CompSci 590.03: Introduction to Parallel Computing

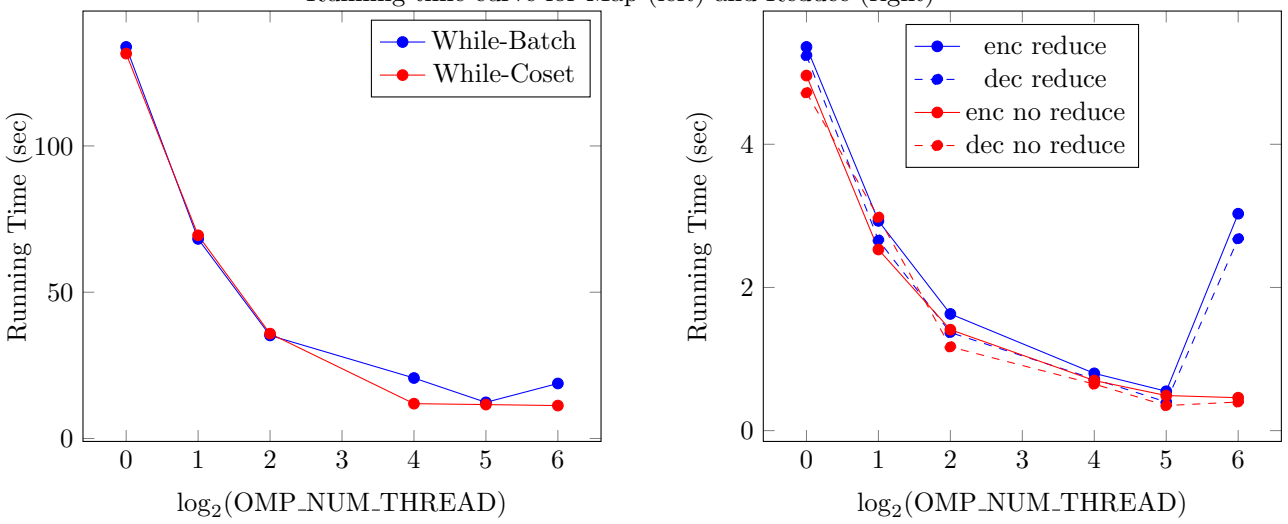
Homework 1 - Map & Reduce (09/08/15)

Lecturer: Alvin R. Lebeck *Scribes: Mengke Lian, Kai Fan, Chaoren Liu*

Contribution distribution:

- Mengke Lian: finish map using OpenMP, Cilk and TBB with the two models discussed below, finish reduce using OpenMP, Cilk and TBB by parallelizing outer loop with/without reducing inner loop.
- Kai Fan: implemented the "map" project by using OpenMP (two methods), Cilk and TBB separately, and the "reduce" project by OpenMP, Cilk and TBB with reduce (may not work as well as the case without reduce). I am also responsible for recording the performance of reduce project.
- Chaoren Liu: Solved the "map" problem using OpenMP method and solved the "reduce" problem with TBB method. In TBB method, I fuse the map and reduce procedures. In the next HW, I will practice CilkPlus and either one of OpenMP and TBB.

Running time curve for Map (left) and Reduce (right)



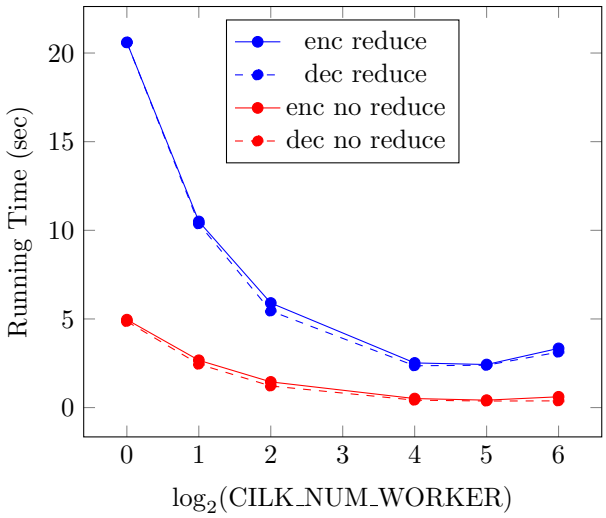
Discussion

For map part, we implemented both While-Batch model and While-Coset model using OpenMP, and the plot shows that when `OMP_NUM_THREADS` is small, approximately the running time halves when `OMP_NUM_THREADS` doubles. However, the slope of each segment becomes flatter, which is caused by overhead. When `OMP_NUM_THREADS` is 16 or greater, the While-Coset model benefits merely from increasing `OMP_NUM_THREADS`, the While-Batch model reaches minimum at `OMP_NUM_THREADS` $= 32$ and runs slower when `OMP_NUM_THREADS` $= 64$.

