AN SIMD PARALLEL \( \varepsilon \)-APPROXIMATION SCHEME FOR 0/1 KNAPSACK

Thomas E. Gerasch
SPARTA, Inc.
7926 Jones Branch Drive
Suite 1070
McLean, Virginia 22102

ABSTRACT

A parallel version of a well-known \( \varepsilon \)-approximation scheme for 0/1 knapsack problems is presented. The model of computation for the parallelization is a shared memory machine in which processors have exclusive read, exclusive write access to memory (an EREW PRAM). The scheme separates the knapsack items into two sets, one of which is used in a dynamic programming-based optimization procedure, and the other of which is used in a greedy selection process. A dominance relation exists for the knapsack problem which is used to limit the growth of feasible solutions during the dynamic programming procedure. The dominance relation permits a simple representation of the feasible solutions which aids in the parallelization of the dynamic programming procedure across all feasible solutions in parallel during the process of considering a new item. The algorithm uses \( \max(n,32/\varepsilon) \) processors and takes \( O(n) \) time. For moderate values of \( \varepsilon \) and values of \( n \) which are quite large (tens of thousands of items), the algorithm is realizable on currently available, massively parallel computer systems, such as the Connection Machine System.

Keywords: knapsack, approximation, SIMD, shared memory, dynamic programming.

1. INTRODUCTION

The 0/1 integer knapsack problem is defined by a set of \( n \) objects having positive integer profits \( p_j \) and weights \( w_j \), and a positive integer knapsack capacity \( M \). The problem is concerned with maximizing the profit \( P = \sum p_j x_j \) subject to a weight constraint \( \sum w_j x_j \leq M \), where \( x_j \in \{0,1\} \) for \( j = 1,\ldots,n \). An instance of the knapsack problem consists of an assignment of values for the profits, weights and the capacity. A feasible solution to a problem instance is a set of \( x_j \) values which satisfy the capacity constraint by indicating whether an item is to be included in the knapsack. This well-known optimization problem is NP-Complete [GT79], and consequently, considerable interest has been generated in developing algorithms which obtain approximate solutions [LE79; HS78; IK75; SS75].

An \( \varepsilon \)-approximate solution \( 0 < \varepsilon < 1 \) for an instance of the 0/1 knapsack problem is a feasible solution whose profit \( P \) satisfies \( P^* - \varepsilon P \leq \varepsilon P^* \), where \( P^* \) is the maximum profit attainable for the problem instance. An approximation scheme for the knapsack problem is one which takes an \( \varepsilon \) as a parameter and for each instance of the knapsack problem finds an \( \varepsilon \)-approximate solution. Ibarra and Kim [IK75] presented an approximation scheme for the knapsack problem which has both time and space complexities which are polynomial in \( n \) and \( 1/\varepsilon \), i.e., which is a fully-polynomial approximation scheme [HS78]. Lawler [LE79] presented modifications to the ideas of Ibarra and Kim that improved the time and space bounds.

There has been a growing interest in parallel computation and in the parallelization of optimization techniques and algorithms. This interest has extended to the adaptation of knapsack approximation algorithms to parallel models of computation [GRK86; GW88; LS87]. These parallelizations have been for MIMD [LS87] as well as SIMD [GRK86; GW88] models of computation.

The algorithm to be presented here, like that in [GRK86], is an SIMD parallelization of the \( \varepsilon \)-approximation scheme of Ibarra and Kim. Although both algorithms are SIMD parallelizations of the same basic strategy, their approaches and actual parallel models of computations differ. The algorithm in [GRK86] is a recursive implementation of a dynamic programming technique applied to a scaled profit version of the entire problem instance. The algorithm presented here is a more direct implementation of the original dominance-based dynamic programming approach of Ibarra and Kim, using scaling of the profit values as well as an item separation technique. The item separation partitions the knapsack items into two disjoint sets of relatively large and small items, with set membership determined relative to a suitably chosen threshold value. The dominance-based dynamic programming technique is applied to a scaled problem involving the large items. Feasible solutions from the large item optimization are augmented with small items, with the small items selected to augment a feasible solution using a greedy approach.

The SIMD model which is used assumes that processors have exclusive-read, exclusive-write access to a common memory (i.e., an EREW PRAM model). The model used in [GRK86] allows concurrent reading of memory by processors, but exclusive writing, i.e., a CREW PRAM model.) Processors are assigned to manipulate data in a data-parallel fashion using one processor per array index. To ensure that data is accessed in an EREW fashion, processors which are actively participating in a computation step will reference the data in a uniform addressing fashion. For example, each active processor \( i \) will reference data item \( i+k \), where the offset \( k \) is the same for all processors.

