1 Introduction

The purpose of this paper is to explore the use of GPU computing for solving the famous bin packing problem. Specifically, a massively parallel seesaw search program was constructed using Nvidia’s CUDA API and the Parallel Java 2 Library. The speed and quality of the program were compared to results from an optimal search program.

1.1 Combinatorial Optimization

Combinatorial optimization is a class of problems where an optimal solution must be chosen from a finite set of possible solutions. A combinatorial optimization problem consists of 3 parts: a set of configurations, one or more optimizations, and one or more constraints. To solve a combinatorial optimization problem, a configuration must be chosen which is the most optimal of all configurations which meet the constraints.

A stochastic local search is one method of solving combinatorial optimization problems. A random starting configuration is chosen, and is repeatedly modified slightly. After each modification, if the configuration satisfies all constraints and is the most optimal configuration yet discovered, it is recorded. After many configurations are considered, the best recorded configuration is chosen as the solution.

1.2 Bin Packing Problem

The bin packing problem is an NP-Hard combinatorial optimization problem. Given a set of items of various sizes, each item must be assigned to a bin such that the sum of item sizes in a single bin does not exceed the bin capacity. The items must be packed into the fewest number of bins possible.

In generalized bin packing, bins of various capacities may be used, whereas simple bin packing requires that each bin have the same capacity. In online bin packing, items are considered one at a time, and must be permanently assigned a bin before the next item is considered. Offline bin packing allows the use of all items at any time. This paper focuses only on simple, offline bin packing.

1.3 Seesaw Search

A seesaw search is a type of stochastic local search which is split into two phases: the optimization phase, and the constraining phase. During the optimization phase, the configuration’s degree of optimization is increased while the constraints are temporarily ignored. During the constraining phase, the configuration is modified so it once again satisfies all constraints. After either phase, the configuration overwrites the best known solution if it is found to be more optimal. This process is run many times, then the best configuration found during the search is chosen as the final solution.
If many seesaw searches are run in parallel, the best configuration of all runs can be chosen. In a \textit{massively parallel seesaw search}, a huge number of seesaw searches are run in parallel, which can greatly improve the solution quality over a single seesaw search. Massively parallel seesaw searches are made possible with modern multicore CPUs and GPUs.

1.4 Nvidia CUDA & Parallel Java 2

Graphics processors are designed with many hundreds or even thousands of processor cores, and are historically used for highly-parallel graphics calculations. More recently, GPUs have become accessible for less graphics-specific calculations through specialized APIs in what is called \textit{general-purpose computing on graphics processing units} (GPGPU).

Nvidia’s CUDA platform provides one such GPGPU API. Using CUDA, C-like programs can be written to control both CPU and GPU processing, as well as the data transfers between GPU memory and main system memory. By designing algorithms to exploit the parallel architecture of Nvidia GPUs, programs can gain huge boosts in performance. [1]

The Parallel Java 2 Library provides a platform for creating massively parallel programs in the Java programming language. It is also capable of accessing the CUDA API to execute programs on Nvidia GPUs. [2]

2 Related Work

There are several effective approximation algorithms designed for bin packing, such as Next Fit, Best Fit, and Best Fit Decreasing (BFD). BFD specifically has been proven to always provide solutions within $\frac{11}{9}$ of optimal solutions + 1 bin. [3]

Martello and Toth were among the first to create an non-exhaustive optimal bin packing algorithm they called the \textit{MTP} algorithm [5]. Later, Korf used several ideas from MTP to create his \textit{Bin Completion} (BC) algorithm which ran many times faster than MTP [6]. Schreiber and Korf later developed the \textit{Improved Bin Completion} (IBC) algorithm which was found to be as much as five orders of magnitude faster than Bin Completion [7]. Notably, BC and IBC are both \textit{anytime} algorithms (meaning they can be stopped at any time to produce a near-optimal solution), though no analysis of bin packing approximations is given in their original papers.

Attempts have been made to create parallel bin packing approximation algorithms [4] using strategies such as GPGPU [8] and Google’s \textit{MapReduce} programming model [9].

