High Performance Computing
Homework 2
Report

Peiyun Hu, 2010011297
Department of Computer Science & Technology

October 14, 2012

1 A parallel algorithm for a limited Lx=b

1.1 Description

Please refer to the homework file.

1.2 Answer

Actually, there is an order that must be followed. For x_i is determined by $x_{i-1}(i >= 0)$, x_i should wait until new x_{i-1} is calculated.

So, the dependencies graph i actually a complete graph, which is shown as Figure 1. And it is really embarrassing to execute in parallel. However, there is another way to think over this program. And when the definition of task changed, this program could be paralleled better. Specifically, we define (i, j) a task.

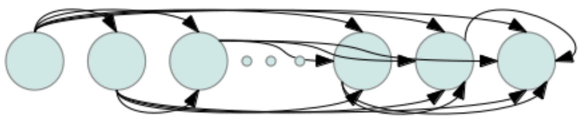

Figure 1: Dependencies

The order could be shown in the Table 1.

As is shown in the table above, we could ideally infer that we could solve this problem in $n+1$ column time, only if the number of processors is more than that of rows. If so, each column will cost almost the same time. While, especially when the size of matrix is very enormous, we should execute them in a specified order.

When considering the order or execution, we should notice that even if we decompose the task into smaller granularity, there are still dependencies among them. To put it in another way, in order to follow the dependencies, we assign the tasks levels of priority. For task (i, j), unless x_j is worked out, it should never be executed. Also, the executed order of tasks (i_1, j), (i_2, j), . . . , (i_k, j), should be in ascending

t	1	2	3	4	5	...	m-1	m
	(0,)					...		
		(1,0)				...		
		(2,0)	(2,1)			...		
		(3,0)	(3,1)	(3,2)		...		
		...	...	...	...	...		
		(n-1,0)	(n-1,1)	(n-1,2)	(n-1,3)	...	(n-1, n-2)	
		(n,0)	(n,1)	(n,2)	(n,3)	...	(n, n-2)	(n,n-1)

Table 1: Ideal order of execution

order of i, that is to say, the smaller i is, the task should be executed earlier. Because the smaller i is, strictly, the possibility of augment of available x_i is higher. In order to expand available tasks, tasks that have smaller i are executed first.

And there is an specific example shown in 1.3.

1.3 An example

If n = 256, and only 16 processors, we could create a working pool which looks like a stack. Each time when we pop out task like (i, i-1), we push in { (i+1, n), (i+1, n-1), . . . , (i+1, i-1) } in successively. In each period, we pop out up to 16 tasks to execute, then push in if needed. There is a master processor to allocate tasks to other processors, and other processors wait for tasks. After every period, the sequence of x should be updated in each processor.(Indeed, I think it should be each 'thread' instead of each 'processor'.)

In this way, the tasks in the stack are all available to be allocated and executed.

2 Simplified Bucket-Sort

2.1 Description

Please refer to the homework file.

2.2 Answer

2.2.1 Analysis

The bucket-sort problem contains the iteration of array A and buckets. So to balance the workload of two aspects, we have two strategies.

2.2.2 Partition the Input Data

The strategy of decomposition is to divide input data into p parts averagely, that is to say, each part has roughly $\frac{p}{n}$ elements. Then every process works as a sequential task. Finally, communication should be done to gather results to one process, then the 'master' process print the results out .

2.2.3 Partition the Output Data

Aside from the strategy above, we develop another which is focued on the workload of buckets' iteration. There could be a 'master' process, which is denoted p_0. The iteration of array is done in p_0, and it will get the value of A[i], then put the index of it into the correponding bucket. When assignment is done, including p_0, each processor is allocated with several buckets averagely, which is in consecutive order of buckets' indices. Then every process do the iteration of buckets and fetch indices from nonempty buckets to fill up a new sorteddd array. Finally, after filling up inside each process, p_0 starts to collect results and print them out.

3 More about Problem 2.

3.1 Description

Please refer to the homework file.

3.2 Answer

It depends which decomposition leads to a better parallel algorithm, as is stated in 2.2.1.

Specifically When n $\gg$ r, which means that iteration of array will cost much more than that of buckets. So the method in 2.2.2 will outperform the one described in 2.2.3. When n $\ll$ r, and the cost of iterating buckets will outnumber that of iterating the array. In this case, the method in 2.2.2 is more appropriate. When the gap between n and r is small, there would be little difference in the level of algorithm.

Moreover When size of both array and buckets are huge, actually, both two strategies could be leveraged to fulfill the task. That is to say, iteration of array and buckets are done in parallel way.

References

Appendix