Project #1 of High Performance Computing

Xin Xing
Georgia Institute of Technology
School of Mathematics

Chenwei Li
Georgia Institute of Technology
Civil and Enviromental Engineering

1. PROJECT DESCRIPTION

Consider a set of n l-bit integers ($l \leqslant 64$): $\{S_1, S_2 ... S_n\}$. Given a number $d (d \leq l)$, the problem is to find all integers which are at a Hamming distance at most d from all of the given n integers.

According to the introduction to master-worker paradigm in the project description, we need to solve two main problems:

1. The **sequential algorithm** for this problem. The key ideas and functions are also used in the parallel algorithm.
2. The **protocol of communication** in the master worker paradigm that should try to overlap computation and communication for best performance.

Our solution design of the two problems are introduced separately in the Section 2 and Section 3. Section 4 shows the results of numerical experiments over the sequential algorithm and the parallel algorithm. At last, Section 5 summarizes the results of our project.

2. SEQUENTIAL ALGORITHM

Following the idea in the project description, the sequential algorithm, first, finds all the potential solutions whose hamming distance with S_1 is not bigger than d. Then, it tests every potential solution with the remaining integers $\{S_2, S_3, \ldots, S_n\}$ to eliminate the unqualified ones from the potential solution.

Define a l-bit integer $e = a_{l-1} \ldots a_1 a_0$ with binary form as a modification 'scheme' that e XOR S_1 gives a potential solution. By the problem requirement, one eligible scheme should at most have d '1' in its bits $\{a_0, a_1, \ldots, a_{l-1}\}$. So, the problem is to systematically find all the possible eligible schemes with at most d '1' in the bits.

We designed an algorithm that can iteratively produce all the eligible schemes(integers) in ascending order. The key function is designed as in Algorithm 1 to generate the next eligible scheme based on a given scheme.

Algorithm 1. *Given an existing scheme $e_0 = a_{l-1} \ldots a_1 a_0$ and the upper bound d of the '1' in bits. The function nextScheme(e_0,d,l) follows the rules below to returns the next eligible scheme e_1 that is bigger than e_0.*

- *If e_0 is of form $a_{l-1} \ldots a_k 11 \ldots 1$, where a_k is the first 0 in the bits. Then return $a_{l-1} \ldots a_{k+1} 100 \ldots 0$.(exactly $e_0 + 1$)*
- *If e_0 is of form $a_{l-1} \ldots a_k 00 \ldots 0$, where a_k is the first 1 in the bits. There are three situations,*
  1. *If there's d '1' in e_0 and $a_{l-1}, \ldots, a_k$ are all '1'. Then, e_0 is the biggest qualified scheme, no bigger scheme exists.*
  2. *If there's d '1' in e_0 and not all of $a_{l-1}, \ldots, a_k$ are '1'. Then, return nextScheme($(a_{l-1} \ldots a_k)_2$, d, $l-k$)$\times 2^k$).*
  3. *If there's less than d '1' in e_0, we can add another 1 into the scheme. So return $a_{l-1} \ldots a_k 0 \ldots 01$. (exactly $e_0 + 1$)*

Then, the whole sequential algorithm is implemented simply as following,

Algorithm 2. *The abstract structure of sequential algorithm is a while-loop as,*

```
Scheme = 0
While (Scheme is not the biggest one)
    Sol = Scheme XOR S1
    Test Sol with S2, S2, ..., Sn
    Scheme = nextScheme(Scheme, d, l)
End
```

Computational Cost.

First of all, there are $\sum_{c=0}^{d} \binom{l}{c}$ eligible schemes. For each potential solution from the corresponding scheme, compute the its hamming distance to $S_2, S_3, \ldots, S_n$ cost at most $O(n)$ computation. And the nextScheme() only takes $O(1)$ computation every time.

Hence, the total computational cost of the sequential algorithm is of scale

$$O\left(n \sum_{c=0}^{d} \binom{l}{c}\right) \tag{1}$$

3. PROTOCOL OF COMMUNICATION

The basic idea of our protocol design is " always assign new task to the idlest processor", which is a kind of heuristic or greedy idea.

