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. また, Turbo Boost Switcher for OS Xを用いてターボブーストを抑制しCPUのクロック周波数上限が規格の1.6GHzとなるようにする. 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.

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，左回転関数という)を複数作成し, 性能をCPE：Cycles Per Elementとして取得する. 左回転関数は次の4つを作成する：

- 愚直に実行を行うもの(以下, naiveという)
- ブロック化を施したもの(以下, blockという)
- ブロック化したのち，各ブロックの処理のループにアンローリングを施したもの(以下, block_unrollingという)
- ループアンローリングを施したもの(以下, unrollingという)

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ディレクトリは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大きすぎて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次元配列のサイズと型を指定すると配列の型を提供するヘッダファイルmy_type.h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.

naiveディレクトリは，サンプルとして配布された愚直な左回転プログラムが1つ入っている. blockディレクトリには，ブロックサイズを指定するとブロック化を施した左回転プログラムを出力するCプログラムmake_rotate.cとその出力であるrotate.cが入っている．block_unrollingディレクトリには，ブロックサイズを指定するとブロック化をしたのち各ブロックの処理のループをアンローリングした左回転プログラムを出力するCプログラムmake_rotate.cとその出力であるrotate.cが入っている．unrollingディレクトリには，アンローリングの回数を指定するとループアンローリングを施した左回転プログラムを出力するCプログラムmake_rotate.cとその出力であるrotate.cが入っている.

さて, 本実験には次の5つのパラメーターがあることに注意してほしい：

- 左回転プログラムの種類
- コンパイラによる最適化レベル
- 2次元配列のサイズ n
- 2次元配列の型
- ブロックサイズおよびループアンローリングの回数

実験1.1 CPE分布の測定

naiveについて, CPEを50回測定し, その分布を取得する. ここで，コンパイラによる最適化オプションは-O2, 2次元配列の大きさは2048, 型はintとする.

実験1.2 コンパイラ最適化レベルによる最小CPEの変化測定

4種類の左回転プログラムそれぞれについて, コンパイラ最適化オプションを-O0, -O1,-O2, -O3と変化させて, CPEを50回測定し, その最小値をそれぞれ取得する. ここで, 2次元配列の大きさは2048, 型はint, ブロックサイズとループアンローリングの回数は8とする.

実験1.3 2次元配列のサイズ, 型による愚直なプログラムの最小CPEの変化測定

naiveについて, 2次元配列のサイズnを64, 128, 256, 512, 1024, 2048, 4096, 型をchar, int, long, doubleと変化させて, CPEを50回測定し, その最小値をそれぞれ取得する. ここで, コンパイラによる最適化オプションは-O2とする.

実験1.4 2次元配列のサイズ, 型とブロックサイズによる最小CPEの変化測定

ブロック化を行なっているblockとblock_unrollingの2つの左回転プログラムについて, 2次元配列のサイズnを64, 128, 256, 512, 1024, 2048, 4096, 型をchar, int, long, double, ブロックサイズを2, 4, 8, 16, 32, 64と変化させて, CPEを50回測定し, その最小値をそれぞれ取得する. ここで, コンパイラによる最適化オプションは-O2する.

実験1.5 2次元配列の型とアンローリング回数による最小CPEの変化測定

unrollingについて, 型をchar, int, long, double, 2次元配列のサイズnを64, 128, 256, 512, 1024, 2048, 4096, アンローリング回数を2, 4, 8, 16, 32, 64と変化させて, CPEを50回測定し, その最小値をそれぞれ取得した. ここで, コンパイラによる最適化オプションは-O2とする.

実験1.6 配列のサイズ n が2のべき乗でない場合の最小CPEの変化測定

ループアンローリングを行なっていないnaiveとblockの2つの左回転プログラムについて, 2次元配列のサイズnを63, 64, 65, 127, 128, 129, 255, 256, 257, 511, 512, 513, 1023, 1024, 1025, 2047, 2048, 2049, 4095, 4096, 4097と変化させて, CPEを50回測定し, その最小値をそれぞれ取得する. ここで, コンパイラによる最適化オプションは-O2, 2次元配列の型はintとする. また, blockにおけるブロックサイズは, 2のべき乗の測定点については実験1.4で最小CPEとなった値を採用し, それ以外では最も近い2のべき乗の測定点のものを採用する.

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
コア単位の2次キャッシュ	256 KB
3次キャッシュ	3 MB
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

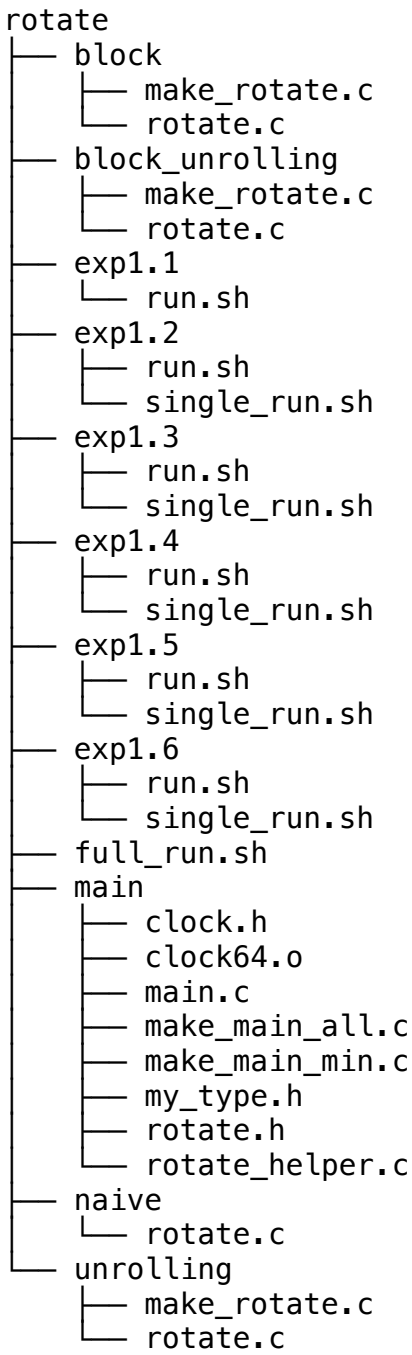

図1.1 本実験のディレクトリ階層

2. 実験結果

全ての実験において, 測定したCPE値は小数点以下4桁まで採用した.

2.1 CPE分布の測定

naiveのCPEを50回測定した結果をCSVファイルとして書き出し, CPEの幅1ごとの度数分布表とヒストグラムを作成した. それらをそれぞれ表2.1および図2.1に示す. ただしヒストグラムにおいてCPE区分値がkのところには, CPEのうち(k-1)より大きくk以下であるようなデータの個数を表記している. ヒストグラムは対称形にはならず, 最小値付近に度数が集中していて一部極端に大きな値が外れ値として存在しているような形状となった.

また, これら50回の測定におけるCPEの最小値, 平均値, 最大値はそれぞれ22.0880, 24.8202, 37.1132であった. CPEの分布の中で度数が多かったのは, 最大がCPEが23より大きく24以下の区分の17, その次ががCPEが22より大きく23以下の区分の12であった

2.2 コンパイラ最適化レベルによる最小CPEの変化測定

4つの左回転プログラムをO0からO3までの異なる最適化レベルでコンパイルしたのち, 50回実行した時のCPEの最小値を測定した. その結果を表2.2に, 表2.2に基づいて作成した最適化レベルと最小CPEの関係をプロットしたグラフを図2.2に示す.

最適化レベルO0では, naive, unrolling, block, block_unrollingの順に最小CPEが小さくなり, naiveが他の3つと比較して特に最小CPEが大きくなった.

O1以上の最適化レベルでは, 全体としてnaiveとunrolling, そしてblockとblock_unrollingの2つの組みにCPE値が分かれた. naiveとunrollingのCPEの差は大きく縮まり, これらの間のO1以上での最小CPEの差の最大値は0.3075となった. blockとblock_unrollingの最小CPEの差は縮まるどころか逆転し, O1以上の最適化レベルではblockの方が最小CPEが小さくなった. 次に, O2, O3では4つの左回転プログラム全てについてCPE値の大幅な改善は見られなかった. それどころか, block_unrolling以外の3つの左回転プログラムでは, 最小CPEがO2のときよりもO3のときの方が悪くなった.

