

ChatGPT(Plus)に並列プログラムを書いてもらう

(説明)

ChatGPT(Plus)に質問をして並列プログラムを書いてもらいます。私はこれまでそれに関係する質問をしたことはなく、下の(1)が初めてです。ゴシック体で私の質問を記載しており、その下にChatGPTからの回答をつけています。ただしそれぞれのプログラムはソースコードのかわりにファイル名(.cなど)を記載しています。プログラムの本体は以下のEMCのGitHub公開サイトにありますので適宜ご参照ください。

https://github.com/embeddedmulticore/MBP_Public/tree/master/ParallelProgramsAI
なお、そのファイル名に合わせて下の記述を修正しております。人的修正ミスが発生している場合がありますが、ご容赦ください。

(1)(質問) 1から1000までの和を計算するCとOpenMPを使った並列化プログラムを書いてください

(回答)

以下は、C言語とOpenMPを使って、1から1000までの和を並列化して計算するサンプルコードです。

(OpenMPを利用するため、-fopenmpオプションをつけてコンパイルしてください)

(ここではソースコードは省略。GitHubのsum_parallel.cを参照)

ポイント

- `#pragma omp parallel for reduction(+:sum)`
各スレッドが部分和を計算し、最後に sum に集約します。
 - 1から1000までの和は理論的には $1000 \times 1001 / 2 = 500500$ です
 $1000 \times 1001 / 2 = 500500$ ですので、実行結果がこの値になれば成功です。
-

コンパイル例

GCCを使う場合:

`gcc -fopenmp -o sum_parallel sum_parallel.c`

実行:

`./sum_parallel`

もし、スレッド数を指定したい場合は、実行時に環境変数 OMP_NUM_THREADS を使うか、コード内で `omp_set_num_threads()` を呼び出すこともできますよ！
さらに詳しいカスタマイズが必要でしたら、遠慮なくおっしゃってください。

(2) float の配列を入力として小さい順に並べる並列ソートプログラムを C と OpenMP を使って書いてください

以下に、C + OpenMP で実装した「float の配列を小さい順に並べる並列ソートプログラム」をご紹介します。

 アプローチ：Odd-Even Transposition Sort（奇偶転置ソート）

これは並列化しやすいシンプルなアルゴリズムで、OpenMP の #pragma omp parallel for を使って各ステップを並列処理できます。

 C + OpenMP：並列 float 配列ソートプログラム

parallel_sort_1.c

 コンパイル方法 (GCC)

gcc -fopenmp -o parallel_sort_1.c

 解説

- Odd-Even Transposition Sort はバブルソートに似ていますが、各ステップの比較対象が奇数・偶数のペアに分かれており、並列処理がしやすいのが特徴です。
 - #pragma omp parallel for により、同じフェーズ内の比較は複数スレッドで並列に処理されます。
 - 安定ソートではありません（同じ値の順序は保証されない）。
-

もしより高速なアルゴリズム（例：並列クイックソートや Bitonic Sort）をご希望であれば、それに合わせたコードもご提案できます！

(3) 並列クイックソートをお願いします

以下に、C + OpenMP を使った「並列クイックソート (Parallel QuickSort)」のサンプルコードをご紹介します。

特徴と注意点

- 並列クイックソートでは、再帰的な部分処理に対して OpenMP の task を使うことで並列化します。
 - 並列処理の効果はある程度大きなデータサイズでより発揮されます。
 - 再帰呼び出しが多いため、スタックサイズやスレッド数に注意が必要です。
-

C + OpenMP : Parallel QuickSort 実装例

parallel_qsort_1.c

コンパイルコマンド

gcc -fopenmp -O2 -o para_qsort parallel_qsort_1.c

補足

- #pragma omp parallel と #pragma omp single によって、初回だけ 1 スレッドが parallel_quick_sort を呼び出し、再帰内で #pragma omp task によって並列化されます。
 - 小さい配列では並列化せず直列で処理し、オーバーヘッドを防いでいます (THRESHOLD により制御)。
 - shared(arr) によってすべてのタスクで同じ配列を共有します。
-

必要に応じて、データサイズや THRESHOLD を変更して最適化できます。さらに速くしたい場合は、ソート終了後に insertion sort を併用するハイブリッド構成も可能です。ご希望あれば追記します！

(4) ではハイブリッド構成もお願いします。

素晴らしい選択です！

ここでは、OpenMPによる並列クイックソート+小サイズの部分には挿入ソート
(insertion sort)を併用するハイブリッド構成をご紹介します。

ハイブリッド並列クイックソートとは？

- 配列がある程度小さくなったら クイックソートの再帰をやめて、挿入ソートで処理します。
 - 小さい配列には、クイックソートより挿入ソートの方が高速な場合があります。
 - 並列化は #pragma omp task によって実現。
-