A massively parallel seesaw search program called \textit{WalkSackGpu} was introduced by Kaminsky for solving the knapsack problem on Nvidia GPUs [10]. The algorithm presented in this paper is largely based on the algorithm used in the WalkSackGpu program.

2.1 Approximation Algorithms

It is worth taking some time to discuss some of the approximation algorithms in greater detail, as they are referenced later in the proposed algorithm. These algorithms are very simple, yet reliably produce good results in most situations.

The Next Fit algorithm is perhaps the simplest of the bin packing approximation algorithms. Items are packed into a bin one at a time until the next item will not fit. Then, a new bin is created and items are packed into it in the same way. This repeats until all items are packed.

The Best Fit algorithm is a simple improvement on the Next Fit algorithm. Rather than packing each item into the bin at hand, the Best Fit algorithm considers all bins that have been created. The item is packed into the bin with the least amount of remaining space, but which will still fit the item. Only when an item will not fit into \textit{any} current bin will a new bin be created. This small change allows the Best Fit algorithm to fill in some of the gaps left behind in the Next Fit algorithm, reducing the number of bins used.
The Best Fit Decreasing (BFD) algorithm operates exactly as the Best Fit algorithm, except that it first sorts the items in decreasing order of size before placing them into bins. This modification tends to group larger items together as much as possible, saving the more versatile smaller items to be squeezed in at the end. In practice, BFD achieves solutions very near to optimal solutions in most cases.

3 Proposed Algorithm

The following sections detail the WalkPackGpu algorithm which can create approximate solutions for the bin packing problem. It is a seesaw search algorithm based on the WalkSackGpu algorithm [10], though it introduces a bit more complexity due to the nature of the bin packing problem.

3.1 Basic Seesaw Search

The conventional method for creating an algorithm which can solve the bin packing problem is to consider each item in some order, and place it in a bin. However, when formulating a seesaw search algorithm it is much more suitable to consider what the optimizations and constraints are for the given problem. For bin packing, the only optimization is that the number of bins used in the solution must be minimized. The two constraints are that the total space taken by all items in each bin must not exceed the bin capacity, and each item must exist within a single bin. With these in mind, we can define configurations as a set of bins which together contain all items in the problem. This eliminates the need to consider the second constraint, as no item may exist outside of a bin. Items can be moved freely between bins, and empty bins may be created or destroyed as needed.

To optimize a configuration, a bin must be removed. In WalkPackGpu, this is done by randomly selecting a bin, moving each of its items to a new bin, then deleting the bin. Items are each moved to a randomly selected bin. In this way, all items still belong to a bin, and the number of bins used in the solution is reduced. At times, an item may be added to a bin which does not have enough remaining space to hold it, causing the bin to be overfilled.

To constrain a configuration, an overfilled bin must have some of its items removed so that the total space taken by its items is brought below the bin capacity. This can be pictured as simply scraping off the top of a bin, leaving some number of items which fit cleanly inside. The removed items must be moved to another bin, so a new, empty bin is created to hold them. In WalkPackGpu, this process is accomplished by creating the new bin, then moving random items from an overfilled bin to the new bin until it is no longer overfilled. It is important to select random items to move to the new bin so the items at the “bottom” of the overfilled bin do not remain there indefinitely through each step of the algorithm.

3.2 Heuristic Improvements

Unfortunately, if a uniform probability distribution is used for bin selection in various parts of this algorithm, the results are unimpressive. We would like to have many bins thoroughly filled, but uniform selections tend to prevent this. Some bins will end up containing a single item for a long period of time, while other bins which become mostly filled will be destroyed in the optimization phase. Erratically relocating items in the optimization phase can also destroy well-filled bins by overfilling them. These problems can be solved by tweaking the distribution of bin selections in the optimization phase of the algorithm.