2.3 2次元配列のサイズ, 型による愚直なプログラムの最小CPEの変化測定

naiveについて, 2次元配列のサイズ nと型を様々に変えて, 50回実行した時のCPEの最小値を測定した. その結果を表2.3から表2.6に示す.

char型のn = 64の場合を除いて, 配列のサイズが等しければ, 1つの要素が占めるメモリサイズが小さいほど最小CPEは小さくなった. 同じ8バイト変数であるlongとdoubleの間では, longの方が最小CPEが小さい配列のサイズが3種類, doubleの方が最小CPEが小さい配列のサイズが4種類であった. また, char型のn = 64とn = 128の間を除いて, 配列の型によらず, 2次元配列のサイズnが大きくなるほど, CPEは大きくなった. ここで得られた値を基準として, どの程度高速化できたのかに注目していく.

2.4 2次元配列のサイズ, 型とブロックサイズによる最小CPEの変化測定

blockとblock_unrollingについて, 次元配列のサイズ nと型, ブロックサイズを様々に変えて, 50回実行した時のCPEの最小値を測定した. blockの結果を表2.7から表2.10に, block_unrollingの結果を表2.11から表2.14に示す. 表2.7から表2.14には次元配列のサイズ, 型を固定しブロックサイズを変えた時のCPEの最小値も合わせて提示している. また, それぞれでCPEが最小となるブロックサイズがわかりやすいように, 最小となったデータは赤色で表記している.

まず, blockついてのべる. char型のn = 64, 128の場合を除いて, 配列のサイズが等しければ, 1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった. 同じ8バイト変数であるlongとdoubleの間では, longの方が最小の最小CPEが小さい配列のサイズが5種類, doubleの方が最小の最小CPEが小さい配列のサイズが2種類であった. ブロックサイズについては, 型によらず全体として, ブロックサイズを変えていくと赤色の最小の最小CPEに近いほど最小CPEの値が小さくなるような傾向が見られた. また, double型, n = 2048の場合を除いて, 2次元配列のサイズnが大きくなるほど, 最小の最小CPEを与えるブロックサイズは小さくなった.

次に, block_unrollingについて述べる. char型のn = 64の場合を除いて, 配列のサイズが等しければ, 1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった. longとdoubleの間では, longの方が最小の最小CPEが小さい配列のサイズが4種類, doubleの方が最小の最小CPEが小さい配列のサイズが3種類であった. ブロックサイズについては, 型によらず全体として, ブロックサイズを変えていくと赤色の最小の最小CPEに近いほど最小CPEの値が小さくなるような傾向が見られた. 2次元配列のサイズnと最小の最小CPEを与えるブロックサイズの間には, 特徴的な傾向は見られなかった.

blockとblock_unrollingを比較してみる. また, char型のn = 64とn = 128の間を除いて, 配列の型によらず, 2次元配列のサイズnが大きくなるほど, 最小CPEは大きくなるという傾向は, blockもblock_unrollingも変わらなかった. ブロックサイズごとに見てみると, ブロックサイズが2のところではblockよりもblock_unrollingの方が最小CPEが小さいが, ブロックサイズが64のところでは, その逆となった.

2.5 2次元配列の型とアンローリング回数による最小CPEの変化測定

unrollingについて, 2次元配列のサイズ nと型, アンローリングの回数を様々に変えて, 50回実行した時のCPEの最小値を測定した. その結果を表2.15から表2.18に示す. 表2.15から表2.18には次元配列のサイズ, 型を固定しブロックサイズを変えた時のCPEの最小値も合わせて提示している. また, それぞれでCPEが最小となるブロックサイズがわかりやすいように, 最小となったデータは赤色で表記している.

char型のn = 64の場合を除いて, 配列のサイズが等しければ, 1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった. longとdoubleの間では, longの方が最小の最小CPEが小さい配列のサイズが3種類, doubleの方が最小の最小CPEが小さい配列のサイズが4種類であった. アンローリングの回数については, ブロック化のように最小の最小CPEを起点とした傾向は見られなかった. また, 2次元配列のサイズnを変えたときの, 最小の最小CPEを与えるループアンローリングの回数は不規則であった. さらに, 2次元配列のサイズが大きくなるほど, ループアンローリングの回数を変更したときのCPEの変化は小さくなる傾向にあった. 例えば, long型の場合, 最小の最小CPEと最大の最小CPEの差は, n = 64のときには2.7041なのに対し, n = 4096のときには1.1713であった.

2.6 4つの左回転関数の最小の最小CPEの比較

各左回転関数によって得られた最小CPEの測定結果である表2.3から表2.18までのうち, 最小の最小CPEまとめたものを表2.19から表2.22に示す. 表2.19から表2.22には, 高速化率として, naiveの最小CPEをそれぞれの最小の最小CPEで割った値も合わせて提示している. 小数点以下4桁までの提示として統一をした. また, 表2.19から表2.22までに基づいて得られた, nは2次元配列のサイズnと最小の最小CPEの関係をプロットした片対数グラフを図2.3から図2.6に示す.

図2.3から図2.6に注目する．型によらず，全体としてnaiveとunrolling，そしてblockとblock_unrollingの2つの組に最小の最小CPEが分かれた．naiveとunrollingについて述べる．これらは型によらず，対数グラフ上で log n = 6からlog n = 9までは最小の最小CPEは下に凸な変化をし，log n = 9からlog n = 12までは上に凸な変化をしていた．全体として，naiveよりもunrollingの方がわずかに最小の最小CPEは小さくなった．次に，blockとblock_unrollingについて述べる．charの場合には，log n = 6，つまりn = 64の時を除いて片対数グラフ上で直線的な変化をした．intの場合には，測定点全体で直線的な変化をした．一方，longとdoubleの場合には，log n = 6からlog n = 8まで，およびlog n = 10からlog n = 12までは対数グラフ上で最小の最小CPE直線的に変化をしたが，log n = 8とlog n = 9の間はほとんど変化せず，log n = 9とlog n = 10の間では大きな跳躍があった．blockとblock_unrollingの間では，どちらの方が最小の最小CPEが小さいといった特徴は見られなかった．

表2.19から表2.22の高速化率に注目する．まず，blockとblock_unrollingについて，char型のn = 64，128，int，long，double型のn = 64のときにはそれ以上の配列サイズのときに比べて極端に高速化率が小さくなった．特に，char型，n = 128でblockを実行したときには，高速化率が0.7124と大きく1を下回っている，すなわち，naiveの方が早く処理が終了していることになる．一方で，同じchar型でもn = 4096のときにはblockの高速化率は5.7907と非常に大きな値となっている．2次元配列の型ごとに比較してみると，最も大きな高速化率を記録した配列の大きさnはblockとblock_unrollingで共通していて，char型ではn = 4096で5.7907と5.5477，int型ではn = 512で5.1969と5.4642，long型ではn=128で3.9305と3.7608，double型ではn=128で3.7817と3.4748となった．すなわち，2次元配列の1つの要素が占めるメモリサイズが小さいほど，配列のサイズnが大きいときにより高速化していた．また，2次元配列の1つの要素が占めるメモリサイズが小さいほど，高速化率の最大値は大きくなった．次に，unrollingについて，高速化率が1を下回り，naiveよりも遅くなった2次元配列のサイズがchar型で2種類，int型で1種類，long型で3種類，double型で2種類あった．高速化率が1を上回ったとしても1.2以下のものがほとんど全てで，最大でも2を超えることはなかった．一方で，blockとblock_unrollingでは高速化率が極端に小さくなったchar型のn = 64，128のときには，高速化率がこれらよりも大きくなった．

2.7 配列のサイズ n が2のべき乗でない場合の最小CPEの変化測定

naiveとblockについて，int型2次元配列のサイズを2の冪乗周辺で様々に変えて，50回実行した時のCPEの最小値を測定した．その結果を表2.23に示す．

まず，naiveについて述べる．nが2のべき乗の場合の方が2のべき乗の前後1の場合に比べて極端に最小CPEが大きくなった．(2のべき乗)+1と(2のべき乗)-1を比較したときには，どちらかのCPEが必ず小さいということはなく，測定点によってまちまちであった．

