Massively Parallel Computing Applied to the One-Dimensional Bin Packing Problem

Judith O. Berkey
George Mason University
Fairfax, Virginia

ABSTRACT

This paper presents a parallel bin packing algorithm for packing \( N \geq n^2 \) pieces in a massively parallel computing environment of \( n^2 \) processors operating in SIMD mode. A new heuristic approach is developed that improves packing efficiency by a careful structuring of the input. Results are compared with parallel versions of the traditional Next Fit and First Fit packing algorithms.

Keywords: Bin packing, SIMD Algorithms, Parallel Processing, Analysis of Algorithms

INTRODUCTION

Bin packing is an optimization problem that plays an important role in many combinatorial problems existing in the areas of computer science and operations research (Ref. 3). In this problem, a set of \( n \) objects having weights between 0 and 1 is placed in a set of unit bins so as to minimize the number of bins used. The optimal solution for this problem is known to be NP-Hard but a number of heuristics have been developed that find solutions with provable and acceptable bounds (Ref. 2).

This paper presents a parallel algorithm for packing \( N \geq n^2 \) pieces in a massively parallel computing environment of \( n^2 \) processors operating in SIMD mode. The new heuristic approach that is developed utilizes a grid topology that structures the input to increase packing efficiency without sorting the data set. The results of implementing this algorithm on a Connection Machine 2 System are compared with the results obtained from parallel versions of two traditional bin packing algorithms.

A parallel algorithm must yield an acceptable solution while maximizing the utilization of the processors and minimizing the total interprocessor communication time (Ref. 4). It is therefore important to choose an appropriate representation for both the pieces and the bins. The algorithm should distribute the pieces to the processors in such a manner that both the movement of pieces between processors and the packing time for the bins is minimized. The grid topology that we use increases the packing efficiency while decreasing the amount of internode communication that is necessary.

THE GRID PACK ALGORITHM

The algorithm begins by dividing the pieces into two classes. Pieces that are between 0.5 and 1.0 in size are labeled "bin starter" pieces. The remaining pieces, that is, those with size between 0.0 and 0.5, are "bin filler" pieces. The goal of the algorithm is to pack pieces in the bin starter class of size \( p_i \) into a bin with capacity as near as possible to \((1-p_i)\). Note that the pieces do not have to be presorted. The division can be done as part of the data input step.

The bin starter pieces are packed first. Observe that since pieces of size greater than 0.5 cannot be combined, each one must be packed in a separate bin. Thus, at this point, the packing is optimal. This packing is done in parallel and the resulting bins are maintained in each processor's local memory. Then the algorithm packs the remaining pieces by matching pairs of bin starter and bin filler pieces whose combined size fills a bin as completely as possible. We want to minimize the amount of empty space remaining in the bin after a packing cycle has been performed. It has been shown that processing the pieces in order of decreasing size will increase the packing efficiency (Ref. 2). Our structuring of the input allows us to do this without actually sorting the input data.

Packing continues in this fashion until all bin filler pieces have been processed. A piece that cannot be packed in any bin in the set of bins resulting from the bin starter packing is labeled as "well-traveled." The
well-traveled pieces will be combined using a next fit packing in the final phase of the algorithm. The packing algorithm is presented in Figure 1.

![Algorithm Parallel-Pack](image)

```plaintext
Algorithm Parallel-Pack
begin
  in parallel
    send pieces to processors
  pack bin starter pieces
Pack-Loop
    begin
      find all bins that can pack bin-filler piece
      then
        pack piece in the smallest cap. bin
      else
        mark piece as well-traveled
        until all pieces have traveled
      end
      if piece is well-traveled
      then
        Next-Fit pack piece
    end
end
```

**Figure 1. Grid Pack Algorithm**

**IMPLEMENTATION**

The Grid Pack Algorithm was implemented on a Connection Machine 2 in *Lisp. The input data, consisting of a list of pieces, was divided as described above and placed in a two-dimensional array on the host machine. Each row of the array received the pieces that had sizes within a specified interval. These intervals are determined by an analysis of the piece size distribution.

The packing loop is performed in parallel by all active processors. The size of each bin filler piece is broadcast in turn to all processors containing bin starter pieces. From the set of all processors that can pack the piece, the processor with the maximum self-address is selected to pack it. When all bin filler pieces have been processed, the well-traveled pieces are combined into new bins using an iterative next fit packing procedure.

Figure 2 is an example of the piece input for a set of one hundred input pieces. In our sample data the piece sizes were randomly generated and are evenly distributed between 0 and 1.

Initially, the 50 bin starter pieces were packed into 50 bins. Forty-seven of the bin filler pieces were added to these bins and one new bin was required to pack the three well-traveled pieces for a total of 51 bins. The optimal packing of this piece set required 49 bins.

![Figure 2. Input Pieces](image)

```
04 04 02 02 08 08 04 01 02 08
12 18 15 18 16 10 10 13 14 18
21 23 21 26 26 28 21 21 22 24
31 31 30 31 30 37 32 32 37 37
41 42 42 40 42 40 41 44 41 43
58 58 53 55 58 54 56 55 56 54
60 62 60 63 64 64 65 61 60 61
77 77 76 77 70 70 70 75 74 75
80 81 80 87 86 81 86 88 84 86
96 92 92 95 92 92 93 93 92 93
```

**RESULTS**

The algorithm was tested with sets of data of size 100 and 1000. The results presented in Table 1 are the average of 5 runs for a data set of size 1000. It compares the results obtained by the Grid Pack algorithm to those obtained by using parallel versions of the traditional Next Fit and First Fit bin packing heuristics. The Parallel Next Fit and Parallel First Fit algorithms that were used for comparison are adaptations of those presented in Reference 1. When the input was unsorted, or sorted into nondecreasing sequence, the Grid Pack algorithms resulted in a packing that was much better than the packing obtained with either Parallel Next Fit or Parallel First Fit. In the case where the pieces are sorted into nonincreasing order, the packing from Grid Pack was still much better than the Parallel Next Fit packing and was comparable to the Parallel First Fit packing.
CONCLUSIONS

Bin packing algorithms can be efficiently implemented in a SIMD processing environment. The use of data partitioning to initialize the packing appears to be a practical method of allocating the packing workload to a set of parallel processors while maintaining the integrity of the packing algorithm.

This study supports our contention that parallelism offers opportunity for improvement in the efficiency of packing algorithms and that the SIMD model of computation with its data parallel programming approach is appropriate for bin packing problems.

ACKNOWLEDGEMENTS

The author would like to thank the Naval Research Laboratory in Washington, D.C., for providing access to the Connection Machine 2 System.