Parallel and Multicore Computing

Project 1B – Knapsack Filling

Boyu Zhou (zhobz), Jinxin Hu(jinxinh)

Task Objective

The 0-1 Knapsack problem is a classic NP-complete problem. There are various design thoughts and solutions to solve this problem (Quan et al., 2012). In this work, we used three popular parallel technologies – MPI technology and MPI + OpenMP hybrid programming model based on the sequential dynamic programming algorithm, in order to achieve better performance.

Problem Solution

The sequential dynamic programming algorithm is designed to consider all subsets of items and calculate the total weight and value of all subsets, and then consider the only subsets whose total weight is small than W and pick the maximum value subset from all considered subsets (Li et al., 2015). In this situation, when MPI technology is chosen, communication solutions between different processes are essential. We designed different MPI_Send and MPI_Recv in different circumstances.

Implementation

During the experiment, MPI is chosen to achieve better performance for the dynamic programming algorithm. First of all, by using MPI_Bcast, the data variables C, n, v, w would be sent to all other processes from process 0. Additionally, MPI_Barrier was used to synchronize processes (List 1) (MPI Tutorials, 2019).

```
    MPI_Bcast (&C, 1, MPI_LONG, 0,
MPI_COMM_WORLD);
    MPI_Bcast (&n, 1, MPI_INT, 0,
MPI_COMM_WORLD);
    MPI_Bcast (v, n, MPI_LONG, 0,
MPI_COMM_WORLD);
    MPI_Bcast (w, n, MPI_LONG, 0,
MPI_COMM_WORLD);
    MPI_Barrier (MPI_COMM_WORLD);
```

List 1: Implementation of Bcast and Barrier

Secondly, we used for-loops to compared values and pick the maximum value. Especially, row 0 is initialized to zero, row 1 only sends values to row 2 and does not need to receive messages, while row n does not need to send messages. Therefore, we considered row 0, 1, n and other rows separately

For row 2 to n-1, MPI_Send and MPI_Recv were used in different situations. Firstly, we judged the size of wt and w[i-1], in order to determine which data do not need communication. If wt < w[i-1], we determined wt data do not need communication, which comes from the last row. The code list is shown below:

```
K[i][wt] = K[i-1][wt];
```

List 2: wt < w[i-1]

Furthermore, k[i][0] to k[i][C-wi] in row i need to send messages, while k[i][wi] to k[i][C] in row i+1 need to receive messages, in order to calculate the local max value (localmax is defined to store the local max value of a process). The code list is shown below:

```
localmax = max(K[i][wt], localmax);
//MPI_send
if(wt >=0 && wt <= C-w[i] && w[i]%size != 0 && i != n)
{
    MPI_Send (&K[i][wt], 1, MPI_LONG, (wt + w[i] -
1)%size, wt + i * C, MPI_COMM_WORLD);
}
//MPI_receive
if(w[i-1]%size != 0 && wt != w[i-1] && i != 1)
{
    MPI_Status status;
    MPI_Recv (&K[i-1][wt - w[i-1]], 1, MPI_LONG, (wt -
w[i-1]- 1)%size, wt - w[i-1] + (i-1) * C,
MPI_COMM_WORLD, &status);
}
```

List 2: Implementation of send and receive

In the end, MPI_Reduce is used to pick the max global value from all local max value. The code list is shown below:

```
MPI_Barrier (MPI_COMM_WORLD);
long int globalmax;
MPI_Reduce (&localmax, &globalmax, 1, MPI_LONG,
MPI_MAX, 0, MPI_COMM_WORLD);
```

List 2: Implementation of Reduce

Results and Analysis

Figure 1 illustrates the running times (us) of the three different technology solutions respectively:

Solutions	Running Time (us)
MPI technology	73231
MPI + OpenMP hybrid programming model	67290
Sequential dynamic programming algorithm	6756

Figure1: Running Time (us) of three solutions

For the above experimental results, we ran the test case provided several times and took the minimum run-time over the runs. It can be seen that the sequential solution has the smallest time, the most likely cause of which is that the communications between different processes cost much time. Furthermore, the test case only has 44276 knapsack occupancy, which is too small if dynamic programming algorithm is chosen.

It is clear that the complexity of time of DP algorithm is O(Cn), however, the complexity of time of Brute algorithm is O(2^n). We believe that when knapsack occupancy is large enough, the running time for MPI technology can be less than the sequential technology.

Additionally, it also can be seen that compared the running time of MPI technology and MPI + OpenMP hybrid programming model, MPI + OpenMP hybrid programming model can improve the performance significantly.

Reference

Quan, L., Shen, W., Cui, J., & Geng, D. (2012, May). Application of 0–1 knapsack MPI+ OpenMP hybrid programming algorithm at MH method. In 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (pp. 2452-2456). IEEE.

MPI Tutorials (2019). Retrieved from https://mpitutorial.com/tutorials/

Li, K., Liu, J., Wan, L., Yin, S., & Li, K. (2015). A cost-optimal parallel algorithm for the 0–1 knapsack problem and its performance on multicore CPU and GPU implementations. Parallel Computing, 43, 27-42.