University of the Witwatersrand

School of Computer Science and Applied Mathematics

COMS3008: Parallel Computing Lab Assignment 2

25 July 2016

*B*y Chalom, J. (711985)

1 General Assumptions and Methodology

The computer which the decomposition of the factorial algorithm is to be run on only has two other processing units. This is a hybrid system which has both private memory spaces and shared memory spaces. The factorial algorithm being investigated will have the number 1 as the last number multiplied to the running variable in order to find the factorial as some integer n.

All drawn diagrams were drawn using `http://draw.io/`, and all results from empirical analysis, were statistically analysed by the data processing program used in lab 1. Those averaged results were then plotted using Microsoft Excel.

The system used for empirical analysis, had an i3 Ivy-bridge CPU, with four processing units and 8GB of RAM.

2 Directed Acyclic Graphs

Chosen Decomposition:
Let n be the number whose factorial is being investigated.
$n = \{n \in Z \mid 1 \leq n \leq Z\}$

Data dependency of the chosen decomposition

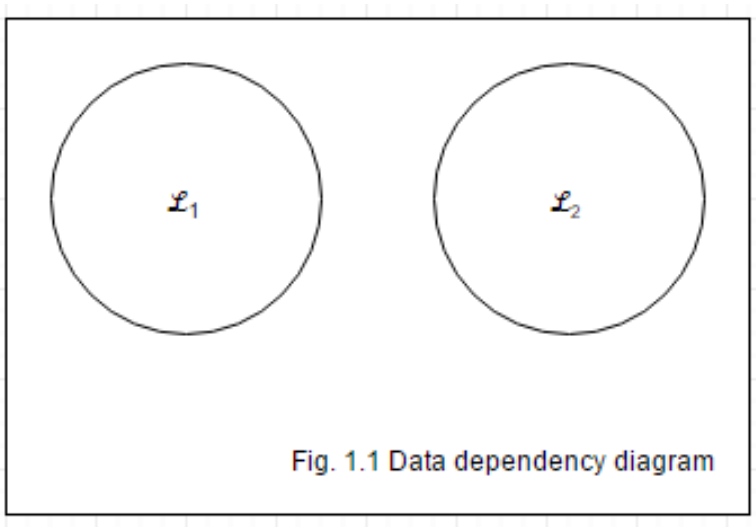
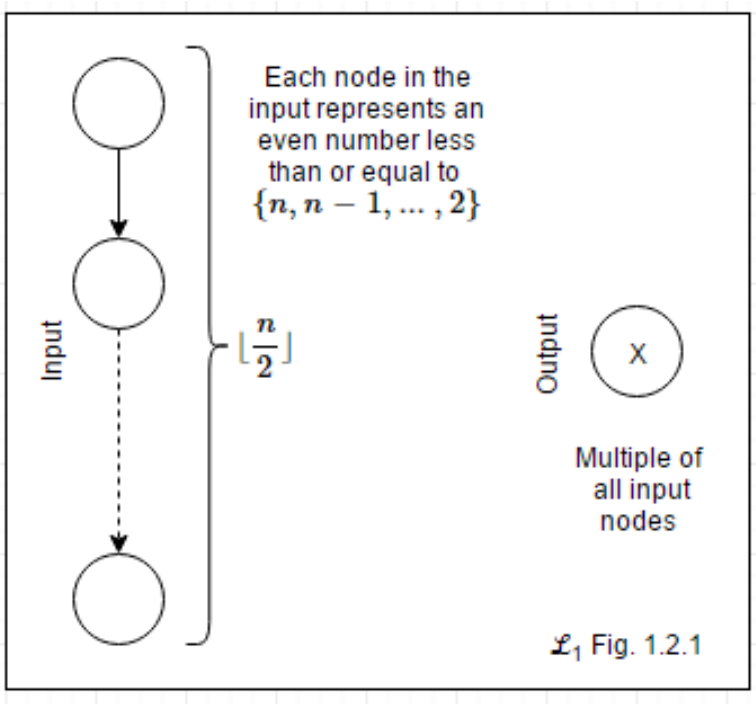
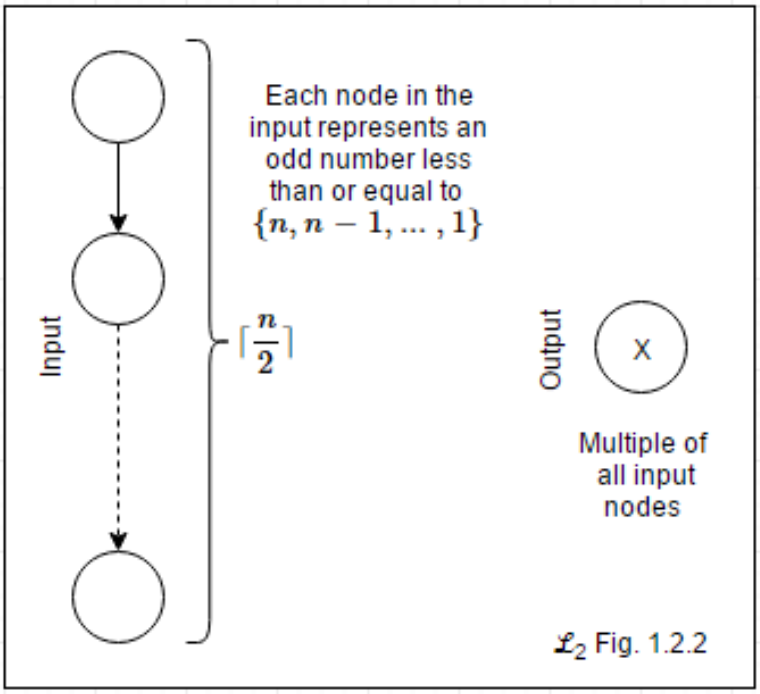


