Assignment 4 Solutions

Xiaoping Li

April 25, 2014

1

1.1 a.

In my personal view, the decomposition of the problem should follows data decomposition. This problem is contains a large data of similar data. I decompose a big grid into smaller grids and solve the problem on those smaller grids in parallel. As for task decomposition, it usually need independecy between each task. So I think this prolem follows data decomposition. Each data partition corresponds to a task.

1.2 b.

Each task has dependency on its neighbors since it needs the information from neighbors. In detail, each task has two neighbor tasks: the upper neighbor and the lower neighbor. Each task needs boundary rows from its neighbor: top row of its lower neighbor and bottom row of its upper neighbor.

1.3 c.

The structure of the algorithms of this solution is Geometric Decomposition. This algorithm is organized around a data structure that has been decomposed into concurrently updatable "chunks". The whole block is decomposed geometrically equal size. Also each task is associated with a data block.

2

2.1 a.

- **Advantage**: Apply the method provided by the problem, we can have more task to perform than just horizontally divide the data. Max number should be 256, but horizontally max task is 16.
- **Disadvantage**: each data partion has 8 neighbors now, compared to the 2 neighbors of horizontal slicing. More messages need to be passed which will introduce more overhead compared to horizontally slicing.

2.2 b.

Denote the grid has width D and the number of processors is N. In this application, $D = 16$.

- **Horizontal Slicing**

  message sizes: message size from/to neighbors is D, message size sent to master is $\frac{D^2}{N}$

  number of messages: this is the number in one iteration—-$(N + N + (N - 1)) = (3N - 1)$. Each processor sends/receives 2 messages to its neighbors. Each non-master processor sends 1 message to master.

- **Square Decomposition**

  message size: since N*A = D*D, we can have $\frac{D}{\sqrt{N}}$ (to/from top/bottom/left/right neighbors); its size in corner is 1 (to/from topright/topleft/bottomright/bottomleft neighbors); size for master processor is $\frac{D^2}{N}$ (to master processor).

  number of messages: $(8N + (N - 1)) = (9N - 1)$ in each iteration.

2.3 c.

Memory overheads. The same notations are used as in the above secton.

- **Horizontal Slicing** : For each processor, it adopts two arrays with size D to keep information from neighbors.
- **Square Decomposition**: For each processor, it needs an array of length $\frac{4D}{\sqrt{N}} + 4$ to keep neighbors' information.

2.4 d.

Square decomposition should have more flexibility over the horizontal slicing if we can have different data size. It has more dimension of flexibility than horizontal slicing.

2.5 e.

For square decomposition, we can apply 4 independent loop of message passing to get deadlock free message passing.

For example, for horizotal slicing, we just need to consider top and bottom neighbors of each data partition, then all processors forms a loop: $0, 1, 2, \cdots, N - 1, 0$. Messages could be passed in a similar way to what we have done in class example. We deploy message sending/receiving based on the processor's rank. If a processor's rank is even, then it first send a message to its next neighbor and then receives a message from its previous neighbor. Then, it first receives a message from its previous neighbor, send a message to its next neighbor. If a processor's rank is even, then it first send a message to its previous neighbor and then receives a message from its next neighbor. Then, it first receives a message from its next neighbor, secondly send a message to its previoust neighbor. It guarantee the free deadlock.

Just like described above, we could get the message passing in the other 3-dimension: left-right, "bottomleft-topright" and "bottomright-topleft".

This ensures free deadlock passing in square slicing.