The key thing is how to clarify what is the "idlest". From the perspective of master processor, the only chance of communication between workers is when workers finish one task and send back the result. So the master processor could know following status information about the worker by simple collections.

- The number of task left in the worker processor, denote as n
- The receiving time of the present solutions, denote as t.

Given two worker processor A and B with their status information collected as $[n_A, t_A, A]$ and $[n_B, t_B, B]$, we define

$$A \text{ idler than } B \text{ if } \begin{cases} n_A = n_B \text{ but } t_A < t_B, \\ \text{or} \\ n_A < n_B \end{cases} . \quad (2)$$

Generally, our protocol is designed as following,

Worker-Processor.
Right after the finish of one task, the worker-processor sends back the solution to master-processor and check whether there are new tasks or terminate signal. If there's remaining task in the buffer, it goes on working on it without waiting. Otherwise, it waits until new task or terminate signal come.

Master-Processor.
Use two vectors to record the number of tasks in every worker processor and the most recent time of receiving results from every worker processor. When master-processor getting a new task, it checks all the status information and picks the idlest worker-processor to assign the task, where the "idlest" follows the order defined in 2. Every time the master processor receives results from the worker or assigns new task to the worker, it updates the corresponding status recording.

3.1 Parallel Algorithm Implementation

In order to choose the idlest worker efficiently, we construct a allocation-manager in the master processor by priority queue. The element in this priority queue is a triplet $[n, t,$ rank$]$ containing the status information with the order defined in 2.

The abstract structure of our implementation is shown in Algorithm 3.

ALGORITHM 3. *Parallel Algorithm*

```
Priority_Queue manager
Bits_t Scheme = 0

Init manager with [0,0,i] for all worker-processor
(0 task at time 0 in the i th worker)

While (Scheme is not the biggest scheme)
    //Assign new task
    Task_Sol = Scheme XOR S1
    Worker = manager.top();  manager.pop();
    Assign Task_Sol to Worker.

    //Update new status
    Push the new status of worker into manager.
    Update recording of worker's status

    //Collect possible status information and result
    Receive possible results from worker.
    Update recording of worker's status

    //Generate the next scheme
    Scheme = nextScheme(Scheme, d, master_depth)
End while
```

3.2 Computation and Communication Overlap

There are mainly three aspects of computation and communication overlap, in worker processor, in master processor, in the overall task allocation.

In worker processor:.
Workers only check for new task from master after finishing one task. If there's still one job remaining in the buffer, it will continue the computation even without even receiving new task. So it reduces the waiting time for making synchronized communication between the master.

In master processor:.
Instead of computing all the possible tasks first and then distribute, every time master processor get a new task, it assigns it to some worker. So, all the other workers don't have to wait for the task before computing. It's kind of like the computation of master and workers are overlapped.

In overall task allocation:.
By the implementation of priority queue, we always choose the idlest worker at present time. According to the defined order in 2, it can gurantee to assign the new task to the worker with least tasks in buffer. Besides, when two processor have the same number of tasks in buffer, the earlier it sent the most recent results to master, the more time it takes to solve the new task and the more possible it will finish the present task earlier.

Hence, it tries to minimize all the possible waiting time of workers with zero task in buffer.

4. NUMERICAL EXPERIMENTS

For the parallelized computation method, we compared the computation time that result from different number of processors. We set the number of processor to be 2, 4, 8, 12, 16, 24, 36, 48, 64 and compared their computation time when applying different master search depth and bound d of accepted hamming distance to the generated sequence. And then discuss their speed up and efficiency.

To generate the ideal input file, we set up the number of the motifs to be 10, the length of the motif to be 40, and the hamming distance to be determined to be 10,12,14, and compared different parameters and their effect on the total time consumption.

4.1 Time Consumption vs Hamming Distance

With the increase of the Hamming Distance from 10 to 14, the total time consumption of the program increases nonlinearly as the number of the possible solution increase exponentially with the increase of the Hamming distance. We set the search depth of the master node to be 4, and varies the size of Hamming distance and compared the total time consumption.

