CS267 ASSIGNMENT 3: PARALLELIZE KNAPSACK

CHUN MING CHIN, CHRIS MELGAARD, VIRAJ KULKARNI

1. Comparing Answers

To ensure we get the correct answer, like the provided serial and fixed parallel codes, we save the outputs of the used, weight, value and total arrays and compare the results with the original serial version. The save function is obtained by modifying source code from homework 2.

In order to compare answers accurately, we changed the srand function to have a constant seed (i.e. stand48(1000) so we can use our serial results to compare with our ups results. In addition, for the initialization stage in the knapsack.upc implementation, we use a for loop on MYTHREAD==1 instead of the upc_forall loop to iterate on the lrand48() operations for the weight and value arrays.

We do blocking operations on rows of T (i.e. operating on contiguous blocks of memory) so as to increase cache hits. This would minimize communication time, hence making the code run faster.

Next, we also considered pipelining our computations, because an object, j, is only dependent on its previous object j-1.

2. Timing

The time required for serial implementation is 0.031931 The time required for upc implementation is 11.4287, 10.9262 Time required is reduced to 10.2737 when 2 upc_forall loops are replaced by for loops

When the number of threads is 4 for a capacity of 1000, the block size is 250 When the number of threads is 8 for a capacity of 1000, the block size is 125 When the number of threads is 16 for a capacity of 1000, the block size is 63 When the number of threads is 32 for a capacity of 1000, the block size is 32

3. Generalized

We make the knapsack capacity an integer multiple of the number of threads so that the total array can be segmented equally into separate blocks, where each thread will operate on separate blocks.

4. Solution

We check the solution by breaking the if statement: $if(best_value! = best_value_serial||best_value! = total_value||total_weight > capacity)$ into 3 separate if statements:

$if(best_{v}alue! = best_{v}alue_{s}erial)printf(''WRONGSOLUTION : Bestvalfrmparallelnotequaltobestva$
$if(best_{v}alue! = total_{v}alue)printf(''WRONGSOLUTION : Bestvalnotequaltototalval'')$
$if(total_{w}eight > padCapacity)printf(''WRONGSOLUTION : Totalweightnotlargerthanpaddedcapaci$
It is reasonable for the 2nd if statement to return true, since for the general padded capacity case, we may not be taking the last element in the T[i,j] table, hence we may miss items that are asserted in the used array. These are not included in the final answer because added such items will cause the weight to be larger than the actual capacity bound, even thought adding their weights is still within the padded capacity bound.

5. Further Work

1. Only transfer data elements that would be used in the next iteration with memput and memget. 2. Ensure you do not transfer elements from the global total array to the local total array, for those elements that are already updated by the current thread locally. 3. Use SSE intrinsics to assign the values in local arrays using vectorization. 4. Can we use the restrict keyword to optimize the code? 5. Run the compiler with the optimization flag -o3