To avoid single-item bins, and to preserve well-filled bins, we can favor the former when selecting bins to be destroyed during the optimization phase. In WalkPackGpu, the amount of empty space in a bin determines its weight in the process of selecting bins to be destroyed. This causes bins with a small number of items, or small items to be merged into other bins, and avoids tampering with near-capacity bins.
Solving the erratic item placement problem can be accomplished by mimicking the Best Fit approximation algorithm: fit an item into the smallest gap available. Rather than doing this exactly, the WalkPackGpu algorithm favors bins with less empty space when selecting a destination for an item. As with the first heuristic, the amount of empty space in a bin determines its weight. However, bins which do not have enough empty space to fit the item are not considered unless no bin has enough empty space space to fit the item. In that case, the selected bin is guaranteed to be overfilled so the selection is done with a uniformly random distribution.

3.3 Initial State and Algorithm Warm-up

The introduced heuristics greatly improve the quality of solutions produced by the WalkPackGpu algorithm, however it may still take a large number of steps for the algorithm to close in on good solutions. This “warm-up” period is caused by a bad initial configuration, and is solved quite easily by using a quick approximation algorithm to reach a decent initial configuration. Originally, WalkPackGpu used an initial configuration which contained a single item in each bin, but this was changed to use the Next Fit approximation algorithm due to its simplicity.

3.4 Parallel Execution on GPU

To improve the performance of the seesaw search algorithm described above, many such searches can be run in parallel as a massively parallel seesaw search. Each available thread of a GPU can run an independent search, then a reduction can be performed to find the solution which uses the least number of bins. Nvidia’s CUDA API provides the tools required to run this massively parallel search. See Kaminsky’s implementation of the WalkSackGpu algorithm for a detailed explanation of how the reduction can be performed on a GPU using the PJ2 library [10].

4 Experimental Results

4.1 Uniform Case

For testing of bin packing algorithms, Korf recommends bin packing problems with a bin capacity of 1,000,000, and items whose sizes are chosen uniformly between 1 and 1,000,000 [6]. This class of problems will be called the uniform case. Uniform problems were used to compare WalkPackGpu to both the original Bin Completion algorithm and the Best Fit Decreasing approximation algorithm in Figures 1 and 2. Bin Completion can take a very long time if it encounters a difficult uniform problem with more than 100 items, so the number of items per problem ranges from 70 to 110.

Figure 1 shows the packing efficiency of each algorithm by measuring the number of items in a uniform problem against the number of bins into which each algorithm packed the items as a percentage of the optimal solution. Since Bin Completion is an optimal algorithm, it is guaranteed to remain at exactly 100% of the optimal number of bins. The “1 Optimal Bins + 1” is the theoretical upper bound for the BFD algorithm, and was included to show how BFD tends to stay closer to the optimal solution with uniform problems. Notice how both BFD and WalkPackGpu stay just over 100% of the optimal solution. BFD seems to use slightly less bins than WalkPackGpu, though increasing the number of steps closes this gap.

Figure 2 shows how the runtime of each algorithm increases with the problem size. Immediately apparent is the erratic behavior of Bin Completion. The times displayed in this graph are the averages over only 10 problem instances per problem size, and the spikes seen are where only one or two of those instances turned out to be difficult for the algorithm to solve. The other two algorithms have much less variance in their solve times, though it is clear that BFD is much faster than WalkPackGpu with any reasonable number of steps.
Figures 3 and 4 are the same as Figures 1 and 2, except the number of items per problem was scaled up. Bin Completion was not capable of finding timely solutions for this size problem, so it was omitted from these figures. In Figure 3, the number of bins in the optimal solution was approximated using a lower bound value. For these larger values, WalkPackGpu both decreases gradually in solution quality, and linearly increases in runtime. Meanwhile, BFD continues producing very good solutions and takes almost no time to solve problems in comparison to WalkPackGpu.