次に，blockについて述べる．naiveの場合とは逆に，nが2のべき乗の場合の方が2のべき乗の前後1の場合に比べて最小CPEが小さくなった．しかしながら，それらの差は，naiveの時よりも小さかった．(2のべき乗)+1と(2のべき乗)-1を比較したときには，どちらかのCPEが必ず小さいということはなく，測定点によってまちまちであった．この点はnaiveと共通していた．

表2.1 naiveのCPEの度数分布表(int, n=2048)

CPE区分			度数
20	−	21	0
21	−	22	0
22	−	23	12
23	−	24	17
24	−	25	8
25	−	26	3
26	−	27	4
27	−	28	1
28	−	29	0
29	−	30	1
30	−	31	0
31	−	32	1
32	−	33	1
33	−	34	0
34	−	35	0
35	−	36	1
36	−	37	0
37	−	38	1
38	−	39	0
39	−	40	0

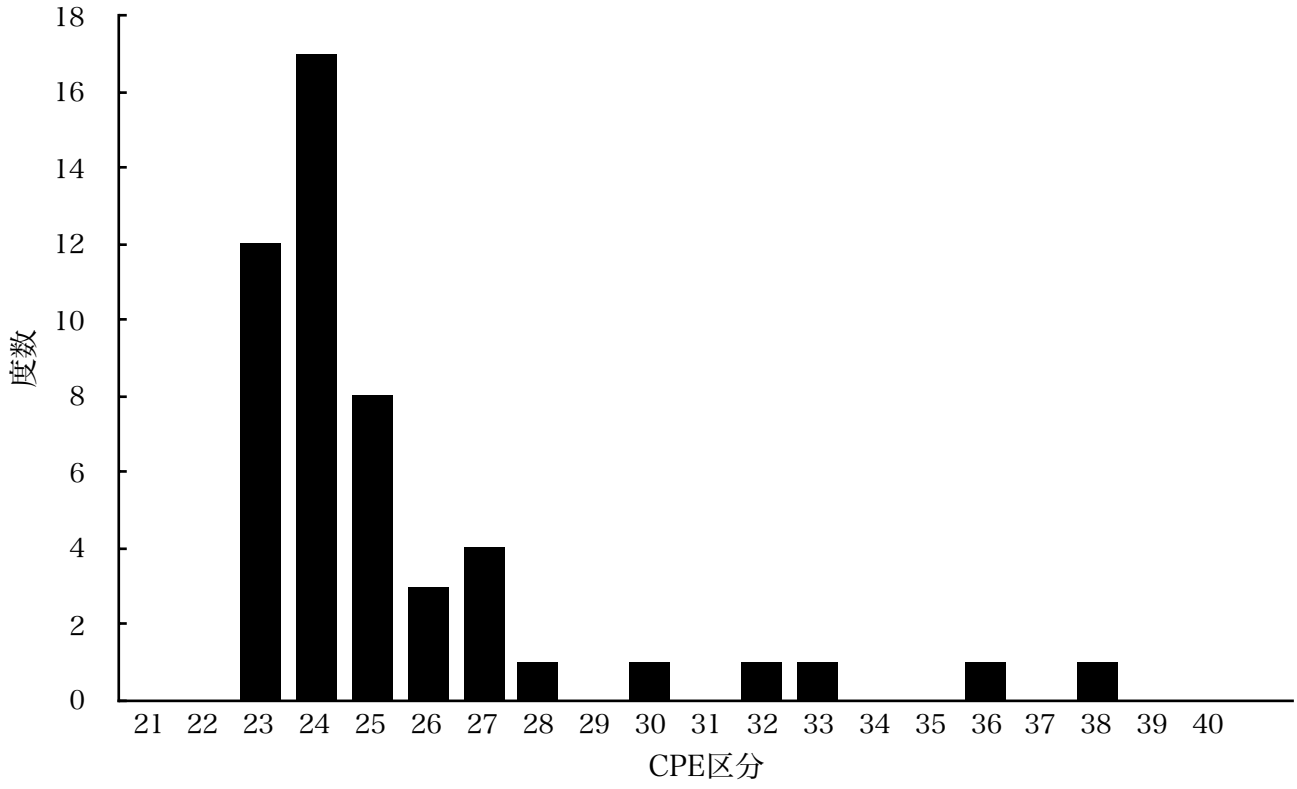

図2.1 naiveのCPEのヒストグラム(int, n=2048)

表2.2 最適化レベルと最小CPE(int, n=2048)

最適化レベル	レベル値	naive	unrolling	block	block_unrolling
O0	0	58.3671	23.7805	14.4342	8.9202
O1	1	21.8601	21.9760	4.8313	7.4055
O2	2	21.8171	22.0934	4.6422	4.8643
O3	3	21.8551	22.1626	4.6737	4.8161

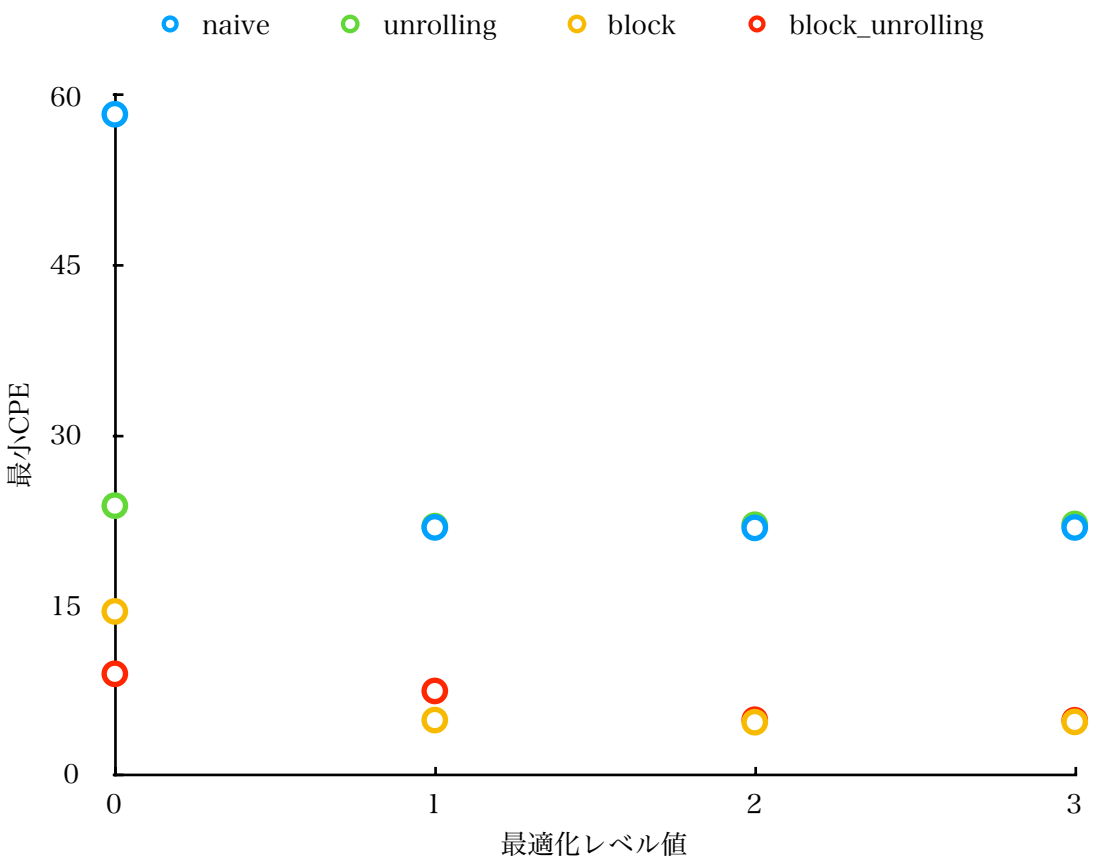

図2.2 最適化レベルと最小CPEの関係(int, n=2048)

表2.3 naiveの最小CPE(char)

配列のサイズn	最小CPE
64	2.3926
128	1.4688
256	6.4219
512	8.9573
1024	12.0215
2048	17.5077
4096	23.7681

表2.4 naiveの最小CPE(int)

配列のサイズn	最小CPE
64	1.7734
128	6.7070
256	10.3131
512	14.5686
1024	16.9230
2048	22.1672
4096	24.5397

表2.5 naiveの最小CPE(long)

