Abstract: *In this paper, we describe our experience in developing parallel implementations for the* Bin Packing *problem. Bin packing algorithms are studied to understand various resource allocation issues and the impact of the different packing heuristics on the packing efficiency. The seemingly serial nature of the bin packing simulation has prohibited previous experimentations from going beyond sizes of several thousands bins. We show that by adopting fairly simple data parallel algorithms a linear speedup is possible. Sizes of up to hundreds of thousands of bins have been simulated for different parameters and heuristics. In our experiments, we focussed on the well known first-fit heuristic. We have also considered another potentially superior heuristic which we have called k-delayed best fit.*

* This work was supported by DARPA N00039-88-C-0163.

1 Introduction

New, increasingly complex hardware configurations require software developers to implement the same software system on a multitude of platforms, thus making the software development process even more challenging. Our approach to this problem is based on the knowledge-based software development approach [3] in which programs are specified at a very high level, and, via transformations, efficient implementations are developed for each hardware configuration. In this paper, we describe our experience in developing parallel implementations for the *Bin Packing* problem on massively parallel systems [1] – namely the Connection Machine (CM) [9] – using this approach.

Bin packing is studied to understand various resource allocation issues. Of particular interest is the impact of packing heuristics on packing efficiency. In this respect, the asymptotic behavior of the different heuristics is considered as a major criterion in comparing their relative performance. The seemingly

serial nature of the bin packing simulation has prohibited previous experimentations from going beyond sizes of several thousands bins. In this paper, we show that by adopting a fairly simple data parallel algorithm [5] a linear speedup over straightforward serial implementation is possible allowing us to simulate the packing of hundreds of thousands of bins. We begin the paper by introducing the Bin Packing problem through a sequence of abstract implementations. We follow that with a description and comparison of the different algorithms we tried on the CM. Next, we briefly present our simulation results for the well known *First Fit* heuristic and the potentially superior *k-delayed Best Fit* heuristic. We end the paper with a summary of open questions and conclusions.

2 Bin Packing

Bin packing is the study of packing a series of requests (or weights) into a number of fixed size resources (or bins). The requests usually follow a specific distribution. For a given number of such requests, *Optimal fit* is the arrangement of values that uses the fewest number of bins. Finding this optimal arrangement is known to be exponentially difficult [7], and therefore online heuristics to approximate this optimal performance have to be used (an online heuristic must sequentially service the requests as they arrive with no relocation of weights). One such heuristic is the well-known *First Fit* approximation where the bins are inspected sequentially and the newly arriving weight is placed in the first bin that has enough room for it. Another heuristic is *Best fit* which places the weight in the bin that will leave the smallest possible unfilled space. For many practical distributions the performance of these heuristics is acceptable. An interesting question, however, is where, and how well, do these heuristics approximate optimal fitting.

We adopt the expected-case model which assumes that requests follow a uniform distribution over the interval (a, b), for $0 \leq a \leq b \leq 1$. To judge the goodness of a given packing – that is, how dense the packing is – we use the *Bin Ratio* (BR) measure [7, 2]. The BR is the ratio between the available space and the used space. Obviously, we would like this ratio to approach unity, in which case we have a *Perfect Packing*. The BR measures how full the bins are rather than how optimal the packing is. It measures the "badness" of the packing by comparing the total available slack with a theoretic lower bound that might be impossible to achieve. It has been shown that a perfect packing (i.e. a packing that results in a BR of 1) exists in an expected-case model when $a = 0$, $b = 1$, $n \to \infty$. Generally speaking, it is possible to show that if the interval (a, b) can be partitioned into units each symmetric around $1/k$, for any set of integers $k > 1$, then as $n \to \infty$ a perfect packing is possible. For a given value of $a \neq 0$, values of b that satisfies this condition can be enumerated. It is easy to prove that this makes a perfect packing always possible whenever $a = 0$.