As Hamming distance's upper bound d increases, the number of the possible solution increase in an exponential way. Therefore, the time consumption of the Motif-Find implementation increases nonlinearly. As we could see from the

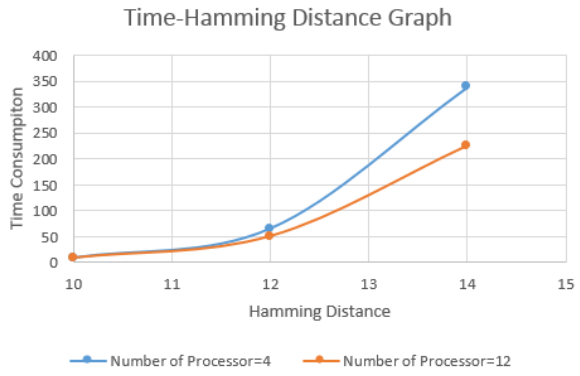

Figure 1: $n = 10, l = 40, masterDepth = 4$

figure 1, d would greatly affect the computation time. For the size of the project, we should consider a relatively reasonable size of the d.

4.2 Time consumption Vs Number of processor

To compare the effect of the number of processor on the total time consumption, we first compare the effect on different size of Hamming Distance. As the total time consumption consists of the computation time and the communication time , when the size of the hamming distance is not large, which means the computation time is relatively small, the total computation time would be affected by the communication time, the latency of the transferring message and the communication bandwidth.

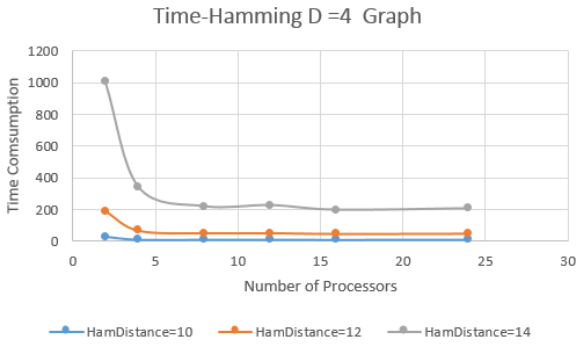

Figure 2:

The time consumption would decrease dramatically with the increase of processor number at first but would remain at a relatively stable stage when the number of processor reaches a certain amount. This could be caused by the communication time that actually existed, and the effect would be better illustrated with a larger problem size, saying that the computation time are relatively way larger than the communication time. As we could observe from the Figure2, when the problem size is relatively small, the decrease percentage of time would be relatively small compared with the problem with hamming distance equals 12.

When look into the effect of the number of processor on the time consumption with a fixed Hamming distance in Figure 3, we would look at the speed up rate and the efficiency to compare the parallel algorithm's effectiveness

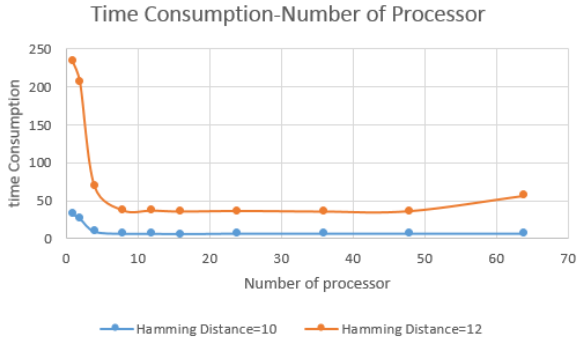

Figure 3:

4.3 Speed up and Efficiency of the parallel algorithm

When we look at the speed up rate of the parallel method, we could see that the increase rate are not in linear relation with the number of processor. Problem with larger Hamming distance would have larger speed up rate because the computation time would be the dominant factor to affect the total tine consumption. Problem with smaller hamming Distance would be more likely to be affected by the communication time as they would usually take very small amount of time on the computation part. And for the number of node, when we implement the method using 2 node, the total time consumption is a little bit larger than the sequential one. The reason for this is with 2 node computation, the process would be influenced by the communication protocol between master node and worker node, which would not happen in the sequential computation.