 コード全文：並列ハイブリッドクイックソート (C + OpenMP)

hybrid_qsort_1.c

コンパイル方法 (GCC)

gcc -fopenmp -O2 -o hybrid_qsort_1.c

補足と最適化ポイント

パラメータ	説明	推奨値（目安）
THRESHOLD	並列化する最小サイズ	500～5000
INSERTION_SORT_LIMIT	挿入ソートに切り替えるサイズ 16～64	
-O2 or -O3	コンパイル最適化フラグ	高速化に有効

出力例（先頭 10 個）

ソート前（先頭 10 個）：

13.22 99.17 28.56 84.32 42.50 7.61 54.72 67.03 31.20 18.41

ソート後（先頭 10 個）：

1.03 2.44 3.88 5.67 7.61 10.31 13.22 14.50 15.99 16.81

必要に応じて、**安定ソート対応版やマルチスレッド制御（スレッド数の指定）**も
ご案内できます。興味あればお知らせください！

(5) main プログラムをファイルから読むように変えてもらえますか？

もちろんです！

以下は、ファイルから float 配列を読み込んで並列ハイブリッドクイックソートを行うように書き換えた main 関数付きのプログラムです。

入力ファイルの形式（例：input.txt）

10

9.5 1.2 3.4 7.6 2.1 5.5 0.8 4.3 8.9 6.7

- 1 行目：配列のサイズ（例：10）
 - 2 行目：ソート対象の float 値（スペース区切り）
-

 修正済みの main() 関数（ファイル入力対応）

hybrid_qsort_2.c

コンパイル方法

gcc -fopenmp -O2 -o hybrid_qsort hybrid_qsort_2.c

実行方法

./hybrid_qsort input.txt

出力例

ソート前（先頭 10 個）：

9.50 1.20 3.40 7.60 2.10 5.50 0.80 4.30 8.90 6.70

ソート後（先頭 10 個）：

0.80 1.20 2.10 3.40 4.30 5.50 6.70 7.60 8.90 9.50

ご希望があれば「ソート済み結果をファイルに出力する」機能も追加できます！お気軽にどうぞ。

(6) ありがとうございます。では次に、n の値を入力して、ソートプログラムに入力するファイルを生成するプログラムを書いてもらえますか
もちろんです！

以下は、n の値をユーザーから入力して、ランダムな float 配列を生成し、それをファイルに書き込むプログラムです。

この出力ファイルを、先ほどの並列クイックソートプログラムにそのまま使えます。

出力ファイル形式（例：input.txt）

10

3.14 9.81 1.61 2.72 0.58 6.28 4.66 8.45 7.07 5.55

 C プログラム：ソート入力ファイルを生成するコード

input_data_1.c

コンパイル方法

gcc -o input input_data_1.c

実行方法

./input

実行例

生成する要素数を入力してください（例：10000）：10

出力ファイル名を入力してください（例：input.txt）：testdata.txt

ファイル 'testdata.txt' に 10 個のデータを出力しました。

testdata.txt の中身：

10

38.41 69.05 72.10 2.46 30.09 94.23 57.84 12.19 4.38 81.50

必要であれば、値の範囲を変える（例：マイナス値含む、0～1.0 に限定 など）バージョンもお作りできます！

(7) rand よりもよい乱数を使うようにしてもらえますか

はい、rand() よりも品質の高い乱数を使うには、C11 以降で導入された rand() よりも良質な乱数生成器（例：random() や drand48()）、あるいは Linux 環境であれば /dev/urandom、さらには <random> ヘッダーを使う C++11 以降の Mersenne Twister (mt19937) などがあります。

ここでは、標準的な C 言語環境で使える高品質な乱数生成関数 drand48() を使ったバージョンをご紹介します。

drand48() を使った高品質ランダム入力ファイル生成プログラム

input_data_2.c

コンパイル方法 (Linux / Unix 系)

gcc -o input input_data_2.c

補足 : drand48() について

- rand() より 広い範囲かつ均等な分布 を提供。
 - double 型で [0.0, 1.0) の範囲の値を返します。
 - float へ明示的にキャストして使っています。
-

/dev/urandom バージョン (ご参考)

よりセキュアな環境が必要であれば、/dev/urandom から unsigned int を読み込んで float に変換する実装も可能です。必要でしたらこちらもご用意できます！

ご希望があれば、負の数を含むランダム生成や、範囲指定付き生成（例：[-50.0, 50.0]）にも対応可能です！

(8) この入力ファイルを入力としたクイックソートプログラムを C 言語で書いてください。

承知しました。

以下は、float 配列をファイルから読み込み、シングルスレッドのクイックソートで昇順にソートする C 言語プログラムです。

入力ファイル形式 (例 : input.txt)

