コンピュータ科学特別講義４　レポート課題３

コンピュータ科学専攻　修士課程１年　今村秀明　48-186112

1　ソースコードについて

まず、作成したプログラムのソースコードがある github リポジトリは以下である。

`https://github.com/HideakiImamura/Parallel_Computing`

課題の作成に利用したソースコードは、ex3/src 内にある。各コードについて簡単に説明する。main.c は実験用のコードである。本レポートでは Maximum Sub Sequence 問題を対象とし、愚直な逐次アルゴリズムと parallel prefix を用いた並列アルゴリズムの比較を行った。コマンドライン引数として生成するランダムな整数配列の最大の長さの指数を与え、指数を１からこの数字まで変化させる。さらにスレッド数を変化させて、かかった時間を計測した。スレッド数の最大値 8 は私の使用した PC である MacBook Pro, 2.9GHz Intel Core i7 の使用可能な最大プロセッサ数である。csv ファイルの形式で (使用スレッド数、配列の長さの指数、逐次アルゴリズムが処理にかかった時間、並列アルゴリズムが処理にかかった時間) を並べた物を出力した。

また、graph_gen.py は matplotlib を用いて上で得られた csv ファイルをグラフ化するための補助的なコードである。

はじめに本レポートの最大にして最悪の欠点について説明しておく。それは、「並列アルゴリズムの実装に間違いがあること」である。具体的には、逐次アルゴリズムと出力は一致しているが、８並列で実行しても逐次アルゴリズムよりはるかに遅いということである。(自分の OpenMP に対する知識不足によるものです。不完全なままにレポートを提出することをどうかお許しください。大変申し訳ございません。)

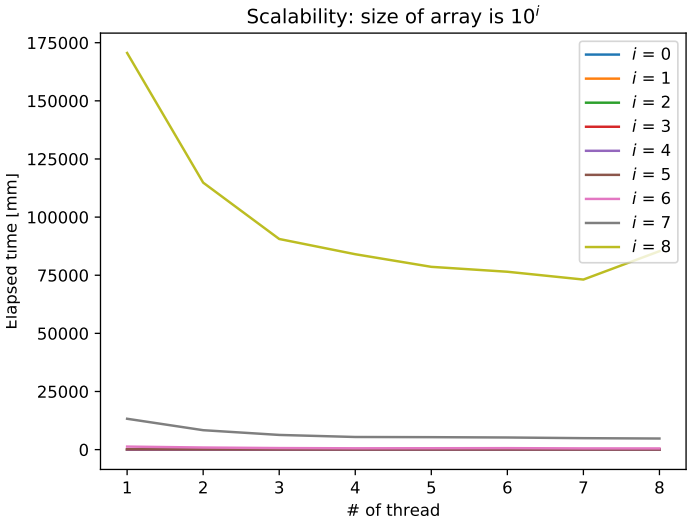
2　二つのアルゴリズムの一致性について

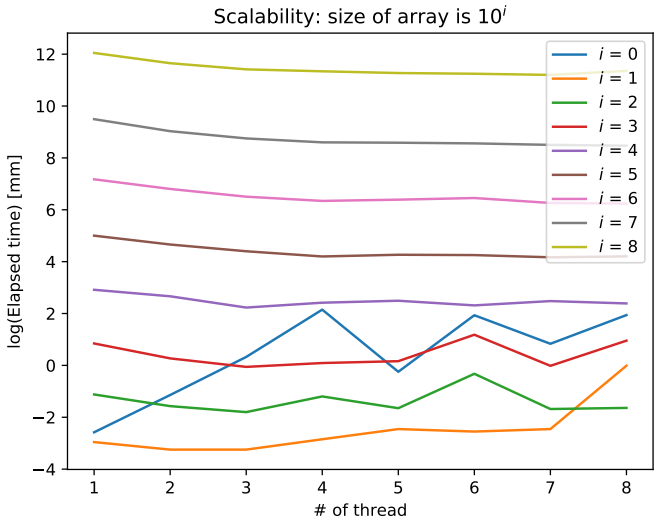
プログラムの実行画面の例を以下に示す。

a は生成されたランダムな整数配列で、長さが１０のものである。その下には i を 1 から 9 まで変化させて、長さ 10^i のランダム整数配列を作り、Maximum Sub Sequence の問題を解いた結果が示されている。上の (u, v) が逐次アルゴリズムの結果で、下が並列アルゴリズムの結果である。確かに両者は一致している。その下の (seq, par) は、得られた (u, v) に対して部分列の和を出力している。両者によって得られた解は確かに同じ値を返す。