The analysis of the algorithm will be similar to that typically given for serial algorithms in that arithmetic operations, comparisons, and memory references are counted as unit time operations. However, the same operation performed in parallel by many processors will only be counted as a single (parallel) instruction execution. The assumption that all of the operations mentioned take unit time is not strictly true in actual machines, but has been adopted to simplify the analysis of algorithms. It has recently been argued by Blelloch [1987] that since memory references in the shared memory models are taken as unit time operations, other useful primitive operations should also be allowed as unit time operations. Scan, or parallel prefix operations [KRS85; BG87], are among the candidates as useful operations, since they can be implemented with less circuit depth and in comparable circuit area than a general shared memory circuit [FF83; LT84]. A scan operator takes an associative operator \( @ \) and a sequence of elements \( a_1,a_2,\ldots,a_n \), and returns the...
sequence l_{a},(a_{1}a_{2})\ldots,(a_{r}a_{2}\ldots a_{t})a_{s}$. The scan operations used in the algorithms will be plus\_scan, which uses addition to obtain the sequence of partial sums. copy\_scan to replicate a data item a specified number of times (a plus\_scan with all items but the first contributing 0), and max\_scan to find the maximum of n items. A scan operation can also be run in parallel on subsequences of an array, and will be called a segmented scan operation.

Scan operations are logarithmic time operations in the usual EREW PRAM model. Taking the scan operations to be unit time subsequence of an array, and will be called the scan model. The algorithm will be analyzed in both the EREW and scan models.

The EREW PRAM, and scan model, knapsack approximation scheme to be presented requires $O(n)$ time and $\max(n, 326^{1/3})$ processors. This time requirement is greater than that of the CREW PRAM algorithm of [GRK86], which is $O(\log n \times \log \log(1/e))$. However, this $O(n)$ time requirement is not unreasonable. A more interesting measure is the number of processors required by these two algorithms. The CREW model algorithm requires $2^{2+\log_25}$ processors, in its worst case. For $n=1000$ and $e=1/10$, the EREW and scan algorithms require 31920 processors, while the CREW algorithm may require up to $10^{12}$ processors. For $e=1/10$ and a number of processors equal to $2^{16}$, the CREW algorithm of [GRK86] can only be guaranteed to handle instances of knapsack problems with fewer than 350 items, while the EREW and scan algorithms could handle problem instances of size $2^{16}$. The processor requirement for the algorithm of [GRK86] can quickly exceed the capability of current technology. However, the processor requirement for the algorithm presented here is within the realm of current technology, and allows for problems of considerable size to be realistically attacked, assuming moderate values of $e$.

The Connection Machine System of Thinking Machines Corporation has up to $2^{16}$ processors, and supports virtual processors into the millions. Although the memory in the Connection Machine is local to each processor, one can program it in the EREW memory reference paradigm by using interprocessor fetch and send operations to simulate the shared memory read and write operations, and by designing algorithms to not have collisions at any processor during the memory reference operations. It was with the Connection Machine in mind that the EREW PRAM parallelization of the $\epsilon$-approximation scheme was initially designed.

2. $\epsilon$-APPROXIMATION SCHEME

The approximation algorithm to be parallelized is the $\epsilon$-approximation scheme of Ibarra and Kim [IK75], using the scaling refinements due to Lawler [LE79]. The $\epsilon$-approximate solution to a knapsack problem instance is obtained by first separating the items into two sets of relatively large items and relatively small items. The item separation is performed using scaling and threshold factors which ensure that one-half of the relative approximate error is allocated to each of the two calculation stages. These stages correspond to computations involving each of the two sets of items [LE79]. The threshold and scale factors are determined by using an estimate $P_0$ of the optimal profit value for the knapsack instance. This estimate satisfies $P_0 \leq P^* \leq 2P_0$ where $P^*$ denotes the optimal profit value for the problem instance.

The large item computation uses a dynamic programming approach to find an optimal solution to a knapsack instance that consists of the items with scaled large profit values and has capacity equal to the original capacity. To restrict the growth of the number of feasible solutions during the large item computation, the following dominance relation is used. Let $S_1$ and $S_2$ be two feasible solutions (i.e. sets of elements whose weights do not exceed the knapsack capacity). The feasible solution $S_2$ dominates $S_1$, written $S_2 \geq S_1$ if $P(S_1) \leq P(S_2)$ and $W(S_1) \leq W(S_2)$, where $P(S) = \sum_{i \in S} P_i$ and $W(S) = \sum_{i \in S} w_i$. It should be noted that if $S_2 \geq S_1$ and if $S_1$ is a set of indices disjoint from both $S_1$ and $S_2$, then $S_1 \cup S_2 \geq S_2$. During the large item computation, a sequence of feasible solutions is maintained whose profit and weight sums are in increasing order and for which no solution is dominated by any other in the sequence. As each item is considered, the dominance rule is applied to preserve the properties of the sequence of feasible solutions. The last solution in the sequence is actually the optimal solution to the scaled large item problem instance. At the end of the large item computation, the entire sequence of feasible solutions is retained, not just the last in the sequence.

