1. 実験方法

以降のすべての実験における計算環境を表1.1に示す. 本実験中においてはOSやコンパイラのアップデートを行わないように十分注意する. なお, 実験に使用したプログラムは, レポート末尾に参考として示しておく.

表1.1 計算環境

項目名	バージョン, 製品名
プロセッサ	1.6 GHz Intel Core i5
メモリ	4 GB 1600 MHz DDR3
OS	macOS High Sierra ver. 10.13.6
Cコンパイラ	gcc (Homebrew GCC 5.5.0_2) 5.5.0

1.1 2次元配列の左回転

n×nの2次元配列src[n][n]の左回転回転を実行して結果をdst[n][n]に格納する関数(以降，右回転関数という)を複数作成し, 性能をCPE：Cycles Per Elementとして取得した. 右回転関数は愚直に実行を行うもの, ループアンローリングを施したもの, ブロック化を施したもの, 斜めに配列をアクセスしていくものの4つを作成しようと考えた．2次元配列のサイズや型，ブロックサイズ，ループアンローリングの回数を様々変えて実験を行うため, まずディレクトリ階層を図1.1のようにした.

mainはsrc[n][n]を初期化してから， ループの中で右回転関数を呼び出して性能測定結果を出力してdstを使用するという処理を性能測定実行本体main.cを含むディレクトリである. 2次元配列srcとdstはスタックフレームの中ではなく， プロセス実行開始前にデータエリアに確保しておかないとセグメンテーションフォルトを起こしてしまう関係上， 2次元配列のサイズを変えるにはmain.cを毎回書き換えて実行せざるを得なかった. そこで, 2次元配列のサイズを指定すると指定通りのサイズのsrc[n][n]とdest[n][n]を大域変数として確保したメインプログラムmain.cを出力するプログラムmake_main.cを書き, 実験を効率化した.

```
.
├── block
│   ├── O1.out
│   ├── O2.out
│   ├── Og.out
│   ├── make_rotate.c
│   ├── maker.out
│   ├── no_op.out
│   └── rotate.c
├── main
│   ├── clock.h
│   ├── clock64.o
│   ├── main.c
│   ├── make_main.c
│   ├── maker.out
│   ├── my_type.h
│   ├── rotate.h
│   └── rotate_helper.c
├── naive
│   ├── O1.out
│   ├── O2.out
│   ├── Og.out
```

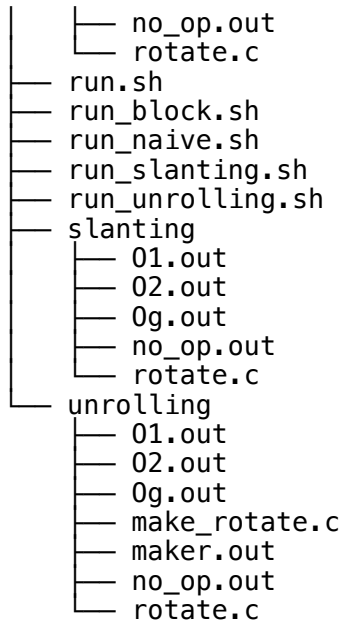

図1.1 本実験におけるディレクトリ階層