A packing heuristic is termed *optimal* if it results in a BR of 1 as $n \to \infty$. In order to demonstrate the non-optimality of a bin packing heuristic, it is sufficient to show that for a distribution of weights for which a perfect packing exists, the empty space for that heuristic is $\Omega(n)$ (that is the BR approaches some constant greater that 1). Since a perfect packing exists whenever $a = 0$, it follows that if the BR achievable using a heuristic does not approach 1 when $a = 0$, then the heuristic is not optimal. McGeoch and Tygar [7] demonstrated that: for $b = 0.85$, First Fit is not optimal for value of $0 \leq a < b$, $a \neq 0.15$. That is, although optimal for $a = 0$, $b = 1$, First Fit is not optimal for $a = 0$, $b = 0.85$. Their proof is extended to the Best Fit heuristic as well.

An interesting problem is the characterization of First Fit (or any other heuristic) for arbitrary values of a and b. Although

theoretical analyses like those done by McGeoch and Tygar are useful, they do not provide precise answers to questions concerning optimality for arbitrary values of a, b. The solution is therefore to simulate the behavior of these heuristics as $n \to \infty$ for any possible value of a and b. Such a simulation provides us with an integrated "picture" of the performance of different heuristics under different distribution. This picture might help to establish a hypothesis as to a closed form representation (or approximation) for the value of the BR. The problem, however, is that the convergence of the BR is slow making it necessary to use very large numbers of bins in the simulation. Experiments with n as large as $64,000$ have been tried [7] to compute the packing densities achievable by First Fit. Results have confirmed the aforementioned anomaly around $b = 0.85$ (the value of a being fixed to 0). Moreover, results have shown that the BR function is smooth in the neighborhood of $b = 0.85$. Using the CM we were able to simulate the bin packing for sizes up to $512,000$ for uniform distributions of values over the open interval (a, b) where $0.0 < a < b \leq 1.0$.

3 Program Models for Bin Packing

Under the knowledge-based software development approach, system implementors do not develop platform customized programs. Rather, they produce a family of very high level programs, together with one or more formal *plans* for its realization as a concrete system. A plan is a structured description of the mechanical steps needed to refine a program model by applying program transformations. The transformations may either be developed by the implementor or they may be drawn from a knowledge-base of rules whose validity has been certified and whose domains of effectiveness are well documented. Knowledge-based software development increases productivity and reliability without dimin-

ishing the efficiency of systems that can be produced.

In this section, we briefly present a scenario for the development of program models for the Bin-Packing problem. We shall proceed by describing some nodes of interest in an acyclic graph of abstract implementations. The graph edges signify the transformations necessary to travel between the nodes. We start by considering the typical abstraction for the sequential case:

```
B1. BinPack(Bins,Result)
    begin
        repeat
            ChooseWeight(Weight)
            InsertBin(Bins,Weight)
            if FailInsert then exit
    end
```

Each time through the loop a weight is chosen, then it is inserted in some available bin according to the packing heuristic. If the insertion fails the loop is exited. No other weight is generated until the current weight finds a repository.

An obvious candidate implementation for massively parallel systems is the *Pipelined Request* approach (see section 4.3). Here, a sequence of weights traverses the pipeline of bins each of which decides, locally, whether to accept the weight or not. Upon acceptance a weight is removed from the sequence. Thus, a bin sees a weight only if all its predecessors have rejected it. An abstract implementation to accommodate this solution generalizes from one weight insertion to a sequence of weights insertion and uses `UpdateWeights`, an abstraction which manages the sequence of weights:

```
B2. BinPack(Bins,Result)
    begin
        repeat
            UpdateWeights(Weights)
            InsertBin(Bins,Weights)
            if FailInsert then exit
    end
```

Program B1 is a "specialization" of B2 if the sequence of weights is constrained to a unit length.

To accommodate algorithms which consider only a subset of all the bins at any given time, we introduce the following abstract implementation:

```
B3. BinPack(Bins,Result)
    begin
        repeat
            UpdateWeights(Weights)
            ChooseBinSubsequence(Bins,RestrictedBins)
            InsertBin(RestrictedBins,Weights)
            if FailInsert then exit
    end
```

For the *Candidate set* parallel algorithm (see section 4.2), `ChooseBinSubsequence` will be refined to produce the subset of *active* bins, *i.e* those containing some weights, plus the next available empty bin.