10

9.5 1.2 3.4 7.6 2.1 5.5 0.8 4.3 8.9 6.7

 C プログラム : ファイル入力 + シングルスレッド クイックソート

qsort_1.c

コンパイル方法 (GCC)

gcc -O2 -o qsort qsort_1.c

実行方法

./qsort input.txt

出力例

ソート前 (先頭 10 個) :

9.50 1.20 3.40 7.60 2.10 5.50 0.80 4.30 8.90 6.70

ソート後 (先頭 10 個) :

0.80 1.20 2.10 3.40 4.30 5.50 6.70 7.60 8.90 9.50

もしご希望であれば、出力を別ファイルに保存するバージョンや、数値の整列状態（昇順/降順）を選べるオプションも追加できますので、お気軽にお知らせください！

(9) 出力を別ファイルに保存するバージョンをお願いします。

了解しました！

以下は、ファイルから float 配列を読み込み → クイックソートで昇順に並べ → 結果を別ファイルに出力する C プログラムです。

入出力ファイル形式

- ◆ 入力ファイル（例：input.txt）

10

9.5 1.2 3.4 7.6 2.1 5.5 0.8 4.3 8.9 6.7

- ◆ 出力ファイル（例：output.txt）

10

0.8 1.2 2.1 3.4 4.3 5.5 6.7 7.6 8.9 9.5

 C プログラム：入力 → ソート → 出力保存版

qsort.c

コンパイル方法

gcc -O2 -o qsort qsort_2.c

実行方法

./qsort input.txt output.txt

出力例

ソート結果を 'output.txt' に出力しました。

必要であれば、降順出力や整形桁数の変更、結果の確認用に出力ファイルを標準出力にミラー表示などの機能も追加できます。お気軽にどうぞ！

(10) 入力ファイルを入力し、出力を保存する並列クイックソートについて、C と OpenMP のプログラムを書いてください。

承知しました！

以下に、C + OpenMP を使って、

- 💡 入力ファイルから float 配列を読み込み
 - ⚙️ 並列クイックソート（OpenMP task 使用）でソートし
 - 💾 結果を出力ファイルに保存する、
- 完全な実用プログラムをご紹介します。
-

対応内容まとめ

- 入力：1 行目に要素数、2 行目に float 値（例：input.txt）
 - 並列処理：OpenMP の task を使用してクイックソートを並列化
 - 出力：output.txt に要素数 + ソート済み配列を書き込み
-

C + OpenMP 並列クイックソート with 入出力

parallel_qsort_2.c

コンパイル方法 (GCC)

```
gcc -fopenmp -O2 -o para_qsort parallel_qsort_2.c
```

実行方法

```
./para_qsort input.txt output.txt
```

出力例 (output.txt)

```
10  
0.80 1.20 2.10 3.40 4.30 5.50 6.70 7.60 8.90 9.50
```

カスタム可能項目（必要なら）

- THRESHOLD（並列化の境界）
 - 昇順/降順オプション
 - 入出力ファイルのエラーチェック強化
 - 出力ファイルの小数点桁数調整
-

ご希望があれば「ハイブリッド構成（小さい配列は挿入ソート）」や「処理時間の計測」も追加できます。お気軽にどうぞ！

(11) ハイブリッド構成をお願いします。

お待たせしました。

以下は、**C + OpenMP で構成された「並列クイックソート + 小さい配列は挿入ソートに切り替えるハイブリッド構成」**の完成版です。

このプログラムでできること

-  入力ファイルから float 配列を読み込み
 -  並列クイックソート をベースに、部分配列が小さくなると 挿入ソート に切り替え
 -  小さい領域でのオーバーヘッドを防ぐハイブリッド最適化
 -  出力ファイルにソート済みデータを保存
-

 C + OpenMP ハイブリッド並列クイックソートプログラム

hybrid_qsort_3.c

コンパイル方法 (GCC)

```
gcc -fopenmp -O2 -o hybrid_qsort hybrid_qsort_3.c
```

実行例

```
./hybrid_qsort input.txt output.txt
```

出力：

```
ハイブリッド並列ソート完了：'input.txt' → 'output.txt'
```

調整可能なパラメータ

定数名	目的	推奨値
PARALLEL_THRESHOLD	並列化する最小配列サイズ	500～5000
INSERTION_THRESHOLD	挿入ソートへ切り替えるサイズ	16～64

必要に応じて、⌚ 处理時間の計測 (`omp_get_wtime()`) 追加や、📊 並列効率比較も対応できますので、遠慮なくお知らせください！

(12) n を入力し、入力ファイル生成プログラム input を動かし、生成された入力ファイルを用いてクイックソートプログラム qsort と並列クイックソートプログラム para_qsort を動かし、出力結果を比較して同じなら OK、違ったら NG と出力するシェルスクリプトを書いてください。