配列のサイズn	最小CPE
64	3.6094
128	10.2771
256	14.2515
512	14.6393
1024	22.8936
2048	23.1915
4096	25.8773

表2.6 naiveの最小CPE(double)

配列のサイズn	最小CPE
64	3.5879
128	10.2454
256	14.2323
512	14.6947
1024	21.9068
2048	23.1950
4096	26.2706

表2.7 ブロックサイズを変えた時のblockの最小CPE(char)

配列のサイズn	2	4	8	16	32	64	最小
64	5.6475	2.9277	3.8965	2.2480	2.1152	2.0635	2.0635
128	4.3652	2.8364	2.3599	2.2310	2.1021	2.0618	2.0618
256	5.6646	2.7880	2.3314	2.2223	2.1110	2.1272	2.1110
512	8.3246	3.7437	2.3378	2.3717	2.2428	2.5566	2.2428
1024	12.2984	4.5947	2.3559	2.6750	2.7314	10.5465	2.3559
2048	16.1153	5.8324	3.0353	3.2668	10.6975	10.7957	3.0353
4096	17.3115	7.0905	4.1045	11.2885	10.4766	11.7984	4.1045

表2.8 ブロックサイズを変えた時のblockの最小CPE(int)

配列のサイズn	2	4	8	16	32	64	最小
64	4.2119	2.9717	2.0342	2.8066	1.6123	2.1992	1.6123
128	5.6208	3.8977	2.0239	2.0818	1.9397	3.7180	1.9397
256	8.5176	3.9315	2.1815	2.6411	3.5361	10.4262	2.1815
512	15.8054	5.0600	2.8033	3.5883	9.9126	9.7760	2.8033
1024	17.2016	6.5559	4.3241	12.5127	11.2828	12.0200	4.3241
2048	17.4593	7.5724	4.6557	14.7207	11.9754	12.3405	4.6557
4096	17.5262	7.7163	5.2445	15.1548	12.0216	12.3327	5.2445

表2.9 ブロックサイズを変えた時のblockの最小CPE(long)

配列のサイズn	2	4	8	16	32	64	最小
64	4.1084	3.8584	3.3086	2.4150	2.4990	4.0967	2.4150
128	6.1125	3.2478	2.6147	2.8015	4.1274	10.0164	2.6147
256	15.7695	5.2929	3.9171	4.6636	10.0692	10.0217	3.9171
512	16.6616	6.4095	6.9249	11.7024	10.9326	12.0502	6.4095
1024	17.5021	7.3692	7.6397	14.9115	12.9913	13.0492	7.3692
2048	17.6160	7.9661	8.0820	15.8722	13.5845	12.9703	7.9661
4096	18.0155	8.4569	8.6095	16.2550	13.4443	13.1131	8.4569

表2.10 ブロックサイズを変えた時のblockの最小CPE(double)

配列のサイズn	2	4	8	16	32	64	最小
64	4.3848	2.9268	2.5410	2.4551	3.8818	3.9277	2.4551
128	6.0693	3.2637	2.7092	2.8198	6.6472	10.3445	2.7092
256	15.7186	5.3269	4.4296	4.6377	10.0803	10.0237	4.4296
512	16.5763	6.4010	6.8520	11.7072	10.9532	12.0753	6.4010
1024	17.5301	7.3233	7.6023	14.6573	12.7637	13.6005	7.3233
2048	17.5705	7.9863	7.9532	15.8935	13.2926	13.0326	7.9532
4096	17.9120	8.3961	8.5763	16.6665	13.4439	13.0900	8.3961

表2.11 ブロックサイズを変えた時のblock_unrollingの最小CPE(char)

配列のサイズn	2	4	8	16	32	64	最小
64	1.8018	1.3975	1.6377	2.0586	2.4414	3.3643	1.3975
128	1.7129	1.4214	1.5779	2.1311	3.7583	3.7632	1.4214
256	5.7504	2.4695	1.6418	2.8223	2.9633	4.2256	1.6418
512	8.8672	3.5599	1.9213	3.5923	3.4251	5.9727	1.9213
1024	11.9934	4.6057	2.1953	4.1234	4.9115	11.2922	2.1953
2048	15.6108	5.8272	3.3875	5.7021	12.9496	11.8964	3.3875
4096	17.1417	7.3294	4.2843	13.6233	11.7736	12.9001	4.2843

表2.12 ブロックサイズを変えた時のblock_unrollingの最小CPE(int)

配列のサイズn	2	4	8	16	32	64	最小
64	1.0918	1.4033	1.6455	1.7871	2.2959	1.6211	1.0918
128	5.4260	2.5730	1.7925	2.2324	2.1194	4.6853	1.7925
256	8.3062	3.9761	2.1972	2.8262	4.9256	10.9373	2.1972
512	15.8689	5.1108	2.6662	4.6441	11.7542	10.8013	2.6662
1024	16.8061	7.1838	4.4814	14.1264	12.1964	12.9104	4.4814
2048	17.1348	7.5186	4.8120	16.0722	12.7026	13.0007	4.8120
4096	17.3472	7.8441	5.1717	16.7503	12.9846	13.0843	5.1717

表2.13 ブロックサイズを変えた時のblock_unrollingの最小CPE(long)

配列のサイズn	2	4	8	16	32	64	最小
64	2.9629	2.4395	2.6445	2.5557	2.4561	4.9424	2.4395
128	6.0620	3.0823	2.7327	2.8904	5.1113	10.6514	2.7327
256	15.6693	5.2603	3.9919	5.4200	11.4777	11.0710	3.9919
512	15.8304	6.3958	6.8428	13.3825	12.1981	13.0464	6.3958
1024	17.1035	7.3156	7.5400	16.0693	14.3779	13.8228	7.3156
2048	17.3221	8.0634	7.9556	17.1310	14.1871	13.7683	7.9556
4096	17.7327	8.7047	8.6670	17.8368	14.3525	13.8264	8.6670

表2.14 ブロックサイズを変えた時のblock_unrollingの最小CPE(double)

配列のサイズn	2	4	8	16	32	64	最小
64	3.7305	2.3936	2.6201	2.5176	2.6914	5.0938	2.3936
128	5.9160	3.1094	3.6101	2.9485	5.3679	10.6643	2.9485
256	15.5508	5.3021	4.4429	5.1630	11.7814	10.9102	4.4429
512	15.5868	6.3355	6.8309	13.2662	12.2228	12.9189	6.3355
1024	17.1898	7.3454	7.6900	15.8517	14.0996	13.5844	7.3454
2048	17.3163	8.1253	7.9966	17.0831	14.4168	13.6425	7.9966
4096	17.7853	8.4956	8.5684	17.6860	14.4712	13.7274	8.4956

表2.15 ループアンローリングの回数を変えた時のunrollingの最小CPE(char)

配列のサイズn	2	4	8	16	32	64	最小
64	1.6650	1.2266	1.2227	1.4658	2.5488	2.4727	1.2227
128	1.3542	1.2173	1.6648	1.3708	2.2095	2.2034	1.2173
256	6.5562	6.4525	6.4763	6.4741	6.4922	6.6393	6.4525
512	8.5198	7.7914	8.9641	8.4450	8.4871	9.2770	7.7914
1024	12.0284	11.9989	12.0154	12.0461	12.2518	12.1546	11.9989
2048	17.5458	21.7766	17.5128	17.5687	23.3305	17.4871	17.4871
4096	24.4710	24.7907	23.8806	23.9113	24.5008	24.4907	23.8806

表2.16 ループアンローリングの回数を変えた時のunrollingの最小CPE(int)

配列のサイズn	2	4	8	16	32	64	最小
64	1.2930	1.8984	1.2197	1.4922	2.9238	2.5693	1.2197
128	6.7151	6.7512	6.7161	6.8044	6.9421	7.3840	6.7151
256	10.2440	10.4000	10.4366	10.2939	10.7375	10.6855	10.2440
512	14.6550	14.5015	14.4803	14.6126	14.5568	14.5779	14.4803
1024	16.3444	20.3324	22.7854	16.5699	16.5408	16.5709	16.3444
2048	21.9441	21.8955	25.1268	23.3443	22.1112	22.4178	21.8955
4096	24.4976	24.5615	24.5452	24.8215	25.2620	25.8913	24.4976