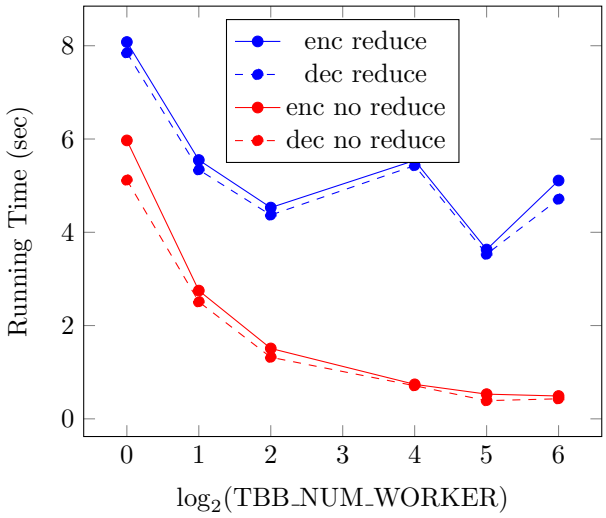
For reduce part, we tried to parallelize the outer loop (length is 1213228) using map pattern and ran the inner loop (length is 1000) using reduce pattern or just serially. For cases `OMP_NUM_THREADS` smaller than 64,

the curves are similar to map part. Compared to 32, he case which inner loops are parallelized by reduce pattern gets worse when `OMP_NUM_THREADS` is 64, while the serial case has no significant change.

In the reduce project, we also tested the performance of cilk_plus implementation by different numbers of workers. The performance without reduce is almost the same as OpenMP; however, the reduced version seems accelerate twice with 16 or 32 threads (This result is coincident with the posted result in piazza).



For the TBB performance on reduce project, the similar result of cilk_plus is shown below. The best reduced performance is 32-thread.



Besides, the case without reduce is always faster than the case with reduce, the reason is that the length of the inner loop is too small and applying reduce puts no gain on the performance.

Two Models to Parallelize While Loop

While-Batch Model

Initialize not found indicator `flag` = `true`;
Initialize temporary indicator `temp_flag`[i] = `true` for `i` = $0, \ldots,$ `BATCH_SIZE` $- 1$;
Initialize temporary result `temp_result`[i] for `i` = $0, \ldots,$ `BATCH_SIZE` $- 1$;
Initialize current index `i_curr` = 0;
while `flag` **do**
  Run the following for-loop parallelly;
  for `i` = `i_curr` : `i_curr` + `BATCH_SIZE` $- 1$ **do**
    [`temp_flag`[i], `temp_result`[i]] = `process`(i);
  end
  Check whether while loop should be stopped in this batch, if so give hit index;
  [`flag`, `i_hit`] = `check`(`temp_flag`);
  `i_curr`+ = `BATCH_SIZE`;
end
`final_result` = `temp_result`[`i_hit`];

In this model, the performance will get worse if it is severely unbalanced in each batch.

While-Coset Model

Initialize not found indicator `flag` = `true`;
Initialize temporary indicator `coset_flag`[i] = `true` for `i` = $0, \ldots,$ `NUM_COSET` $- 1$;
Initialize temporary result `temp_result`[i] for `i` = $0, \ldots,$ `NUM_COSET` $- 1$;
Initialize current index `i_curr`[i] = i for `i` = $0, \ldots,$ `NUM_COSET` $- 1$;
Run the following for-loop parallelly;
for `i` = `i_curr` : `i_curr` + `NUM_COSET` $- 1$ **do**
  while `flag` **do**
    [`temp_flag`[i], `temp_result`[i]] = `process`(i);
    if `coset_flag`[i] == `false` **then**
      `flag` = `true`;
      `final_result` = `temp_result`[i];
    end
  end
end

In this model, the performance will get worse if it is severely unbalanced in each coset.

Notice

While-Batch model and While-Coset model can be seen as parallelize a while loop horizontally and vertically respectively. The two models above is suitable for solving problem with unique solution (only one case can let `flag` to be true). For example, find the integer number equals factorial of 100.

If the solution is not unique, race condition exists and the result may randomly become one of the solutions. For example, find the least common multiple of $13, 15, 17$, without further modification it is possible that the

programming model returns a multiple of $\mathsf{LCM}(\{13, 15, 17\})$ if the batch size is large in While-Batch model or the coset number is large and performance for cosets are unbalanced in While-Coset model.