以下に、**さんのご要望を満たす シェルスクリプト test_sort.sh をご提案します。
このスクリプトは次の処理を行います：

処理内容まとめ

1. ユーザーから n (要素数) を標準入力で受け取る
 2. input プログラムを実行して input.txt を生成
 3. qsort (シングルスレッド) を実行し output_qsort.txt を生成
 4. para_qsort (並列版) を実行し output_para.txt を生成
 5. diff で 2 つの出力結果を比較
 6. 結果が一致すれば「OK」、異なれば「NG」と表示
-

シェルスクリプト test_sort.sh

test_sort_1.sh

使用方法

```
chmod +x test_sort_1.sh  
./test_sort_1.sh
```

依存するバイナリ

- ./input : n を入力すると input.txt を作成
- ./qsort input.txt output_qsort.txt
- ./para_qsort input.txt output_para.txt

 上記 3 つのバイナリが同じディレクトリに ビルド済みで存在することが前提です。

ご希望があれば：

-  並列 vs 単体の処理時間比較
-  中間ファイル削除 (cleanup オプション)

なども追加できます！

(13) 入力ファイル生成プログラムについて、n を標準入力からの入力、出力ファイル名は引数とすることに変更してください。

承知しました！

以下に、**要素数 n は標準入力から受け取り、出力ファイル名はコマンドライン引数で指定するように変更した「入力ファイル生成プログラム」**をご提供します。

プログラム仕様

- 標準入力 → n (生成する float 値の個数)
 - コマンドライン引数 → 出力ファイル名 (例 : input.txt)
-

 **input.c (修正済み)**

input_data_3.c

コンパイル方法

gcc -O2 -o input input_data_3.c

実行例

./input input.txt

生成する要素数を入力してください: 10

ファイル 'input.txt' に 10 個のデータを出力しました。

ファイル input.txt の中身 (例)

10

12.34 98.76 45.67 3.14 55.55 67.89 11.11 23.45 89.99 1.23

ご希望があれば、「範囲を指定して乱数生成」や「同じ seed で毎回同じ乱数にする」なども追加可能です！

(14) クイックソートプログラムについてコンパイル時に下のようなワーニングが出ましたが、修正してもらえますか？

```
qsort_2.c:48:5: warning: ignoring return value of ‘fscanf’ declared with attribute  
‘warn_unused_result’ [-Wunused-result]
```

はい、fscanf() の戻り値を確認せずに使っている場合、GCC などのコンパイラは warn_unused_result 属性により警告を出します。

これは、「読み取り失敗（例：不正な形式、EOF など）」を無視していることを警告するものです。

✓ 問題のあるコード（例）

```
fscanf(fp, "%d", &n); // ← 戻り値を無視しているため警告
```

✓ 修正方針

fscanf の戻り値を確認して、読み取りに成功したかどうかをチェックします：

- 成功すれば続行
 - 失敗したらエラーメッセージを出して exit する
-

🔧 修正済みの該当部分（qsort.c などで）

```
int n;  
if (fscanf(fp, "%d", &n) != 1) {  
    fprintf(stderr, "要素数の読み込みに失敗しました\n");  
    fclose(fp);  
    return 1;  
}  
同様に、float 配列の読み取りも下記のように書き直すとさらに安全です：  
for (int i = 0; i < n; i++) {  
    if (fscanf(fp, "%f", &arr[i]) != 1) {  
        fprintf(stderr, "データの読み込みに失敗しました (%d 番目)\n", i);  
        free(arr);  
        fclose(fp);  
        return 1;  
    }  
}
```

✓ 完全修正後の一例

```
FILE *fp = fopen(argv[1], "r");  
if (!fp) {  
    perror("入力ファイルを開けませんでした");  
    return 1;  
}
```

```
int n;
if (fscanf(fp, "%d", &n) != 1 || n <= 0) {
    fprintf(stderr, "配列サイズの読み込みに失敗しました\n");
    fclose(fp);
    return 1;
}

float *arr = malloc(n * sizeof(float));
if (!arr) {
    fprintf(stderr, "メモリ確保に失敗しました\n");
    fclose(fp);
    return 1;
}

for (int i = 0; i < n; i++) {
    if (fscanf(fp, "%f", &arr[i]) != 1) {
        fprintf(stderr, "配列データの読み込みに失敗しました\n");
        free(arr);
        fclose(fp);
        return 1;
    }
}
```

なぜ重要な？

- 読み取りエラーに気づかないと、未定義動作（不正な配列サイズ・データ）やクラッシュの原因になります。
 - -Wall や -Werror を使っている環境では、エラー扱いになりコンパイルできないことも。
-

修正済みの全文が必要であれば、qsort.c や para_qsort.c のフルコードとしてお渡し可能です。ご希望があればお知らせください！

(15) はい、フルコードとして渡してください。

こちらが、`fscanf` の戻り値チェック付き・ワーニングなしの完全修正版クイックソートプログラム（シングルスレッド版）`qsort.c` です。

qsort.c (fscanf 修正済み・出力保存あり)