To flesh out various heuristics (*e.g.* First-Fit, Best-Fit), we need to refine `InsertBin`, the function that inserts a sequence of weights in a sequence of bins.

```
InsertBin(Bins,Weight)
    begin
        SelectBins(Bins,Pred(Weight),GoodBins)
        If EmptyGoodBins then exit
        ChooseBin(GoodBins,Bin,Weight)
        Putweight(Weights,Bin)
    end
```

In the above abstraction, a refinement of B1's `InsertBin`, the function `SelectBins` selects GoodBins, a subset of all the bins that satisfy the predicate `Pred`. In the case of First-Fit, `Pred` chooses all the bins which can accept the current weight, whereas in the case of Best-Fit only the bins that can *minimally* accept the weight are preferred. Finally, the bin selected for insertion is chosen by the function `ChooseBin`.

Next, we use program models B1-B3 in course of developing implementations for the CM ([9]), a finely grained, massively parallel system.

4 First-Fit Bin Packing Simulation

The inherently sequential nature of the bin packing simulation is evident from the observation that: "To take any actions or decisions on behalf of a request, all the previous requests should have been processed". Since at least one step would be required to be able to service a request, and the steps for the N requests are necessarily sequential, it follows that a lower bound on any deterministic algorithm for bin packing is $o(N)$. This lower bound holds for any underlying computational model (SISD, SIMD, or MIMD). To achieve this bound, the amount of processing per request should be independent of the size of the problem.

4.1 A Hand-Waving parallel algorithm

The main idea of this algorithm is to make all the bins observe the new request as it is generated, and then each of them (in parallel) decides whether it can accept it. If it can, it remains active. The central control is then used as an arbitrator to decide who will get the generated weight. In terms of program models, we refine B3 by choosing a sequence of weights of length 1 and by making `ChooseBinSubsequence` the identity function.

Initially, we assign an empty bin to each processing element (PE). The PEs are indexed and thus the bins are totally ordered. At each iteration, the front-end generates a new request by drawing from the uniform distribution (a, b) a new weight w. Next, it broadcasts this value to all the bins. Next, in parallel, each bin tests whether it has enough space to hold the new weight. If it does, the bin remains active. Next, the front-end, through a *re-*

duction operation, picks from among the set of active bins the one that comes first in the aforementioned total order (this entails a refinement of the predicate `Pred` in the abstract implementation of `InsertBin`) and informs it to add the newly generated weight to its local contents. This process repeats until a failure is encountered, that is, until a new weight is generated that cannot fit in any bin. Knowing the total number of bins (initially allocated), and by keeping track of the total of all the generated weights, the front-end can now compute the BR function. By repeating the experiment, the average BR as well as the standard deviation of the measurements can be estimated.

The reduction step required to pick up the bin that receives the new weight is implemented by taking the minimum over the indices of all the selected PEs. The cost of this operation is approximately $log_2 r$, where r is the number of active PEs. Unfortunately, this number can be extremely large especially at the beginning when almost all the bins are still empty.

4.2 A Candidate set parallel algorithm

When trying to pack a new weight, we already know that it is going to land in one of the active bins (i.e. non-empty bins) or otherwise it would have to be packed in the next fresh (empty) bin. Thus, there is no point in looking at all the other empty bins (since they cannot be chosen anyway). Hence, to reduce the total number of selected bins we should focus our attention on the set of non-empty bins as well as the first empty bin. It is important to realize that although we have eliminated the "unnecessary" bins, we also added the overhead of keeping track of the "candidates set".

4.3 A Pipelined Request parallel algorithm

Instead of making all the PEs see the newly generated request at the same time, we only allow a PE to see a request if all its predecessors have rejected that request. The rationale here is to make the weight acceptance decision a local one. That is, if a PE decides to accept a weight, this decision is final. An interesting feature of this technique is that it eliminates totally the reduction step required in the aforementioned "Hand-Waving" approaches. This is replaced with a regular communication pattern between the PEs which always takes a constant time.

In terms of program models, we use B3 in which a sequence of weights is inserted in a sequence of bins. Naively, each weight and/or bin can be assigned to a different PE. However, guided by the bin insertion process, a more efficient data mapping technique is used which merges the sequences of bins and weights and assigns a bin/weight per PE.