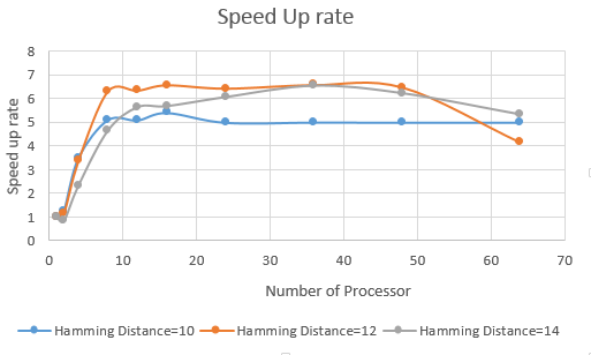

Figure 4:

For the speed up, we could see that when the number of node is small, the speed up rate would be more likely to be linear increase, but when the node reaches certain amount, the speed up rate would remain at a relatively steady level, and even drops for some instances. That could be easily observed from Figure 4. Communication time played a very important role in the speed up rate.

For the efficiency of the parallel computation, we could see from Figure 5. The efficiency would decrease with re-

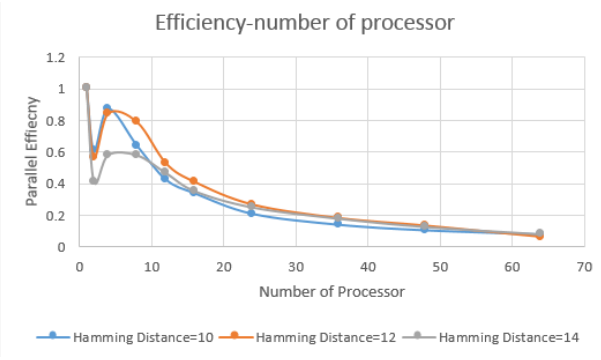

Figure 5:

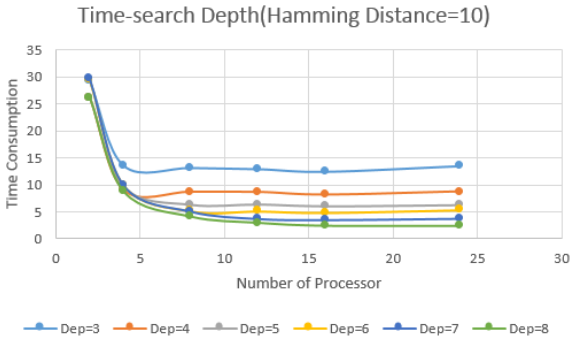

Figure 6:

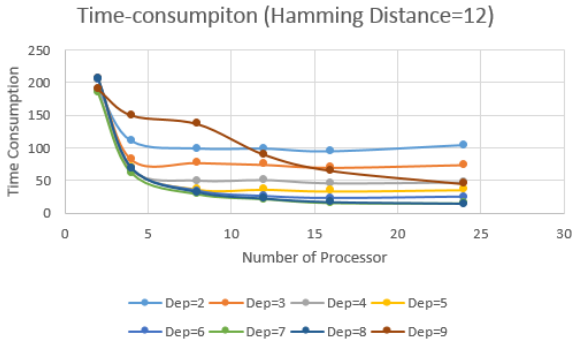

Figure 7:

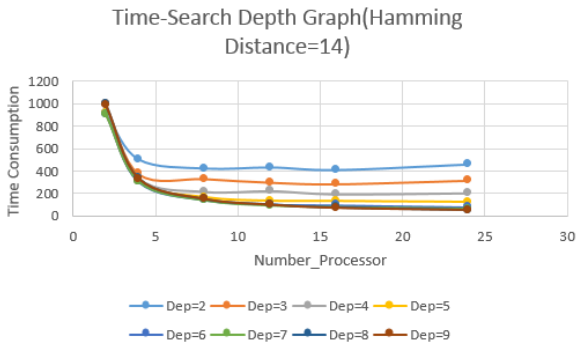

Figure 8:

spect to the increase of the number of processor. When the number of processor is around 4 to 8, the efficiency would remain relatively high while the computation time relatively reduced, but when the number of processor continues to increases, the efficiency would continue to drop and the time decrease rate would also decrease, thus make the parallel implementation not efficient.