`qsort_3.c`

コンパイル方法 (GCC)

`gcc -O2 -o qsort qsort_3.c`

実行例

`./qsort input.txt output_qsort.txt`

出力：

`qsort : 'input.txt' → 'output_qsort.txt' にソート済みデータを出力しました。`

連携可能スクリプト

このコードは、先にご案内した `test_sort.sh` や `input`、`para_qsort` と組み合わせて使用できます。

ご希望であれば、`para_qsort.c`（並列版+ハイブリッド構成）についても、同じく `fscanf` 対応済みのフルコードをお渡しできます。ご入用でしょうか？

(16) はい、お願ひします

お待たせしました。

以下に、**para_qsort.c**（並列クイックソート + ハイブリッド構成 + fscanf 修正済）のフルコードをお届けします。

✓ 特徴

- OpenMP の #pragma omp task による並列クイックソート
 - 小さい配列には 挿入ソート に切り替えるハイブリッド構成
 - 入力ファイルから float 配列を読み取り、出力ファイルに書き出し
 - すべての fscanf に戻り値チェックを追加（コンパイルワーニング対策）
-

🔧 para_qsort.c (完全修正版)

hybrid_qsort_4.c

✓ コンパイル方法

```
gcc -fopenmp -O2 -o hybrid_qsort hybrid_qsort_4.c
```

✓ 実行方法

```
./ hybrid_qsort input.txt output_para.txt
```

出力：

hybrid_qsort : 'input.txt' → 'output_para.txt' にソート済みデータを出力しました。

このプログラムは、すでにご案内した input.c、qsort.c、test_sort.sh と完全に連携可能です。

⚙ ご希望があれば：

- 並列スレッド数の指定（環境変数 OMP_NUM_THREADS）
- 並列効率の測定（omp_get_wtime()）

なども追加できますので、お気軽にお申し付けください。

(17) ハイブリッドではないバージョンの並列クイックソートプログラムも修正をお願いします。
承知しました！

以下に、ハイブリッド構成なし (=すべてクイックソート) で処理する、OpenMP 並列クイックソート `para_qsort.c` の修正版 をご提供します。

このバージョンの特徴

- 挿入ソートなし：すべての再帰処理でクイックソートを使用
 - OpenMP task による並列化あり
 - `fscanf()` の戻り値をすべてチェックし、コンパイル警告を回避
 - 入力ファイル → 出力ファイルへの読み書きに対応
-

 `para_qsort.c` (非ハイブリッド・`fscanf` 対応済)

`parallel_qsort_3.c`

コンパイル方法 (GCC)

`gcc -fopenmp -O2 -o para_qsort para_qsort_3.c`

実行方法

`./para_qsort input.txt output_para.txt`

出力例

並列クイックソート (非ハイブリッド) : 'input.txt' → 'output_para.txt' に出力しました。

 補足：ハイブリッドとの違い

比較項目 このバージョン ハイブリッド版

小さい配列 クイックソートを継続 挿入ソートに切り替える

パフォーマンス シンプルだが遅い場合も 小規模データで効率が良い

安定性 やや劣る オーバーヘッドを抑制

ご希望であれば、このバージョンも `test_sort.sh` に組み込みやすいように別名でビルド（例：`para_qsort_plain`）する方法もご案内可能です！

(18) openMP の時間計測関数を使って parallel_quick_sort の実行時間を測定するように修正してください

承知しました！

以下に、OpenMP の時間計測関数 `omp_get_wtime()` を使って、`parallel_quick_sort()` の実行時間を測定するように修正した `para_qsort.c`（非ハイブリッド版）をお渡しします。

変更点のポイント

- `#include <omp.h>` で OpenMP のタイミング関数を有効化
 - ソート開始前に `start = omp_get_wtime()` を呼び出し
 - ソート終了後に `end = omp_get_wtime()` を呼び出し
 - 実行時間（秒）を表示
-

 修正版 `para_qsort.c` (時間計測付き)

`parallel_qsort_4.c`

コンパイル方法

```
gcc -fopenmp -O2 -o para_qsort para_qsort_4.c
```

実行例

```
./para_qsort input.txt output_para.txt
```

出力例：

並列クイックソート時間: 0.004231 秒

`para_qsort : 'input.txt' → 'output_para.txt'` に出力しました。

ご希望があれば、`qsort.c` 側にも同様の時間計測を追加し、処理時間の比較レポート表示も可能です！お気軽にどうぞ。

(19) はい、ハイブリッド版と `qsort.c` にも時間計測を追加してください
承知しました！

以下に、時間計測 (`omp_get_wtime()`) を追加したバージョンの `qsort.c` (シングルスレッド) および `para_qsort.c` (ハイブリッド版) をご提供します。