Each of the feasible solutions from the large item computation has its remaining capacity augmented by small items. This augmentation of the solutions is done using a greedy approach. The $\epsilon$-approximate solution to the problem instance is the one with greatest profit value after the large and small item computation stages.

The algorithm for the $\epsilon$-approximation scheme is summarized in the following steps:

1.) Find $P_0$ such that $P_0 \leq \epsilon \leq 2P_0$, and set the scale factor $K = (2/\epsilon)^2P_0$, and the threshold factor $T = (2/\epsilon)^2P_0$.

2.) (item separation) Separate the items into large and small sets, the large items being those for which $P_i \geq \epsilon$. Arrange the items so that the large item set precedes the small item set. For the large items set, $q_i = [P_i/K]$, the scaled profit values.

3.) (Select a minimal set of large items to consider) Arrange the large item set in nonincreasing order of $q_i$ and within each $q_i$ group in nondecreasing weight order. Select the first $(2\epsilon^2/q_i)$ items of each $q_i$ group as candidates.

4.) (large item optimization) Perform the dominance-based optimization of Ibarra and Kim on the minimal set of large item candidates, producing a sequence of feasible solutions whose profit and weight sums are nondecreasing (and consequently for which no solution dominates another).

5.) (Augment feasible solutions with small items) Arrange the small items in nonincreasing order of $p_i/q_i$ ratios and add a small item to a feasible solution if there is sufficient capacity remaining the feasible solution. Select the feasible solution with the greatest profit value.

Theorem 1. The algorithm outlined is an $\epsilon$-approximation scheme for the 0/1 knapsack problem.

That the process outlined above is an $\epsilon$-approximation scheme was shown by Ibarra and Kim [IK75]. The particular scaling and threshold values used in step 1) are due Lawler [LE79]. Additional details concerning the steps of the algorithm will be presented in the following section, where the parallelization strategy is presented.

3. PARALLELIZATION OF THE $\epsilon$-APPROXIMATION SCHEME

The first step in the parallel algorithm can be accomplished by sorting the items into nonincreasing profit/weight ratios so that $p_1 \geq \cdots \geq p_m \geq C$ and next determining the largest $m$ such that $p_1 \cdots p_m \leq C$ but $p_1 \cdots p_m + p_{m+1} > C$. $P_0$ is then taken to be $\max(p_1 + \cdots + p_m, \max(p_i), \epsilon C/n)$ [LE79]. An EREW sorting algorithm using $n$ processors, such as an adaptation of the bitonic merge sort [QM87], will be assumed for this and for subsequent sorting steps. The time to sort $n$ items using $n$ processors in the EREW model is taken to be $O(\log n)$. The CREW model algorithm from [GRK86] uses a time estimate of $O(\log n)$ for sorting $n$ items using $n$ processors, and is based on the algorithm in [AKS83]. As pointed out by Leighton [LT84], the limit obtained in [AKS83] is an asymptotic limit and the constant of proportionality is so large
that this sorting strategy becomes infeasible from a technological standpoint. For this reason, the more realistic time estimate for sorting of $O(\log n)$ using $n$ processors is used here.

In the scan model of computation, a sorting algorithm can be used that is a parallel version of the radix exchange sort [KD73]. If the sort keys are $m$ bits long and $m$ is of the same relative order of magnitude as $\log n$, then a radix exchange sort can be implemented in the scan model which uses $O(m) = O(\log n)$ operations [BC87]. In this case, the constant of proportionality is reasonably small.

Further, this sorting strategy is also applicable when floating point representations of numbers is used. This parallel radix sort is the sorting algorithm assumed in the scan model analysis.

The rearrangement of the items in step 2, the item separation phase, can be accomplished by first using $n$ processors to enumerate the items in each set. This requires parallel prefix summation (plus_scan) operations in which each of the items in the set being enumerated contributes a 1 while the other set's items each contribute a 0. The rearrangement can then be easily accomplished by adding the number of elements in one set to the other's enumeration values and using these values to indicate the new positions for the items. The enumeration steps require $O(\log n)$ time in both models of computation.

The large items are those whose profit values satisfy $p > T$. Consequently, the minimum scaled profit value, and the minimum scaled profit of any feasible solution in step 4, satisfies $q_1 = \max_{i \in \mathbb{K}} \left\lceil \frac{T}{W_i} \right\rceil \cdot W_i \geq \frac{T}{2}$. The largest possible scaled profit obtainable is determined by $P^* = \max_{i \in \mathbb{K}} \frac{b_i T}{2 L_i}$. Hence, no more than $8L_i^2 q_1$ items with scaled profit value $q_1$ can fit in any feasible, large item solution. This justifies the selection of the large item candidates in step 3.

