Aim of the assignment

The aim of this assignment is to implement an NP-complete problem, the knapsack problem on a parallel computer. This problem requires dynamic load balancing, which is the focus in this assignment. More specifically, the following should be implemented and evaluated:

1) Centralized scheduler
2) Decentralized scheduler
   • Distributed termination

Thus, it is not purpose to find the ‘best’ sequential method for solving this problem, instead a ‘branch-and-bound method shall be used.

Problem Description

The knapsack problem is a discrete optimization problem. This problem arises whenever there is a resource allocation problem with financial constraints. For example, given that you have a fixed budget, how do you select what things you should buy? Assume that everything has a cost and value. We seek to provide the most value for a given budget.

The term knapsack problem invokes the image of a backpacker who is constrained by a fixed-size knapsack and so must fill it only with the most useful items. The classical knapsack problem assumes that each item must be put entirely in the knapsack or not included at all. It is this 0/1 property that makes the knapsack problem difficult. Suppose the backpacker must choose among \( p \) different items. Each item \( i \) \((i = 1, \ldots, p)\) has a weight \( w_i \) (in pounds) and a utility \( u_i \) to the backpacker. The objective is to maximize the utility of the backpacker’s trip subject to the weight limitation that the backpacker can carry no more than \( W \) pounds. The following is an integer formulation of this problem.

\[
\max : \sum_{i=1}^{p} u_i x_i \\
\text{Subject to:}
\sum_{i=1}^{p} w_i x_i \leq W \\
\text{where } x_i \in \{0,1\} \text{ for all } i.
\]

This is an NP-complete problem, that is, it cannot be solved in polynomial time.
Branch and Bound

Branch and bound is a general well known method to solve such NP-complete problems. The method is based on an exhaustive search in the solution space. In the knapsack problem case, the idea is the following: think of a binary tree with a depth that corresponds to the number items. At each level of the tree there is a choice of including the item or not. This is carried out from the root to the leaf. It is easy to see that problem grows exponentially. A sequential algorithm:

1. At node_i (WORKER)
   1.1. Compute the value in the knapsack, CAND_VAL and upper bound UB.
   1.2. If CAND_VAL is higher than the current best value (GLOBAL_OPT), then update GLOBAL_OPT.
   1.3. Bound: If UB is lower than GLOBAL_OPT then terminate the branch.
   1.4. ELSE branch into new subproblems (create child nodes)
   1.5. Pick the next subproblem, if none exist, stop.

Upper bound and how to order the items

A crucial component is how to bound, that is, how to compute value that determines whether a path not should evaluated further or not. In order to this, we need to order the items in specific order. This is done by order the items by its ratio value/weight. The highest ratio is on level 0, and the lowest value is at the bottom. The UB can then be computed as

\[
UB = (\text{Weight remaining in knapsack}) \times (\text{ratio value/weight}) + \text{CAND\_VAL}
\]

Sequential implementation

The easiest way of handling the sub problems is use a queue, and thus, a binary tree need not be used. First the queue is empty, and since CAND\_VAL does not exist, step 1.4 is executed, which means that two sub problems are generated in placed into the queue. At next step, a new sub problem can be enqueued, and so on.
Assignments - Parallel implementation

Assignment 1

Implement a centralized scheduler which holds the entire queue. This process should be placed on one core. All other cores deal with examine the binary tree (worker), which is described above.

The scheduler will be a bottle-neck, and your task is to verify that. Moreover, experiment with small local queues for each worker, to decrease the communication with scheduler.

You should present speed-ups as results.

Assignment 2

Implement a decentralized scheduler solution, where each core both acts as a scheduler and a worker. The scheduler can be sender-initiated or receive-initiated, and the main issue here is to develop an algorithm that is as efficient as possible.

A distributed termination algorithm must also be implemented. A ring-based solution is recommended.

Programming Languages and Parallel Computers

The implementation should be done using Fortran/C/C++ and use MPI to communicate between processes.

When you want to run your program (including timings), submit your jobs to the batch system of akka at HPC2N. Please take a look at the support pages at HPC2N to see how to compile and run programs through the batch system. Make sure to start your programs from the kfs file system. You may consider the cluster as having the logical network you prefer.

In the submit file, use 5DV011-VT11 as account (project number), or else you will have trouble getting your jobs through the queue.

In the submit files given as examples you should normally remove the option -pernode to mpiexec.