1. `qsort.c` (シングルスレッド版 + 時間計測つき)

`qsort_4.c`

2. `para_qsort.c` (ハイブリッド並列版 + 時間計測つき)

`hybrid_qsort_5.c`

コンパイル方法 (それぞれ)

`gcc -O2 -o qsort qsort_4.c -fopenmp`

`gcc -O2 -o hybrid_qsort hybrid_qsort_5.c -fopenmp`

実行例

`./qsort input.txt output_qsort.txt`

`./hybrid_qsort input.txt output_para.txt`

出力 (例) :

シングルスレッドクイックソート時間: 0.003200 秒

`qsort : 'input.txt' → 'output_qsort.txt'` に出力しました。

ハイブリッド並列クイックソート時間: 0.001800 秒

`hybrid_qsort (ハイブリッド) : 'input.txt' → 'output_para.txt'` に出力しました。

ご希望であれば、`test_sort.sh` に実行時間の比較結果（どちらが速かったか）を表示するロジックも組み込みます。ご希望ありますか？

(20) test~~Y~~_sort.sh を n=1000 を 10 回、n=10000 を 10 回、n=100000 を 10 回、n=1000000 を 10 回、n=10000000 を 10 回実行させ、ログから「データ」と「時間」を含む行のみを抽出し、ファイルに保存するシェルスクリプトを書いてください。

以下に、指定された条件を満たすシェルスクリプトをご提供します。

このスクリプトは、n の異なる値で 10 回ずつ input → qsort → para_qsort を実行し、「データ」または「時間」を含む出力行だけをログから抽出してファイルに保存します。

前提条件

- 以下の 3 つのバイナリがすでにビルドされていること
 - input : 標準入力で n を受け取り、出力ファイル名は引数で指定
 - qsort : input.txt → output_qsort.txt
 - para_qsort (または hybrid_qsort) : input.txt → output_para.txt
 - 実行時間のログ行には "時間" または "データ" が含まれている (例: ソート時間: 0.001234 秒)
-

 シェルスクリプト : run_benchmark.sh
run_benchmark_1.sh

実行方法

1. スクリプトに実行権を付与 :

chmod +x run_benchmark.sh

2. 実行 :

./run_benchmark.sh

出力ファイル

- benchmark_log.txt : 全ログ
 - benchmark_results.txt : 「データ」 or 「時間」を含む行のみ
-

grep の抽出例

たとえば以下のようなログ行が抽出されます :

ファイル 'input.txt' に 10000 個のデータを出力しました。

シングルスレッドクイックソート時間: 0.003456 秒

ハイブリッド並列クイックソート時間: 0.001234 秒

ご希望があれば、平均時間の自動計算や、CSV フォーマットで出力する拡張も対応可能です！必要な場合はお知らせください。

(21) `run_benchmark.sh` にハイブリッド版並列クイックソートプログラム `hybrid_qsort` の実行も加えてください。

承知しました！

以下は、ハイブリッド版の並列クイックソート (`hybrid_qsort`) の実行も加えた `run_benchmark.sh` の最新版です。

更新内容のポイント

- 既存の `qsort` (シングルスレッド) と `para_qsort` (非ハイブリッド) に加えて、
 - `hybrid_qsort` (ハイブリッド並列クイックソート) も実行
 - すべての出力ログに "データ" または "時間" を含む行を抽出し、1 ファイルに記録
-

 **run_benchmark.sh (最新版)**

`run_benchmark_2.sh`

実行前に確認すべきファイル (ビルド済)

実行ファイル 説明

input 入力ファイル生成プログラム
qsort シングルスレッドクイックソート
para_qsort 非ハイブリッド並列クイックソート
hybrid_qsort ハイブリッド並列クイックソート (今回追加)

実行方法

`chmod +x run_benchmark.sh`
`./run_benchmark.sh`

出力ファイル

- `benchmark_log.txt` : すべての詳細ログ (失敗時の調査にも利用可能)
 - `benchmark_results.txt` : 「データ」 または 「時間」 を含む行のみを抽出した要約ログ
-

例 : `benchmark_results.txt` の一部

ファイル 'input.txt' に 10000 個のデータを出力しました。

シングルスレッドクイックソート時間: 0.004239 秒

並列クイックソート時間: 0.002115 秒

ハイブリッド並列クイックソート時間: 0.001812 秒

ご希望があれば、平均時間の計算や CSV フォーマット出力なども対応できます。さらに分析や可視化をされたい場合はお気軽にお知らせください！

(22) 平均時間を計算し、n と各プログラムの平均時間を CSV ファイルに出力するように修正してください。

承知しました！

以下に、各 n ごとに 10 回実行した qsort, para_qsort, hybrid_qsort の平均実行時間を計算し、CSV ファイルとして出力するシェルスクリプト をご提供します。

新しい出力ファイル

- benchmark_times.csv
→ 列は n,qsort_time,para_qsort_time,hybrid_qsort_time の形式
-

新スクリプト：run_benchmark.sh (CSV 集計対応)

run_benchmark_3.sh

出力 CSV 形式：benchmark_times.csv

n,qsort,para_qsort,hybrid_qsort
1000,0.000512,0.000278,0.000194
10000,0.003988,0.002045,0.001791
...