3　スケーラビリティについて

プログラムを実行して得られたグラフを以下に示す。





各画像サイズごとにスレッド数に対する実行時間をプロットした。上の画像では縦軸は実行時間を mm で示しており、下の画像では縦軸は実行時間の対数を取っている。下の画像を見るとわかるように、配列のサイズが小さい時はスレッド数を増やしても実行時間は必ずしも減少しないことがわかる。これは配列のサイズが小さい時は、並列化による計算時間の高速化よりもその他の部分のオーバーヘッドが支配的であるからだと考えられる。配列のサイズが大きい時は（$i \geq 4$）、スレッド数を増やすと計算時間はわずかに減少していることが見て取れる。実装に間違いがあると考えられるため断言はできないが、この減少度合いがかなり微小なものであるため、さらに並列化するスレッド数を増やせば計算時間がさらに改善するのではないかと考えられる。

ただ、同じ入力に対する逐次アルゴリズムの計算時間は下表のようになる。逐次アルゴリズムの方が並列アルゴリズムよりも早く、並列アルゴリズムの実装に間違いがあると思われる。

配列のサイズ [bytes]	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
逐次 Alg の計算時間 [msec]	10^{-7}	0.001	0.005	0.05	0.3	2.8	27	192	1941
並列 Alg の計算時間 (8 thread) [msec]	0.95	0.98	0.2	2.6	11	67	512	4773	85345

4 アムダールの法則に基づく並列化可能な割合の計算

アムダールの法則によれば、プログラム全体で並列化可能な部分の割合を P、その部分の性能向上率を S、全体の性能向上率を A とすると

$$A = \frac{1}{(1-P) + \frac{P}{S}}$$

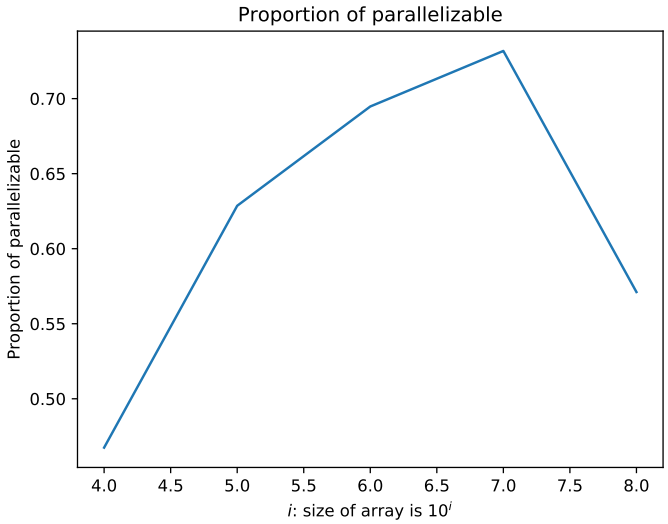
が成り立つ。これを変形すると、

$$P = \frac{\frac{1}{A} - 1}{\frac{1}{S} - 1}$$

が得られる。よって今、S を使用するスレッド数とし、A を実際の性能向上率とすると、P が計算できる。A は具体的には以下のようにして計算する。

$$A = \frac{\text{スレッド数が 1 の時の処理時間}}{\text{スレッド数が } S \text{ の時の処理時間}}$$

これを画像サイズを変化させて計算したグラフが以下である。なお、S の値は 8 とした。



このグラフから分かるように、配列のサイズ 10^4 から 10^8 の間の時は P は 0.5 と 0.7 の間にある。実装に間違いがあると考えられるため断言はできないが、8 並列程度では十分な並列化ができていないのではないかと考えられる。さらにスレッド数を増やせば上の P はある一定の値に収束し、プログラムの中で並列化可能な割合がより正確に推定できると考えられる。