Fig. 1.1 Data dependency diagram

Let L be the set of numbers constructed from n such that $L = \{L \in Z \mid n \leq L \leq 1\}$. Therefore let L_1 and L_2 be subsets of L such that $L \in (L_1 + L_2)$. Let $i \in Z$ *s.t.* $0 < i < n$. Therefore $L_1 \in \{n \mid (n-i)\ mod\ 2 = 0\}$ and $L_2 \in \{n \mid [(n-i)-1]\ mod\ 2 = 0\}$.
i.e. L_1 contains even numbers and L_2 contains odd numbers (up to n).

Figures 1.2.1 and 1.2.2 shows this decomposition of the data for the factorial algorithm.





Task interaction of the chosen decomposition

Let T_1, T_2 and T_3 be the tasks which make up the decomposition of the factorial operation.

T_1 uses the data-set L_1.
T_2 uses the data-set L_2.
T_3 uses the data-set L_3.

T_1 (even) and T_2 (Odd) both iterate through their respective data-sets and multiply a running variable by the next number in the data-set. This variable is initialised by 1 and it becomes the answer of the previous iteration. T_3 is the task which multiplies the respective results of T_1 and T_2.

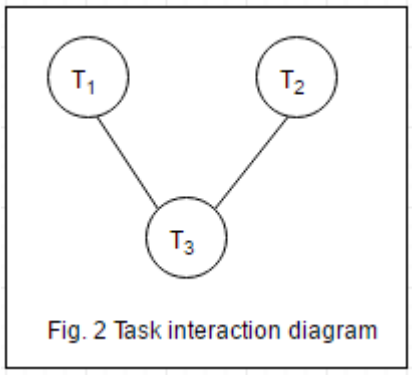


Fig. 2 Task interaction diagram

Task dependency of the chosen decomposition

T_1 (even) and T_2 (Odd) lead to a barrier, which is T_3. This operation waits for both T_1 and T_2 to complete and then a reduction operation occurs - denoted here by: X. T_3 takes the results of T_1 and T_2 and multiplies them together to get the final result, which means that both T_1 and T_2 have to complete before T_3 can be executed.