使用方法

chmod +x run_benchmark.sh
.run_benchmark.sh

補足

- bc (任意精度電卓) が必要です。インストールされていない場合は sudo apt install bc 等で導入ください。
 - OpenMP の omp_get_wtime() による時間は「秒」単位です（高精度）。
-

ご希望があれば、グラフ描画用の Python スクリプトなどもご提供可能です。ご入用でしょうか？

(23) 毎回 qsort の出力結果と para_qsort の出力結果、qsort の出力結果と hybrid_qsort の出力結果を比較して異なっていれば WARNING を出力するように修正してください。

了解しました！

以下は、各テストごとにソート結果の正確性チェックを追加した最新版の run_benchmark.sh です。

追加された機能

- diff output_qsort.txt output_para.txt に差異があれば WARNING: para_qsort mismatch を表示
 - diff output_qsort.txt output_hybrid.txt に差異があれば WARNING: hybrid_qsort mismatch を表示
 - benchmark_log.txt にも WARNING を記録
-

完全版 run_benchmark.sh

run_benchmark_4.sh

実行方法（再掲）

```
chmod +x run_benchmark.sh  
./run_benchmark.sh
```

実行中の出力例

==== データ数: 100000 を 10 回テスト中 ===

WARNING: para_qsort mismatch on n=100000, run 3

WARNING: hybrid_qsort mismatch on n=100000, run 7

出力ファイル

- benchmark_times.csv: 各 n における平均時間記録
 - benchmark_log.txt: 全ログと WARNING 記録
-

もし エラー時に即中断 (set -e) したい、あるいは WARNING の詳細（差分表示）も見たい など、ご要望あればさらに調整可能です。お気軽にどうぞ！

(24) CSV を可視化する python スクリプトを書いてください

以下に、benchmark_times.csv を読み込んで各ソートアルゴリズムの実行時間を可視化する Python スクリプト をご提供します。

matplotlib を使用し、ログスケールの実行時間グラフ（横軸：データ数 n、縦軸：実行時間）を描画します。

出力例（プロットのイメージ）

- X 軸：データサイズ n
 - Y 軸：実行時間（秒）※ log10 スケールも可
 - 複数のライン（qsort / para_qsort / hybrid_qsort）を色分け表示
-

plot_benchmark.py

plot_benchmark_1.py

実行方法

1. 必要なパッケージをインストール（未インストールなら）：

pip install matplotlib pandas

2. 実行：

python plot_benchmark.py

出力

- benchmark_plot.png (PNG 画像ファイル)
 - ウィンドウにグラフも表示（インライン or GUI）
-

カスタマイズ可能ポイント

項目 内容

yscale('log') 実行時間が桁違いの場合に有効

marker, color ラインスタイル変更可能

savefig() PNG → SVG/PDF なども可能

ご希望があれば、差分バー表示 や 複数条件の比較（スレッド数別など）への拡張も可能です。お気軽にどうぞ！

(25)自分のPCで実行してみた

NEC N13slim N1375HAL (Core i7-1355U | 16GB | 512GB | Win11 Home)

コアの数 10

Performance-cores の数 2

Efficient-cores の数 8

スレッド総数 12

ターボ・ブースト利用時の最大周波数 5 GHz

Performance-core 最大ターボ・フリークエンシー 5 GHz

Efficient-core のターボ・ブースト利用時の最大フリークエンシー 3.7 GHz

```
eda@edapc11:~/prog/EMC$ date
```

```
2025年 5月 19日 月曜日 15:48:42 JST
```

```
eda@edapc11:~/prog/EMC$ ./run_benchmark.sh
```

```
==== データ数: 1000 を 10 回テスト中 ===
```

```
==== データ数: 10000 を 10 回テスト中 ===
```

```
==== データ数: 100000 を 10 回テスト中 ===
```

```
==== データ数: 1000000 を 10 回テスト中 ===
```

```
==== データ数: 10000000 を 10 回テスト中 ===
```