Initially, each PE is assigned an empty bin and is responsible for one stage of the pipeline which is originally empty. At each cycle of the computation, the central control "injects" a new request (weight) to the pipeline. Next, each PE (in parallel) checks its stage of the pipeline. If a weight is found, the PE checks whether it has enough space to accept it. If so, the "weight" drops from the pipeline into the bin. Otherwise, the weight is passed to the next stage of the pipeline. This process continue until a weight propagates through the whole pipeline without being accepted anywhere. This indicates a failure to accept that request and signals the termination of the simulation. It is important to note, however, that the state of the bins at this point cannot be used to measure the BR since the PEs might have accepted requests that were generated after the failed request. This can be remedied by propagating through the pipe, along with each request, the cumulative total weight up to that request. Therefore, when a request fails, we can get the total of

the previously generated requests, which have been necessarily accepted (since the request in hand is the first one to fail).

4.4 Comparisons

All the three approaches presented in the previous section require a constant number of steps for each new request. Thus we might expect the required run-time for each to be of the form $k.N$, where N is the number of requests and k is a constant that depends on the number of steps per request as well as the number of cycles required to execute each of these steps. The Pipelined-Requests approach requires a *deterministic* number of cycles to process each weight, wherea both Hand-Waving and Candidate set approaches use a reduction operation which uses the router and hence might use an *undeterministic* (although bounded) number of cycles. The Candidate Set approach aims at reducing the cycles spent in reduction, by restricting itself to a smaller set of candidates.

When we first looked at the Pipelined Request algorithm, we were convinced that it should outperform the other techniques, since it completely eliminates the need for any reductions. However, despite its "apparent" elegance, this technique proved to be very inefficient. The main reason being the inability to predict (without walking through the pipe) whether a generated weight will be accepted. This resulted in delaying the decision to stop the simulation and compute the BR for n iterations. Another weakness is that at the start of the simulation, the whole pipe is empty, and therefore, no useful work is being done by the majority of the PEs. Note that this is also true for Hand-Waving. However, the amount of work done by the PEs is far less.

Our attention was thus focussed on comparing the Hand-Waving and Candidate set approaches. This comparison led to some interesting conclusions. First we found that the Candidate set approach outperformed the Hand-Waving approach only

when the size of the problem was extremely large – over about 512,000 bins. The Hand-Waving approach was decisively better for sizes up to 64,000. The performance was comparable for sizes around 128,000. This behavior is expected, and can be explained as follows. In the Candidate set approach, the price we pay to reduce the number of selected PEs before doing a reduction is *constant*. For smaller problem sizes, this constant overhead actually becomes a burden, since the amount of contention in the router is not that large. Moreover, as we have explained earlier, the router cycle time is far less than the PEs cycle time [8] – thus even with a mediocre router performance, the performance is still comparable. However, with sizes over 512,000 bins, it seems that the constant overhead of the Candidate set begins to pay-off.

5 Simulation Results

In the following sections we briefly describe the different experiments we have done. For detailed results we refer the interested reader to [2].

5.1 First Fit performance

We have simulated the First Fit bin packing process for 128,000 bins for different values of a and b. Figures 1 and 2 show the BR function for $a = 0$ and $a = 0.2$, respectively. These results confirm the anomalies reported in [7]. The complete *BR surface* we obtained is shown in Figure 3. The dominant feature of this surface is a cascade of canopies which exhibit a preponderance of "holes" of size $1/k$ for $k = 2 \ldots \infty$. The "worst" case is in the $1/2$ canopy. There is also a $1/3$ canopy caused, for instance, by values in the range $(1/3, 1/3 + \epsilon)$. The height of canopy $1/k$ is bounded by $k/(k-1)$. Another feature of the BR surface is an oscillation caused by the introduction of new bins. Each new bin

Figure 1: First Fit BR for $a = 0.0$ and $a < b < 1$ (128,000 bins)

suddenly increases the value of BR, by as much as $1/n$. Several values may then be inserted before the next bin is allocated. The consequence is that sampling after N weights are packed is noisy by approximately $1/n$. On the other hand getting smooth data by sampling just before a new bin is allocated causes produces optimistic results.