表2.17 ループアンローリングの回数を変えた時のunrollingの最小CPE(long)

配列のサイズn	2	4	8	16	32	64	最小
64	3.7012	3.7207	3.8877	4.4629	5.2480	6.4053	3.7012
128	10.4424	10.2712	10.5195	10.1707	10.3347	10.7319	10.1707
256	14.3876	14.2722	14.3635	14.2620	14.4009	14.3566	14.2620
512	14.7260	14.6779	14.7135	14.8362	14.7756	14.8528	14.6779
1024	23.2562	22.5514	23.0669	22.4407	22.5702	22.5267	22.4407
2048	24.7931	23.5058	23.1144	23.3400	23.5123	23.6743	23.1144
4096	25.6914	26.0062	25.6764	26.0124	26.7169	26.8477	25.6764

表2.18 ループアンローリングの回数を変えた時のunrollingの最小CPE(double)

配列のサイズn	2	4	8	16	32	64	最小
64	3.9277	4.3193	4.1895	3.9844	5.0010	6.3691	3.9277
128	10.2212	10.2537	10.2812	10.2046	10.2485	10.7166	10.2046
256	14.2842	14.2968	14.2619	14.2382	14.3609	14.5414	14.2382
512	14.7246	14.6525	14.6807	14.6475	14.8440	15.0140	14.6475
1024	21.7381	21.7548	21.7313	23.8990	21.9104	22.9513	21.7313
2048	23.0863	23.5142	23.3477	23.4106	24.3442	23.6244	23.0863
4096	25.9510	25.7145	26.3640	25.9495	27.1473	27.2633	25.7145

表2.19 配列のサイズを変えた時の最小の最小CPEと高速化率(char)

配列のサイズn	log n	naive		block		block_unrolling		unrolling	
		CPE	高速化率※	CPE	高速化率	CPE	高速化率	CPE	高速化率
64	6	2.3926	1.0000	2.0635	1.1595	1.3975	1.7121	1.2227	1.9568
128	7	1.4688	1.0000	2.0618	0.7124	1.4214	1.0333	1.2173	1.2066
256	8	6.4219	1.0000	2.1110	3.0421	1.6418	3.9115	6.4525	0.9953
512	9	8.9573	1.0000	2.2428	3.9938	1.9213	4.6621	7.7914	1.1496
1024	10	12.0215	1.0000	2.3559	5.1027	2.1953	5.4760	11.9989	1.0019
2048	11	17.5077	1.0000	3.0353	5.7680	3.3875	5.1683	17.4871	1.0012
4096	12	23.7681	1.0000	4.1045	5.7907	4.2843	5.5477	23.8806	0.9953

表2.20 配列のサイズを変えた時の最小の最小CPEと高速化率(int)

配列のサイズn	log n	naive		block		block_unrolling		unrolling	
		CPE	高速化率※	CPE	高速化率	CPE	高速化率	CPE	高速化率
64	6	1.7734	1.0000	1.6123	1.0999	1.0918	1.6243	1.2197	1.4540
128	7	6.7070	1.0000	1.9397	3.4578	1.7925	3.7417	6.7151	0.9988
256	8	10.3131	1.0000	2.1815	4.7275	2.1972	4.6937	10.2440	1.0067
512	9	14.5686	1.0000	2.8033	5.1969	2.6662	5.4642	14.4803	1.0061
1024	10	16.9230	1.0000	4.3241	3.9136	4.4814	3.7763	16.3444	1.0354
2048	11	22.1672	1.0000	4.6557	4.7613	4.8120	4.6067	21.8955	1.0124
4096	12	24.5397	1.0000	5.2445	4.6791	5.1717	4.7450	24.4976	1.0017

表2.21 配列のサイズを変えた時の最小の最小CPE(long)

配列のサイズn	log n	naive		block		block_unrolling		unrolling	
		CPE	高速化率※	CPE	高速化率	CPE	高速化率	CPE	高速化率
64	6	3.6094	1.0000	2.4150	1.4946	2.4395	1.4796	3.7012	0.9752
128	7	10.2771	1.0000	2.6147	3.9305	2.7327	3.7608	10.1707	1.0105
256	8	14.2515	1.0000	3.9171	3.6383	3.9919	3.5701	14.2620	0.9993
512	9	14.6393	1.0000	6.4095	2.2840	6.3958	2.2889	14.6779	0.9974
1024	10	22.8936	1.0000	7.3692	3.1067	7.3156	3.1294	22.4407	1.0202
2048	11	23.1915	1.0000	7.9661	2.9113	7.9556	2.9151	23.1144	1.0033
4096	12	25.8773	1.0000	8.4569	3.0599	8.6670	2.9857	25.6764	1.0078

※ 高速化率とは, naiveの最小CPEをそれぞれの最小の最小CPEで割った値である.

表2.22 配列のサイズを変えた時の最小の最小CPE(double)

配列のサイズn	log n	naive		block		block_unrolling		unrolling	
		CPE	高速化率※	CPE	高速化率	CPE	高速化率	CPE	高速化率
64	6	3.5879	1.0000	2.4551	1.4614	2.3936	1.4990	3.9277	0.9135
128	7	10.2454	1.0000	2.7092	3.7817	2.9485	3.4748	10.2046	1.0040
256	8	14.2323	1.0000	4.4296	3.2130	4.4429	3.2034	14.2382	0.9996
512	9	14.6947	1.0000	6.4010	2.2957	6.3355	2.3194	14.6475	1.0032
1024	10	21.9068	1.0000	7.3233	2.9914	7.3454	2.9824	21.7313	1.0081
2048	11	23.1950	1.0000	7.9532	2.9164	7.9966	2.9006	23.0863	1.0047
4096	12	26.2706	1.0000	8.3961	3.1289	8.4956	3.0923	25.7145	1.0216

※ 高速化率とは, naiveの最小CPEをそれぞれの最小の最小CPEで割った値である.