In this case both T_1 and T_2 have been denoted in fig 3 by their internal parts - the iterations which occur inside them.

If n is odd then the number of nodes in T_1 is $\lfloor \frac{n}{2} \rfloor - 2$ and T_2 is $\lceil \frac{n}{2} \rceil - 1$. If n is even then the number of nodes is reversed - where T_2 has the floor function applied. One task is left out and that is T_3. The other negative one is there to denote that odd numbers end in one. Anything that is multiplied by one stays the same, this fact helps to make sure that the number of operations at the end is always n, since an odd numbered n will lead to a disparity between the size of L_1 and L_2.

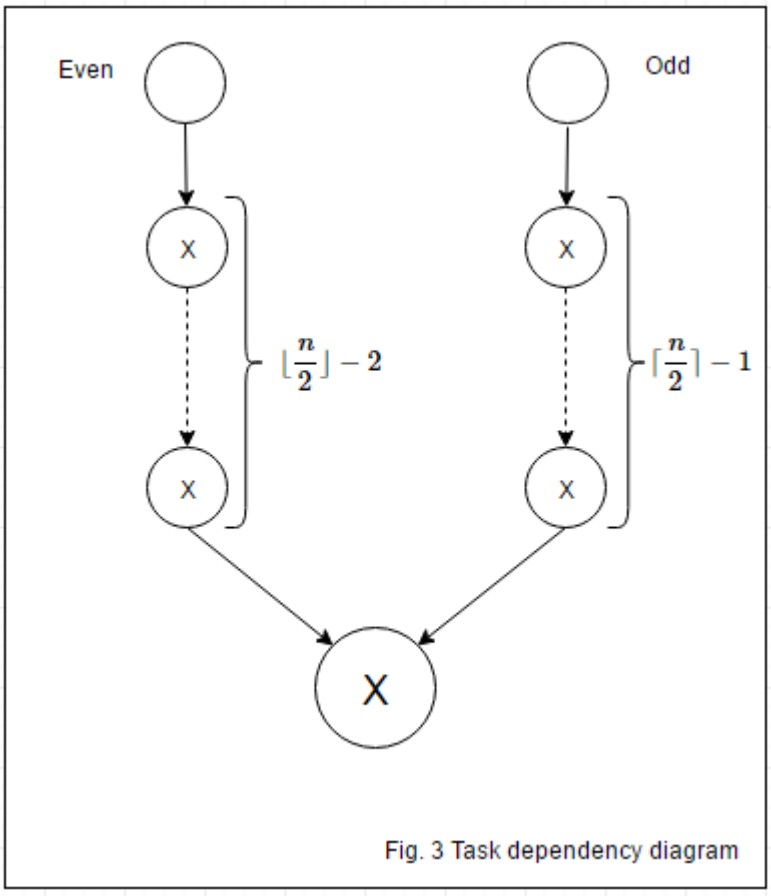


Fig. 3 Task dependency diagram

3 Algorithm Complexity

Assumptions made regarding the Sequential Complexity:
The complexity of the sequential algorithm is: $\ominus(n)$, where n is the number being investigated as that many operations have to be completed to find the factorial.

Recursive Complexity:
The complexity of the recursive algorithm is: $\ominus(n+k)$, where n is the number being investigated as that many operations have to be completed to find the factorial. k is the arbitrary constant which represents the cost of the function recursively spawning itself n times.

Parallel Sequential Complexity:
The internal time complexity of the sequential algorithm when it is parallelised is: $\ominus(log(\lfloor \frac{n}{2} \rfloor) + 1)$, where n is the number being investigated and 1 is the assumption that the reduction operator and final result takes constant time to complete. Another assumption that made is that both T_1 and T_2 complete their operations at roughly the same amount of time due to them having approximately the same number of operations to complete. The reason why this implementation is logarithmic in nature is because the implementation of a parallelised for loop in openMP will produce a parallelised reduction decomposition which is logarithmic.