We have also studied the convergence of the BR function as the number of bins increases. The maximum number of bins we tried is 512,000 and the results were obtained at 512 different points. For the $(0, 1)$ interval, our results show a general trend towards a BR of 1, thus supporting the thesis that First Fit bin-packing is optimal for the interval $(0, 1)$. We repeated the same experiment for the interval $(0.0, 0.85)$ considered by McGeoch and Tygar in [7]). The results we obtained confirmed their conjecture that the BR function approaches a constant value greater than 1 (we found it to be about 1.0166). This means that for

Figure 2: First Fit BR for $a = 0.2$ and $a < b < 1$ (128,000 bins)

Figure 3: First Fit BR surface for $0 < a < b < 1$ (128,000 bins)

this distribution, First Fit is non-optimal. We repeated the experiment for different values of b around 0.8. The results we obtained suggest that First Fit is non-optimal in a region rather than a unique point. The worst performance (according to our experiments) is around $b = 0.79$.

5.2 K-delayed Best Fit

We have modified the aforementioned First Fit algorithms to simulate the Best Fit heuristic. Results of the simulation showed that the performance of Best Fit was no better than that of First Fit.

We have also considered a new packing heuristic, which we termed *K-delayed Best-Fit*. Using this heuristic, requests are not necessarily serviced in order. However, once packed a weight cannot be removed. The idea of this heuristic is to maintain a fixed-size buffer, where new requests are kept until serviced. Whenever the number of pending requests reaches the size of the buffer, we select one of these requests and put it into one of the bins. The request we select is the one that achieves the best "Best-Fit". The Best Fit heuristic is a special case of the "K-delayed Best-Fit", where $K = 0$.

Initial experimentation has shown that for relatively small buffer sizes (4-10), this heuristic outperforms both First Fit and Best Fit. The optimum buffer size to be selected is an interesting point to be addressed. We have noticed from our preliminary experiments that increasing the buffer size from 0 to 4 resulted in a noticeable performance gain. A further increase of the buffer size to 10 resulted in even better performance. Buffer sizes larger that 10 did not seem to provide any better responses. This suggests the existence of a certain *threshold*. We suspect that this threshold relates to the shape of the candidate bins filter (see [7]). Further experiments are needed to support this claim.

6 Conclusion

We have considered the issue of developing implementations of the bin packing problem for different platforms, but especially for massively parallel systems. The approached consisted in a sequence of refinements rooted in a space of abstract implementations. In the quest of efficiency, we had to make choices both in terms of process mapping and data mapping. In the final analysis, we found that our serial intuition did not always serve us well in parallel contexts. Our own transition from serial to parallel thinkers is far from complete. Thus it is important to provide means for explicit control over both processes (see [6])

References

[1]Batcher, "Design of Massively Parallel Processor", *IEEE transaction on computers,* C-29, September 1980.

[2]A. Bestavros, W. McKeeman, "Parallel bin packing using first fit and k-delayed best-fit heuristics." *TR-16-88, CS Dept., Harvard University.* August 1988.

[3]T. Cheatham, D. Stefanescu, "Knowledge-based software development", *Unpublished notes,* CS Dept., Harvard University.

[4]Bouknight, Denenberg, McIntyre, Randall, Sameh, Slotnick, "The ILLIAC IV system", *Proc. IEEE,* 60, 4, April 1972.

[5]W. Hillis, G. Steele, "Data Parallel Algorithms",*CACM,* Dec 1986.

[6]V. Kathail, D. Stefanescu, "A Data Mapping Parallel Language", *TR-21-89,* CS Dept., Harvard University, December 1989.

[7]C. McGeoch, J. Tygar, "When are Best Fit and First Fit Optimal?", *Technical Report,* CS Dept., Carnegie Mellon, October 1987. Also in *1988 SIAM Conference of Discrete Mathematics.*

[8]B. O'Farell, "Instruction timing on the Connection Machine", *Parallel Computing NEWS 1(1),* NPAC, Syracuse Univ., April 1988.

[9]*The Connection Machine Parallel Instruction Set – Ver 5.2,* Thinking Machines Corporation.