Hyper Quicksort implementation in OpenMP

Assignment-2
CS F422: Parallel Computing

Harshit Garg (2018A7PS0218P)
Guntaas Singh (2018A7PS0269P)
K Vignesh (2018A7PS0183P)

1 Introduction

Here we provide an implementation of Hyper Quicksort in OpenMP. First, the algorithm is presented with some explanation. Then, the fraction of the code that is running serially on each parallel thread is estimated, followed by calculations of speedup and efficiency. Then we compare the obtained results with Amdahl's Law. Then we derive asymptotic expressions for isoefficiency. Finally, we estimate the maximum number of processors that can be used to solve this problem cost optimally.

2 Algorithm

1. Each thread starts with sequential quicksort on local list.
2. Choose pivot from any one of the thread and broadcast it across all threads.
3. At each thread, divide the array into low and high list.
4. Swap the list between threads.
5. On each thread, remaining half of local list and received half list are merged into a sorted local list.
6. Recurse with upper half and lower half threads.

3 Speedup, efficiency, serial fraction

The theoretical formulation for speedup and efficiency is given as follows:
(The input sizes are taken as 128, 256, 512, 1024. Effectively translating as 7, 8, 9, 10 in logarithm of 2)

3.1 Speedup

Speedup is given by

$$S = \frac{T_s}{T_p}$$

$$= \frac{O(nlogn)}{O(\frac{n}{p}(logn + logp))}$$

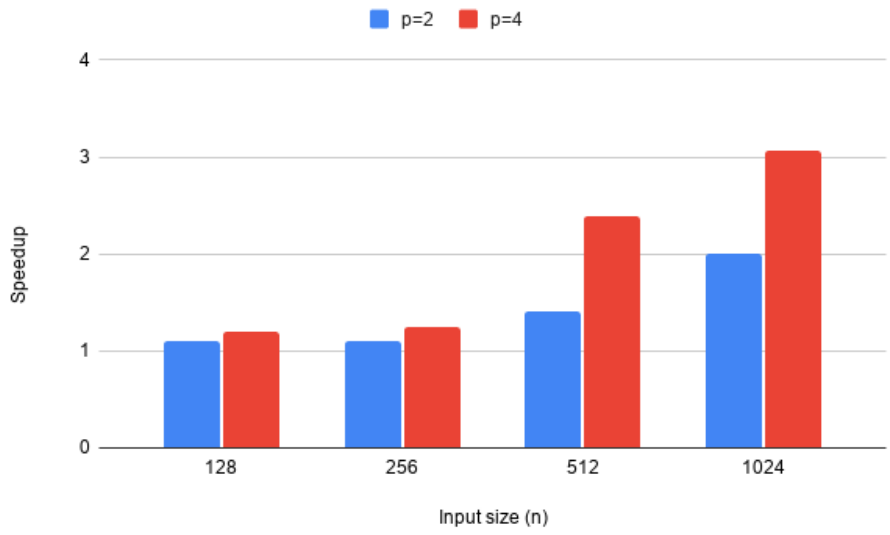

Fig. 1. Speedup vs input size for different number of threads

3.2 Efficiency

Efficiency is given by

$$S = \frac{T_s}{p * T_p}$$
$$= \frac{O(nlogn)}{O(n(logn + logp))}$$

3.3 Serial fraction

Serial fraction is given as

$$f = \frac{T_o}{W}$$
$$= \frac{W - T_{par}}{W}$$

The values of f calculated from simulations are given in Table 1 listed below.

Based on Amdahl's law, we get the upper bound of expected Speedup of the algorithm. It can be seen that the results of the simulation satisfy the Amdahl's law since for a given number of processor and input size, the speedup obtained is lower than that estimated by Amdahl's Law. Complete data is available in Table 2.

3.4 Raw Data

The raw data is available as

4 Isoefficiency Analysis

We assume that p threads are sorting n elements, where n $\gg$ p.