The computational complexity of this decomposition is: $\ominus(n + k)$, where n is the number being investigated as that many operations have to be completed to find the factorial and k is a constant that represents the complexity of the reduction operator and the complexity of the final operation. This is because the parallel version can complete its calculation faster than the standard iterative approach but it cannot make the amount of computations required to complete the calculations any less.

Parallel Recursive Complexity:
The time complexity of a decomposition of the recursive algorithm is: $\ominus(\frac{n}{2} + k)$, where n is the number being investigated and k is a constant that represents the time taken due the reduction operations, wait factor of the different sections used and the time taken in memory due to executing the recursive operations of the algorithm. The algorithm is split into two sections which are computed in parallel which is why the time complexity is halved.

The computational complexity of this decomposition is: $\ominus(n + k)$, where n is the number being investigated as that many operations have to be completed to find the factorial and k is a constant that represents the complexity of the reduction operator and the internal recursive nature of each section. This is because the parallel version can complete its calculation faster than the standard recursive approach but it cannot make the amount of computations required to complete the calculations any less.

4 OpenMP Implementation

The empirical analysis of each of the four algorithms was done by setting n to 1000 and running each algorithm 25 times, and then n is incremented by 1000. This process is repeated 500 times, to get results for n = 1000 to n = 500000.

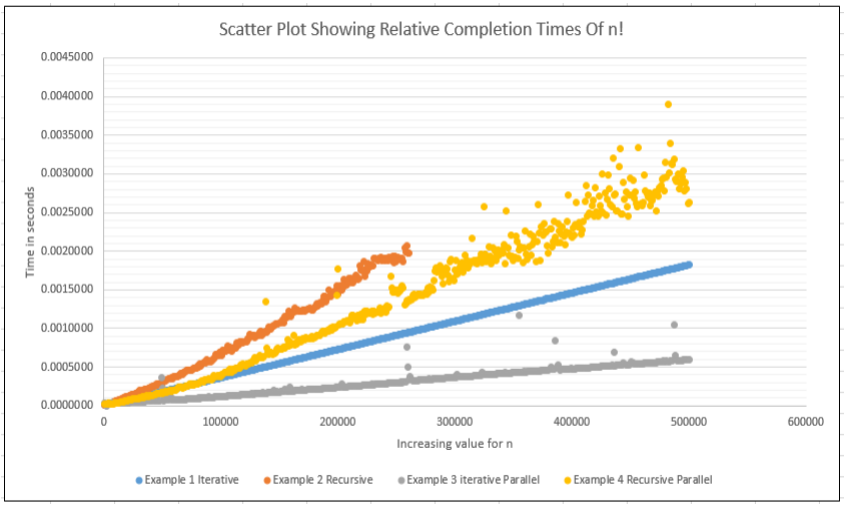
Unfortunately algorithm 2, the sequential recursive algorithm has a stack limit of n = 261000, after which there is a segmentation fault. This is shown in the scatter plot (below) where algorithm 2 is just over half the amount of n values tested than other three plots.

The total execution times of the algorithms (excluding algorithm 2) from n = 1000 to n = 500000 were also recorded.

Algorithm 1	Algorithm 3	Algorithm 4
12.025520	4.927197	19.324828

Table 1: Completion Times Of Algorithms (in Seconds)

From the table above you can see that the third algorithm (The parallel iterative algorithm is the fastest completing every calculation of the other algorithms tested.



The scatter plot concurs with the execution times table because it shows the third algorithm being the fastest of the four. Another interesting obser-

vation is that the first algorithm which is the sequential iterative approach is the most predictable and 'stable' of the algorithms. Algorithm 2 seems to be the worst performing, and Algorithm 4 seems to be the most varied with data-points scattered the furthest from a line of best fit.

Algorithm 3's result can be explained by internal optimisations in openMP, specifically openMP parallelising the reduction operation - producing a tree decomposition of the algorithm.