4.4 Time Consumption vs Search Depth

The effect of the search depth could be seen from Figure 6,7,8, it is apparent that the when the number of the processor is small, the search depth effect is small as the worker processor would have to complete comparatively large percentage of the work, and it matters less if the search depth is small for the master node depth. But with the increase of the number of the processor, the effect would be more apparent as the master node could sent out the problem immediately right after it finishes its part. And with the right depth for the master node's search depth, there could expect an ideal setting in which the implementation would achieve its best performance.

Take an example with the number of node is 12,16 and 24 with hamming distance is 14, and we take a look at the search depth's effect on the time consumption of the implementation. And we use the time reduction rate to measure the effectiveness. From Figure 9 we could see that when the processor size increases, the larger the search depth is, the larger the time reduction rate would be. And for each set with a fixed number of processor, there exist a best combination that with the number of the processor and the master node's depth. The best combination should also take the speed up rate and efficiency into consideration. And we could see from Figure 9, when number of processor equals 12, there exists an obvious peak point, and when the Number of processor increase, the peak also moves. This is because with more processor join into the working space, there exist a balance point that the master node could work efficiently to get every set of problem done with the worker node.

And we could see from Figure **??**, with the different size of the problem, the time reduction rate differs. And the best balanced location differs with as various factors would affect the reduction rate. As for small size problem , the time reduction could be apparent as this could shorten the computation time for the workers and the overlapping effect would reaches a good performance.

5. SUMMARY

The main high light of our design is the task-allocation with priority queue. By this heuristic design, we always first allocate new task to idlest worker. While using some direct parallel computation method can indeed exceed the performance of sequential algorithm, we believe that our parallel algorithm with optimized task allocation could improve more. However, due to the limitation of time, a comparison between parallel algorithms with and without optimized task allocation hasn't been finished. It would be a good ending to compare the two to show the effect of optimized task allocation.

Besides, through all the numerical experiments in Section 4, we get following conclusion over parallel algorithm and sequential algorithm.

1. Parallel computation would decreases the total time

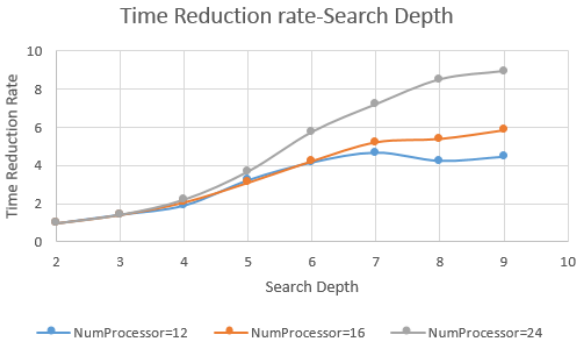

Figure 9:

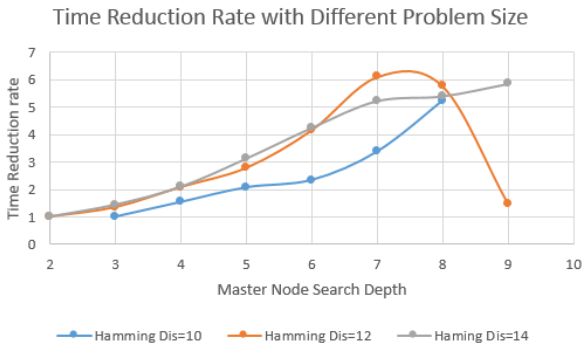

Figure 10:

consumption of the implementation. But its efficiency would be compromised as the number of processor increases.

2. Best combination exist in the different problem size with different search depth for the master node. Best combination could help in reduce the time consumption while keep the efficiency of the parallel algorithm.

3. Speed up rate would stay on a stable level after increase of the number of processor reaches certain point. And the efficiency of the parallel algorithm would stays at relatively high stage at the begging, but would keep on dropping as the number of processor increases to a larger number.

4. The parallel algorithm is effective in reducing the time consumption.