### 4.2 2/3 Case
BFD clearly can produce results very close to the optimal solution in the uniform case. It can be shown, however, that there are some cases where this quick approximation algorithm uses significantly more bins than the optimal solution.
The order in which BFD considers bins makes it prone to a “spoiler” effect, where an item is placed in a bin but takes up too much space to allow any other item into the bin while still leaving a sizeable chunk of empty space. For example, take the bin packing problem with a bin capacity of 10 and the following item sizes: [4, 4, 3, 3, 3, 3]. The single optimal solution contains only two identical bins which are completely filled: [4, 3, 3] and [4, 3, 3]. But with this particular problem, BFD will pack the first 4 into a bin, then pack the second 4 into the same bin because it is the tightest fit. Since the remaining capacity of this first bin is only 2, the leftover items of size 3 will not fit. This means the BFD solution uses 3 bins: [4, 4], [3, 3, 3], and [3]. In effect, the second 4 “spoiled” the first bin because it was slightly too large to allow any other item to fit with it.

Using this effect, several classes of bin packing problems can be created which are handled poorly by BFD. One such case is identical to the uniform case described previously, except that items are chosen uniformly between 200,000 and 300,000 rather than between 1 and 1,000,000. We will call
this the 2/3 case. One would correctly expect most bins in optimal solutions for these problems to contain four items due to the average item size being 250,000, or exactly one quarter of the bin capacity. BFD has difficulty with the 2/3 case because it attempts to pack many of the items near 300,000 together, which takes up too much space to allow four items. The result is that BFD solutions tend to use more bins than necessary. This can be seen in Figure 5.

Figures 5 and 6 are identical to Figures 1 and 2 except that 2/3 problems are generated rather than uniform problems. Also, Bin Completion was found to have great difficulty with the 2/3 case so it was once again replaced with a lower bound value for Figure 5. It is easy to see that BFD struggles with these problems. Its solution sizes are consistently about halfway between optimal and the theoretical upper bound, as shown in Figure 5. Meanwhile, WalkPackGpu is still able to find nearly optimal solutions. WalkPackGpu runtimes are somewhat lower for the 2/3 case than they were for the uniform case, though they are still substantially higher than BFD.
As can be seen in Figure 7, WalkPackGpu continues to produce significantly better results than BFD for much larger problem sizes. There is a gentle loss in quality as the number of items per problem increases, but this is easily remedied by increasing the number of steps. Unfortunately, there is still a great cost for increasing the number of steps for such large problems. Figure 8 shows how BFD continues to find solutions in far less time than WalkPackGpu for large problems.

5 Conclusion

The WalkPackGpu algorithm has shown that a GPU-accelerated parallel seesaw search is capable of producing good solutions for bin packing problems with a large number of items where optimal algorithms such as Bin Completion are infeasible. The WalkPackGpu algorithm has also been shown
to find results which approach the quality of the popular Best Fit Decreasing algorithm in general, and surpass it significantly in certain cases. Unfortunately, WalkPackGpu cannot match the speed of BFD.

The seesaw search algorithm has been shown to be flexible enough to apply to a problem without straightforward optimization and constraint operations. It has also been shown that the usage of heuristics in the stochastic portions of these operations can significantly improve the results of a seesaw search. The heuristics used in WalkPackGpu were based on simple approximation algorithms, and were able to overcome some of the original algorithms’ shortcomings. This suggests that seesaw searches may be a means to improve existing approximation algorithms for other difficult problems.

References


## Appendix A  Project Hours

<table>
<thead>
<tr>
<th>Week</th>
<th>Hours</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>Literature Review</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>Wrote Introduction and Related Works sections</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>Implemented general bin packing classes</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>Implemented sequential bin completion</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>Started implementing parallel bin completion</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Studied CUDA architecture</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Fully implemented parallel bin completion</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Started writing basic GPU algorithm</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Testing different implementations for basic GPU algorithm</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Began preliminary performance analysis of basic GPU algorithm</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Analyzed other algorithms for weaknesses</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>Started developing heuristics to improve GPU algorithm</td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>Analyzed and refined GPU algorithm performance</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>Finished writing final report</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>Final touches on code and report</td>
</tr>
</tbody>
</table>

Total | 128 | -  |