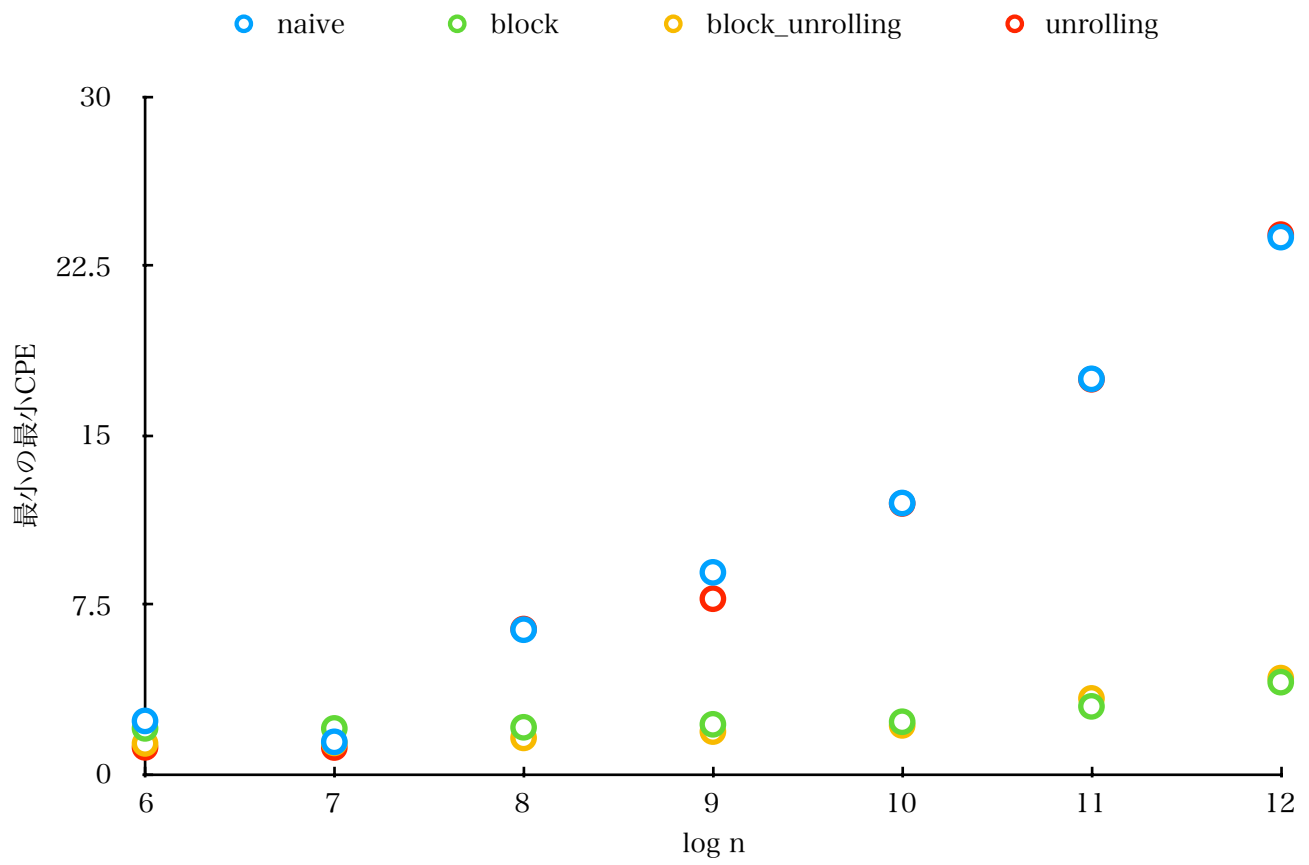

図2.3 配列のサイズnと各左回転関数の最小の最小CPEの片対数グラフ(char型)

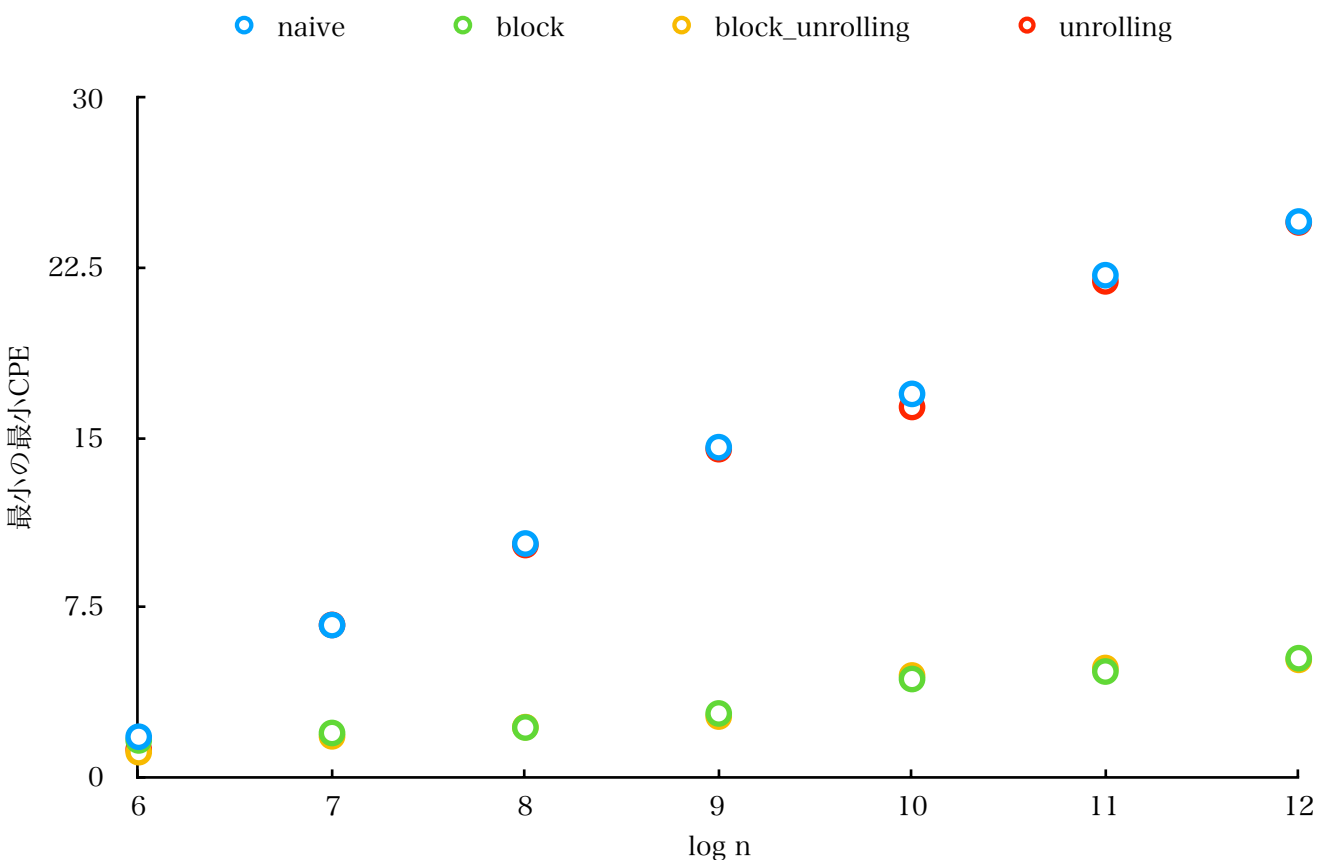

図2.4 配列のサイズnと各左回転関数の最小の最小CPEの片対数グラフ(int型)

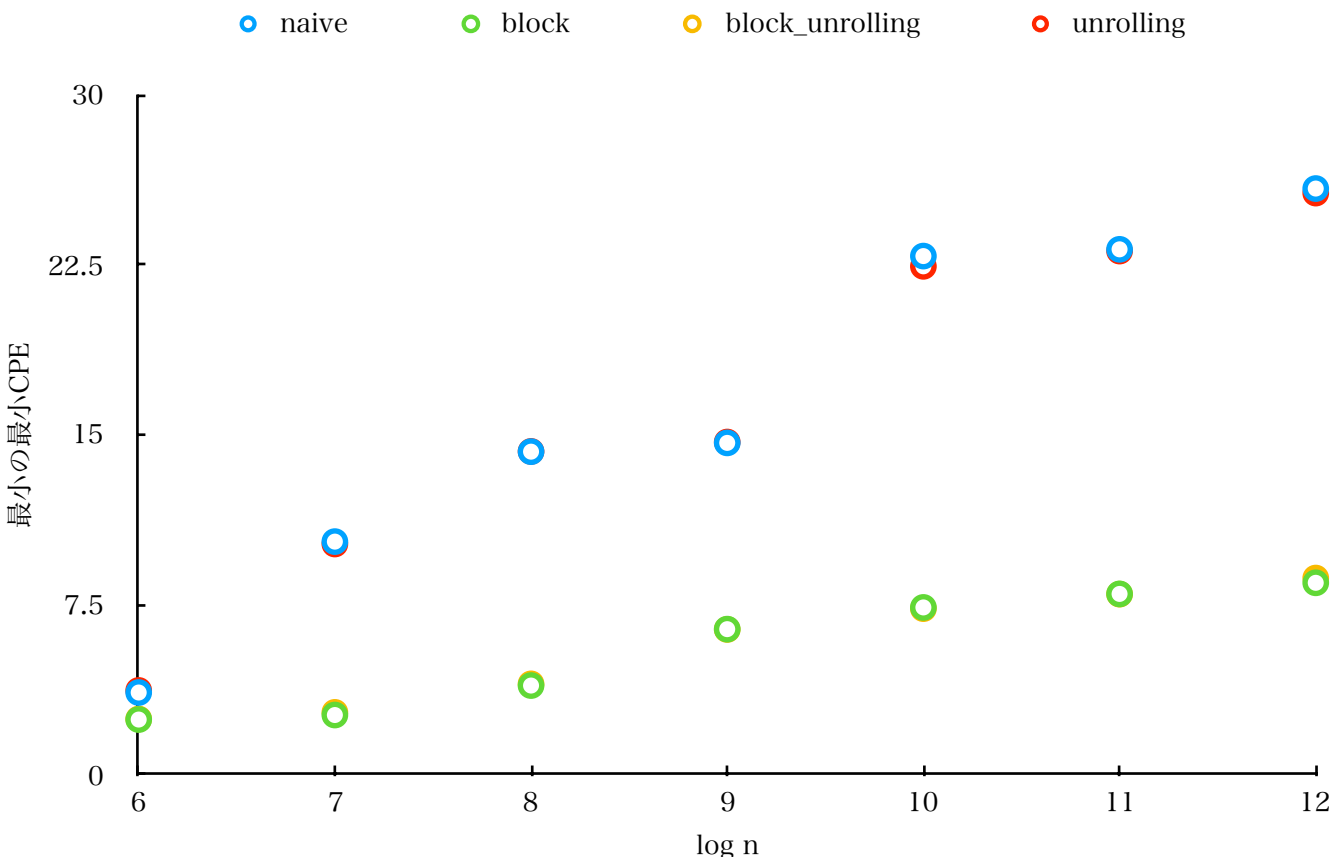

図2.5 配列のサイズnと各左回転関数の最小の最小CPEの片対数グラフ(long型)