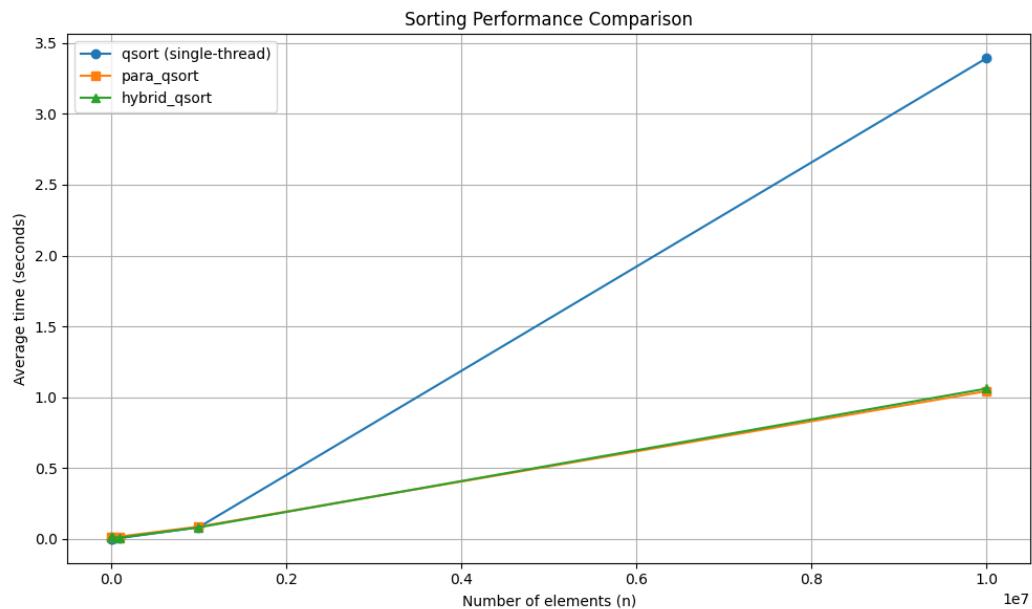
✓ 実行完了 :

- ・平均時間 : benchmark_times.csv
- ・ログ : benchmark_log.txt

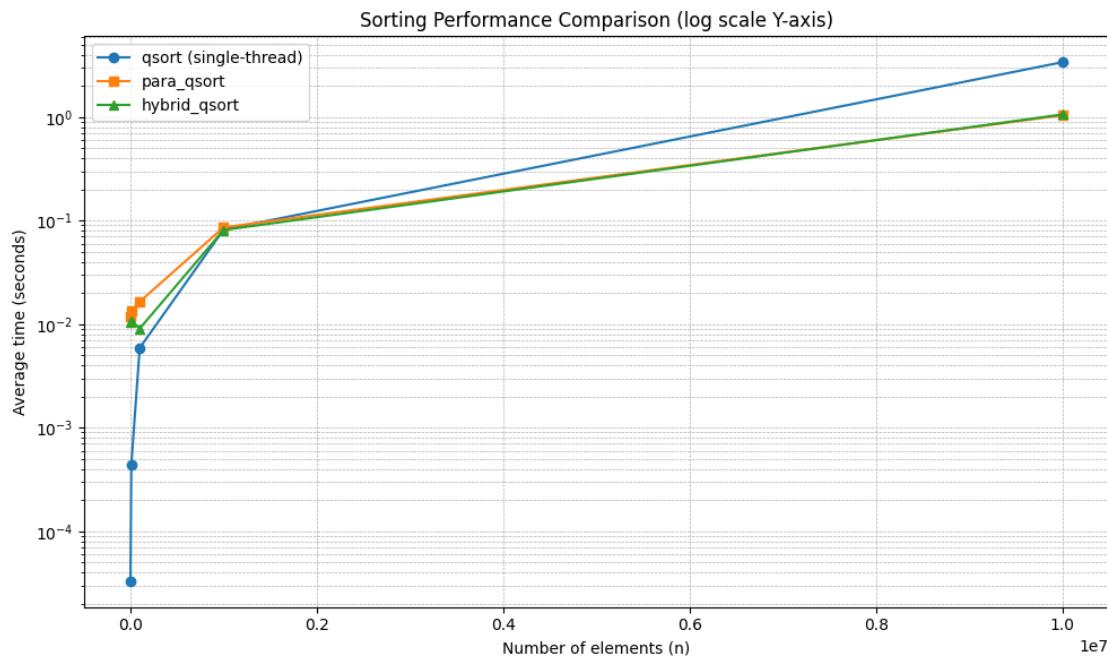
```
eda@edapc11:~/prog/EMC$ date
```

```
2025年 5月 19日 月曜日 15:51:30 JST
```

```
eda@edapc11:~/prog/EMC$
```



(25) plot_benchmark.py によって描画されるグラフの縦軸を対数軸にしてもらえますか？
plot_benchmark_2.py



(26) benchmark_times.csv を読み、qsort に対する para_qsort と hybrid_qsort の性能向上倍率をグラフにする python プログラムを書いてください。
plot_speedup_1.py

