pages 1026–1049, 2009.