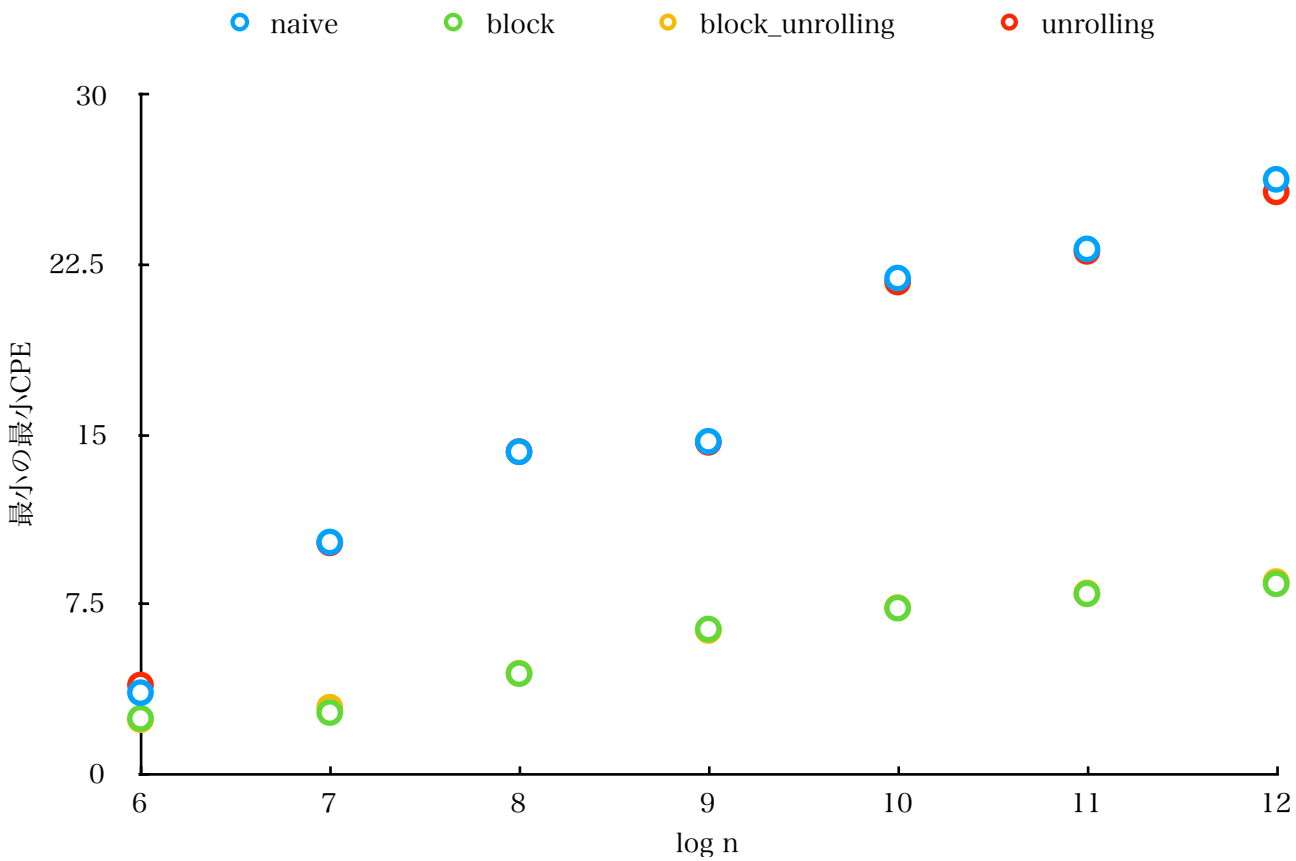

図2.6 配列のサイズnと各左回転関数の最小の最小CPEの片対数グラフ(double型)

表2.23 naiveとblockの最小CPE(int)

配列のサイズ n	naive	block
63	1.7697	1.6085
64	1.7734	2.1797
65	1.7581	2.2561
127	2.1638	2.2273
128	6.6523	1.9395
129	2.1927	2.2044
255	2.5182	3.2917
256	10.0797	2.2102
257	2.8484	3.0846
511	10.2971	3.8122
512	14.4478	2.6845
513	6.4394	3.6237
1023	9.9166	5.9444
1024	17.9853	4.3535
1025	9.9899	5.6872
2047	9.1923	7.1763
2048	21.7412	4.6642
2049	9.1537	6.3729
4095	15.7475	7.1661
4096	24.4657	4.9847
4097	15.8295	6.6981

3. 考察

3.1 CPE分布の測定

CPE分布のヒストグラムは対称形にはならず，最小値付近に度数が集中していて一部極端に大きな値が外れ値として存在しているような形状となった．この理由は，1回の左回転関数の実行中に，どの程度マシンの中で動いている他のプロセスへのコンテキストスイッチによる影響が入ったかによって実行が遅くなっている回があるためだと考える．1回の左回転回数の実行中に何度か他のプロセスへコンテキストスイッチを起こすことはあっても，私が実行しているプログラムから生成されたプロセスになかなか実行権が回ってこず，極端に実行時間が遅くなることは稀なのだと考えることができる．

50回の測定におけるCPEの最小値と平均値がそれぞれ22.0880, 24.8202であり，CPEの分布の中で度数が最大だったのは23より大きく24以下の区分で，その次ががCPEが22より大きく23以下の区分あった．このことから，各プログラムの性能として平均値を用いるのは，頻度が十分に高くな区分の値を採用していることになり適切ではないと言える．一方，最小値は，以下の2点の理由により，プログラムの性能を比較する基準として平均値よりも適切であると考える：

- 最頻区分には含まれないものの2番目に頻度の高い区分の値であること
- プログラムの実行時間が長くなる場合があるのは，そのプログラムそのものの影響ではなく，他のプロセスの影響であると考えられること

全てのCPEの測定結果の報告において最小値を用いているのはこのためである．

3.2 コンパイラ最適化レベルによる最小CPEの変化測定

まず，全く最適化をしないO0の時にはバラバラに分かれていた各左回転プログラムの最小CPEが，O1以上の最適化レベルではnaiveとunrolling，そしてblockとblock_unrollingの2つの組みに最小CPEが分かれた．より具体的には，naiveの最小CPEがそれにループアンローリングを施したunrollingの最小CPEに，blockの最小CPEがそれにループアンローリングを施したblock_unrollingの最小CPEに近く形で2つに分かれた．これは，コンパイラが最適化処理の中でループアンローリングと同等の効果を得られる工夫を行なっているため，及びブロック化を施しているかどうかが最小CPEに大きく関係しているためだと考える．

次に，block_unrolling以外の3つの左回転回数において，O2の最適化レベルの方がO3の最適化レベルの時よりも最小CPEが大きくなったのは，測定時に生じる偶然の差のためだと考える．実際，block_unrolling以外の3つの左回転回数でのO2とO3のときのCPEの差は最大でも0.0692である．実験1.1によって，CPEの測定値が0.1程度変化することは簡単に起こることがわかるから，これは偶然に生じた誤差だといって良い．したがって，最適化レベルをO2からO3に上げても全く効果はないと考え，実験1.3以降では最適化レベルをO2としている．

3.3 2次元配列のサイズ, 型による愚直なプログラムの最小CPEの変化測定

2次元配列のサイズが等しければ，1つの要素が占めるメモリサイズが小さいほど最小CPEは小さくなった理由を考える．1つの要素が占めるメモリサイズが小さければ，キャッシュにのるデータの数が多くなるため，キャッシュミス率が低くなって高速化がなされるためであると考える．このように考えれば，同時に，1つの要素が占めるメモリサイズが等しいlongとdoubleでは最小CPEに優劣がつけられないことも説明ができる．