The rearrangement of the large items for the purpose of selecting a minimal set of large item candidates can be performed by first sorting the large items using the values $(b_i L_i q_1) W_i w_i$ as the sort keys. Here, $W_i$ is the maximum of the weights $w_i$ of the large items. This will arrange the large items into decreasing order of $q_1$, and within each segment of items having the same scaled profit $q_1$, the items will be in nondecreasing order of $w_i$. A segmented plus_scan of $T$ is used to obtain an enumeration of the numbers in the elements of the $q_1$ segments in parallel. Next, a simple comparison of each item's segment index with the number of items allowed to be in the segment is performed to determine the items in each segment which should remain active for the large item optimization. The time requirements for step 3 are $O(\log n + \log n)$ in the EREW model, owing to the sort and the segmented plus_scan operation, and $O(\log n)$ in the scan model.

The feasible solutions in the dominance-based optimization on the large items are indexed by the possible profit values in the scaled problem. These index values are, in addition to $0$, between $2T/e$ and $8L_i^2$. It can be shown that the maximum number of items in any of the scaled large item solutions is bounded above by $4T/e$. Consequently, the feasible solutions in the large item computation can be maintained in $8L_i^2 e$-size arrays with each section being of length $4T/e$. A group of similarly indexed array sections is used to maintain the list of indices of the large items placed into the feasible solution. In addition to the indices of large items placed into a solution, there must also be a flag to indicate which is the next available position for an index, and the capacity remaining for a particular feasible solution. Each feasible solution will be managed by 4/e processors. To insure EREW memory access, each processor assigned to manage an array section (i.e. a scaled feasible solution) will have its own copy of the capacity remaining in the knapsack for its feasible solution, as well as its own copy of the actual, non-scaled profit for its feasible solution.

The general form of the large item computation is as follows:

- For each large item candidate, let $i'$ denote the item's index.
- For all $s$, let $s = (s+q_1')/q_1$ and $F(s)$ to $F(s+q_1)$ if $F(s+q_1') > W(s)$, then
- For all $s$, let $s = (s+q_1')/q_1$ and $F(s+q_1)$, add index $i'$ to $F(s+q_1')$, and add $W_i$ to $W(s+q_1')$.

This algorithm is a direct parallelization of the dominance-based optimization originally given by Ibarra and Kim [IK75]. For convenience, a feasible solution is denoted by $F(s)$ and its weight by $W(s)$, where the scaled profit of the feasible solution $s$ is its index. Any test involving a feasible solution indexed by $s$ is performed by all $4/e$ processors which manage the feasible solution, and is done in parallel. This is possible since the necessary flags and weight sums are replicated so that each processor has exclusive access to its own copy. The key to the SIMD parallelization is this data replication together with the fact that the feasible solutions can be checked for the additional of the new item candidates in a scan step in parallel.

In a feasible solution and the resulting new solution dominates another in the list, the dominated solution is replaced in a single step by using $4/e$ processors. Since any feasible solution can be done in parallel and in an EREW memory reference fashion.

The number of processors needed for the large item computation is $4T(8L_i^2 - 2T/e)$. There are $\max\{n, 8L_i^2 - 2T/e\}$ large item candidates, and hence this determines the time requirement for step 4.

The implementation of the small item augmentation of the feasible solutions in step 5 of the approximation scheme is carried out in two stages. In the first stage, the process for obtaining the final collection of feasible solutions does not attempt to remember the indices of any small items which would have fit into the remaining capacities of the solutions. In this way, all of the feasible solutions can test the same small item simultaneously for inclusion, by having the profit and weight of the item broadcast to all feasible solutions in parallel. After the feasible solution with the largest profit over both the large and small items is selected, a greedy algorithm is again performed on the small items. However, on this second execution of the greedy algorithm, the indices of the small items which augment the large item solution are saved.

Step 5 iterates over the small items. It can be seen that steps 4 and 5 together can iterate over all of the items, and hence the time requirement for these two steps together is $O(n)$. The following theorem summarizes the time and processor requirements for the parallel E-approximation scheme.

Theorem 2. The parallel E-approximation scheme takes $O(n + \log n)$ time in the EREW model and $O(n + \log n)$ time in the scan model. The scheme requires $\max\{n, (4T)(8L_i^2 - 2T/e)\}$ processors in both parallel models of computation.

4. CONCLUSIONS

An SIMD algorithm has been presented which is a parallel E-approximation scheme for the 0/1 knapsack problem. The algorithm implements a dominance-based dynamic programming technique in an exclusive read, exclusive write independent memory model of SIMD programming. The number of processors needed by the algorithm is within current technological bounds for moderate values of $e$ and for problem instances into the tens of thousands of items.
REFERENCES