1. At the start, each thread sorts the respective list. Expected time complexity is $\Theta[(n/p)log(n/p)]$.

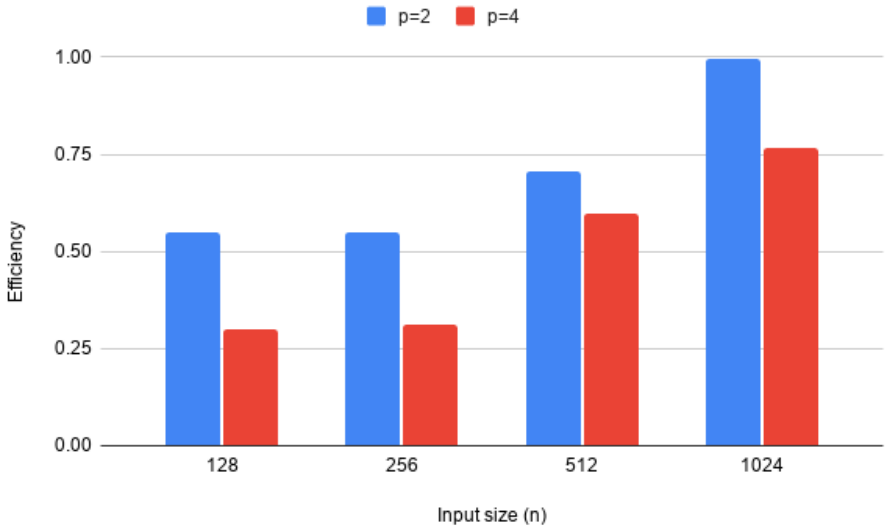

Fig. 2. Efficiency vs input size for different number of threads

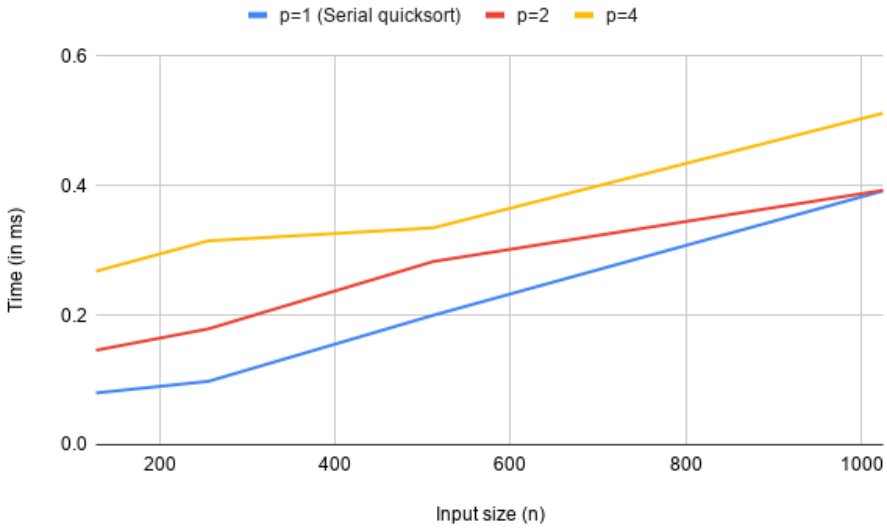

Fig. 3. Execution time vs input size for different number of threads

n	p=1	p=2	p=4	f@p=2	f@p=4	S@p=2	S@p=4	E@p=2	E@p=4
128	0.08	0.146	0.268	0.0875	0.1625	1.095890411	1.194029851	0.5479452055	0.2985074627
256	0.098	0.179	0.315	0.08673469388	0.1964285714	1.094972067	1.244444444	0.5474860335	0.3111111111
512	0.2	0.283	0.335	0.2925	0.58125	1.413427562	2.388059701	0.7067137809	0.5970149254
1024	0.392	0.393	0.512	0.4987244898	0.6734693878	1.994910941	3.0625	0.9974554707	0.765625

Table 1. Raw Data Calculation based on simulations in OpenMP

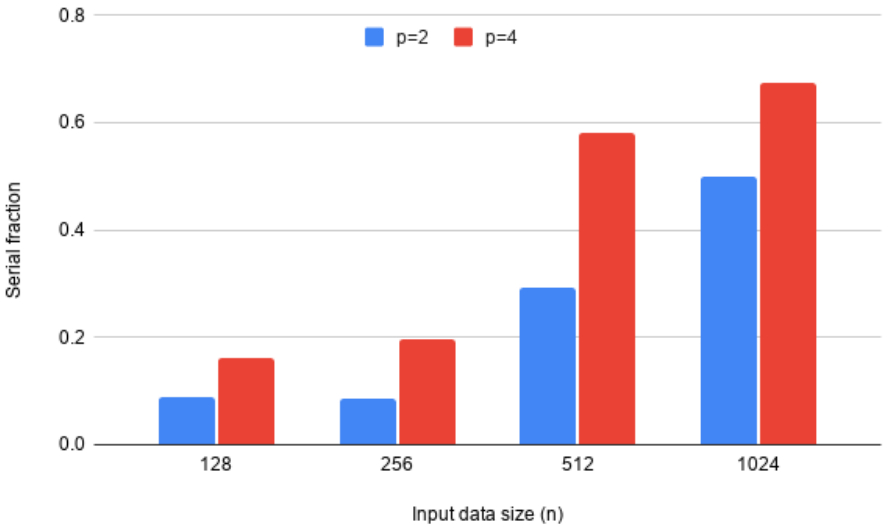

Fig. 4. Serial Fraction vs input size for different number of threads

n	S@p=2	S@p=4
128	1.83908046	2.68907563
256	1.840375587	2.516853933
512	1.547388781	2.457858772
1024	2.334468085	3.324324324

Table 2. Speedup Calculation based on Amdahl's Law

2. Assuming that each thread keeps n/2p values and transmits n/2p values in every split-and-merge step, the expected number of comparisons needed to merge the two lists into a single ordered list is n/p. Since the split-and-merge operation, is executed for hypercubes of dimension (log p), log(p)-1, ... , 1, the number of comparisons performed over the split-and-merge phase of the algorithm is $\Theta[(n/p)logp]$.
3. The number of comparisons performed during the entire algorithm is $\Theta[(n/p)(logn + logp)]$.
4. If the threads are organized as a d-dimensional hypercube, then for the broadcast of the pivot $\Theta(d)$ time is required. However, since n $\gg$ p, the time to send part of an array overshadows the broadcast time.
5. Assuming each thread send half the array each time, the time needed to send n/2p values each time is $O(n/p)$. For log(p) iterations, it comes out to be $O(n.logp/p)$. Original quicksort algorithm asks for no interprocess communication, so this is the entire communication complexity for hyperquicksort.
6. The communication overhead for entire hyperquicksort algorithm for p threads would be $p * \Theta(n.logp/p)$ i.e. $\Theta(n.logp)$.

   Sequential quicksort time is $nlogn$. Hence isoefficiency function would be

$$nlogn \geq Cn.logp \implies logn \geq Clogp \implies n \geq p^C$$

where C is any constant that determines the scalability of the system.

4.1 Maximum processor limit for cost optimality

From the expression derived above, we can conclude that the maximum number of processors that can be used to solve the problem cost optimally is

$$p = O(n^{\frac{1}{C}})$$

.