次に，配列の型によらず，2次元配列のサイズが大きくなるほど，CPEは大きくなった理由を考える．nが大きくなるほど広い範囲のメモリアドレスにアクセスする必要があり，キャッシュミスが発

生する回数が増えるためであると考える．また，飛び飛びでアクセスをする方の配列については，そのアクセスの飛び幅が広くなるため，よりキャッシュミスの頻度も多くなることも，最小CPEの増加に大きく関係していると考える．

char型で2次元配列のサイズが小さいところでの例外的な挙動については，グラフと合わせて3.6節で述べることにする．

3.4 2次元配列のサイズ，型とブロックサイズによる最小CPEの変化測定

まず，blockについて考える．

配列のサイズが等しければ，1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった理由はnaiveと同様だと考える．1つの要素が占めるメモリサイズが等しいlongとdoubleでは，longのほうが最小CPEが小さい場合が多かったことから，ブロック化においては整数型の方が，浮動小数点数型よりもデータの高速に動作すると言える．

2次元配列のサイズが大きくなるほど，最小の最小CPEを与えるブロックサイズは小さくなった理由を考える．2次元配列のサイズnが大きくなると，飛び飛びでアクセスされる配列のアクセスの飛び幅が広くなるため，キャッシュミスがより起こりやすくなる．ブロック化は，配列のアクセス範囲を制限することによってキャッシュミスの頻度を下げるための工夫であったから，nが大きくなれば，配列のアクセス範囲の制限はブロックサイズが小さい方がより適切となるためである．ブロックサイズが小さくなりすぎるとブロック数が増えるため，各ブロックを順に処理するループの回数が増え，オーバヘッドが大きくなる．この理由から，必ずしもブロックサイズが小さいほど最小CPEが小さくならなかったのだと考える．この説明が正しいとすれば，最小の最小CPEを与えるブロックサイズを下回るとブロックサイズが小さいほど最小CPEが大きくなり，結果として最小の最小CPEに近いほど最小CPEの値が小さくなることとも矛盾しない．

次に，block_unrollingについて考える．

配列のサイズが等しければ，1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった理由はnaiveと同様だと考える．1つの要素が占めるメモリサイズが等しいlongとdoubleでは，longのほうが最小CPEが小さい場合が多かったことはblockと共通しており，やはりブロック化においては整数型の方が，浮動小数点数型よりもデータの高速に動作すると言える．

最小の最小CPEに近いほど最小CPEの値が小さくなる理由は，blockと同様であると考える．2次元配列のサイズと最小の最小CPEを与えるブロックサイズに関係性が見られないのは，ループアンローリングの影響によると考える，ループアンローリングの影響の詳細は次節で述べる．

char型で2次元配列のサイズが小さいところでの例外的な挙動については，グラフと合わせて3.6節で述べることにする．

3.5 2次元配列の型とアンローリング回数による最小CPEの変化測定

配列のサイズが等しければ，1つの要素が占めるメモリサイズが小さいほど最小の最小CPEは小さくなった理由および1つの要素が占めるメモリサイズが等しいlongとdoubleでは最小CPEに優劣がつけられない理由はnaiveと同様だと考える．

ループアンローリングの回数と最小CPEの間に関係性が見られなかった理由を考える．ループアンローリングは，ループの中身を展開することで，オーバヘッドを小さくするとともに並列実行を可能にする工夫である．したがってメモリアクセスの順序に対する工夫ではないため，配列の要素をコピーするプログラムのCPEを大きく支配するとは考えにくい．したがって，関係性が見られなかったのだと考える．

2次元配列のサイズが大きくなるほど，ループアンローリングの回数を変更したときのCPEの変化は小さくなる傾向にあった理由を考える．2次元配列のサイズが大きくなれば，ループの実行予測が当たる回数が大きくなる．ループアンローリングは命令が分岐なく必ず順に進むことをプログラマが保証しているために高速化されるとも考えることができる． 2次元配列のサイズが大きくなれば，ループアンローリングでプログラマが分岐なしと保証しても，処理ユニットがループが分岐なく実行されるはずだという予測を行っても変化がなくなるからだと考える．

3.6 4つの左回転関数の最小の最小CPEの比較

図2.3から図2.6に注目する． 型によらず， 全体としてnaiveとunrolling， そしてblockとblock_unrollingの2つの組に最小の最小CPEが分かれた理由は， 3.2節で述べた理由と同様であると考える．このことが，複数の要素型および配列のサイズで確認できた．

naiveとunrollingについて考える． 2次元配列のサイズと最小の最小CPEの関係を示した対数グラフ上で凸性が途中で変化したように見えたのは， 実は2次元配列のサイズが小さいところで例外的に最小の最小CPEが大きくなっていたからであると考える．2次元配列のサイズnが小さいところで例外的に最小の最小CPEが大きくなっているという現象は， 3.4節や3.5節を含めて， char型のn = 64, 128等ではっきりと見られている．これは，nが小さい場合には，実行ユニットが速度の限界を迎えたために一定以上には速くならなかったためだと考える．

次に，blockとblock_unrollingについて考える．char型，n = 64で最小の最小CPEが直線的な変化にしたがっておらず大きな値となったのは，先と同様，実行ユニットの限界によると考える．longとdoubleにおいて，log n = 9とlog n = 10の間で 最小の最小CPEに跳躍があったのは，キャッシュに2つの2次元配列がどの程度のるかの差であると考える．log n = 9のとき，2つの2次元配列が必要とするメモリサイズは，$2 \times \mathrm{sizeof(double)} \times 2^9 \times 2^9 = 4 \times 2^{20} = 4$ [MB]であるが，私のマシンの1次キャッシュと2次キャッシュを合わせたサイズは$3 + 0.25 \times 2 = 3.50$[MB]である．ここで，キャッシュサイズの計算の中で2をかけたのは，私のマシンのコア数が2だからである．したがって，log n = 9のときにはほとんどのデータが2次キャッシュまでに乗るために， それより大きいときと比較して高速であると考える．

次に，高速化率を考える．blockとblock_unrollingについて，2次元配列のサイズが小さいときには高速化率が小さくなったが，サイズが大きくなれば高速化率が非常に大きくなった．その一方で，unrollingについて高速化率が1を下回り，naiveよりも遅くなった2次元配列のサイズの種類が少なくなかったが，char型のn = 64, 128のときには，高速化率が大きくなった．このことより，扱うデータの量が小さいときにはループアンローリングが高速化に効果があり，扱うデータの量が大きいときにはブロック化によってメモリアクセスの順序を工夫する方が効果があると言える．

さて，最終的に4つの左回転関数のうちどれを用いるべきかをまとめておく．扱うデータの量が32[KB]程度までと小さいときにはunrolling， それ以上の場合にはブロック化を用いると効果的である．blockとblock_unrollingの間には，最小の最小CPEの明確な差は見られないので，ループアンローリングを持たずに実装の煩雑さが少ないblockを用いるのが最も良い．

3.7 配列のサイズ n が2のべき乗でない場合の最小CPEの変化測定

まず，naiveについて考える．nが2のべき乗の場合の方が2のべき乗の前後1の場合に比べて極端に最小CPEが大きくなった理由を述べる．これは，nが2のべき乗のときには，キャッシュの特定のラインのみが使われてしまうことで起こるコンフリクトミスが発生するため，広いキャッシュのメモリ領域を十分に生かせていないからだと考える．

次にblockについて考える．naiveの場合とは逆に，nが2のべき乗の場合の方が2のべき乗の前後1の場合に比べて最小CPEが小さくなった理由を述べる．これはブロック化において，nが2のべき乗でないと，最後に残った部分を別で処理しなければならないために遅くなったのだと考える．

さらに，余りの処理によるCPEの増加よりも，コンフリクトミスによるCPEの増加の方が影響が大きいということがわかる．

4. 感想

最初のプログラムは非常に単純であるにもかかわらず，それぞれの工夫によって特徴的なCPEの測定結果が得られたのは興味深かった．また，データのコピーにおいてはループアンローリングよりもブロック化によるアクセス順序の工夫の方が重要そうなことは，感覚的にわかるが，それが測定データとしてここまではっきり現れるとは思っておらず，楽しい実験となった．