全体ゼミ

大石純平（筑波大学）
プログラム論理研究室

2015/4/24

Offshoreとは

委託

Offshoreで目指すこと

- 人間が書きやすい高級な言語から，低級な言語へ変換を行う.

Offshoreで目指すこと

- 人間が書きやすい高級な言語から，低級な言語へ変換を行う.
- 変換後のコードが実行効率を犠牲にすることなく，低級な言語が得意な最適化を行うことができれば嬉しい.

Offshoreで目指すこと

- 人間が書きやすい高級な言語から，低級な言語へ変換を行う.
- 変換後のコードが実行効率を犠牲にすることなく，低級な言語が得意な最適化を行うことができれば嬉しい.
- 一つのコード生成器から，目的に応じた様々なコードを生成する.

使用した言語

BER MetaOCaml

マルチステージプログラミング言語

使用した言語

BER MetaOCaml

マルチステージプログラミング言語

- コードを生成する前に，生成したコードの安全性を保証.

使用した言語

BER MetaOCaml

マルチステージプログラミング言語

- コードを生成する前に，生成したコードの安全性を保証.
  - 生成したコードは実行時エラーが起きない.

今回用いたシステム

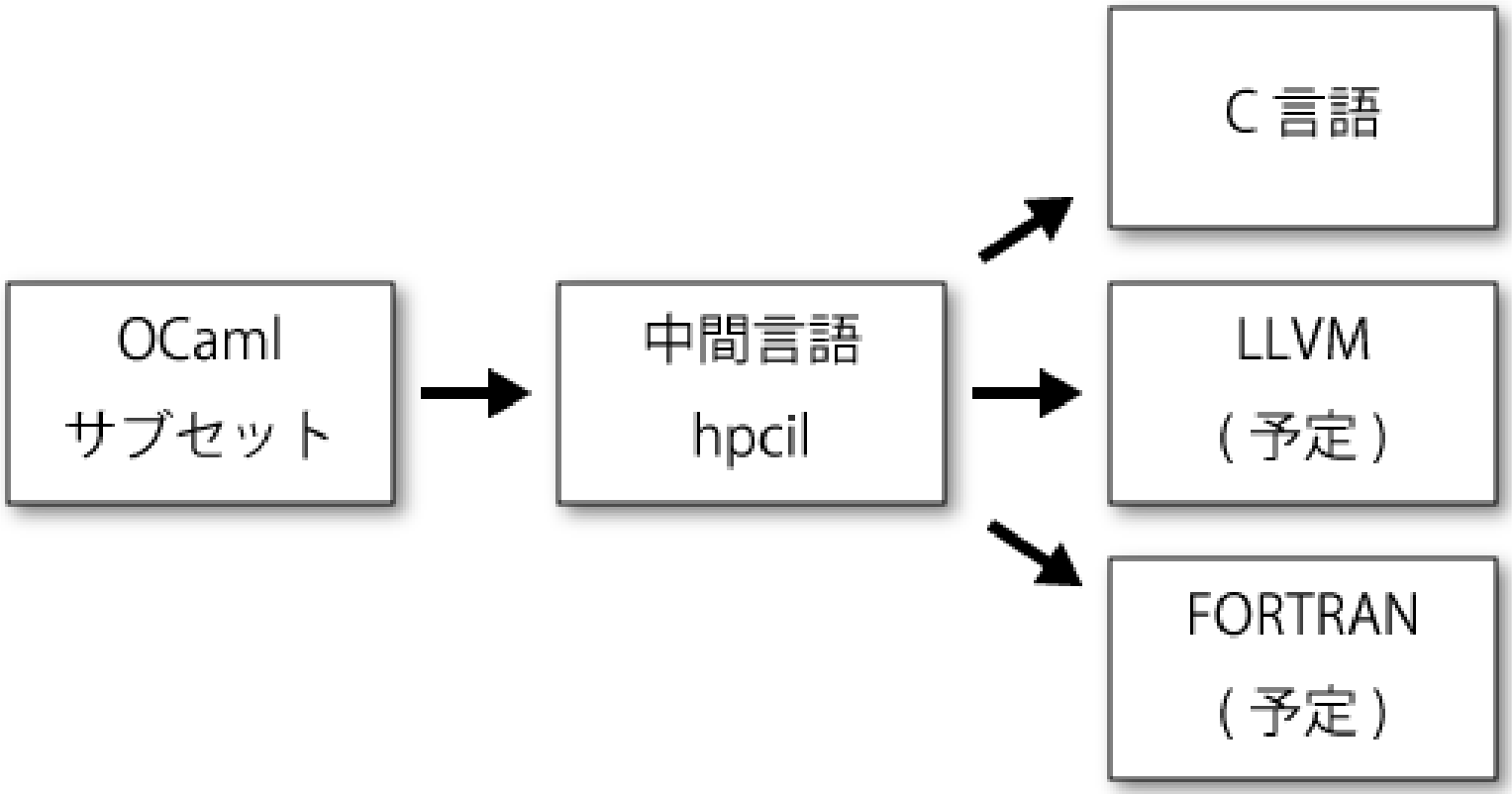
明日奈システム

今回用いたシステム

明日奈システム

高島さんの作ったやつ

今回用いたシステム



実験の趣旨

最適化を行わない場合の OCaml のコードと変換後の C 言語のコードとの実行時間を調べたい.

実験内容

gcc -O3, gcc, clang -O3, clang, ocamlc, ocamlopt によってコンパイルしたものをそれぞれ 5 回実行し，その結果の平均を求めた.

実験

C

- clang -O3
- gcc -O3

OCaml

- ocamlopt
- ocamlc

スペック

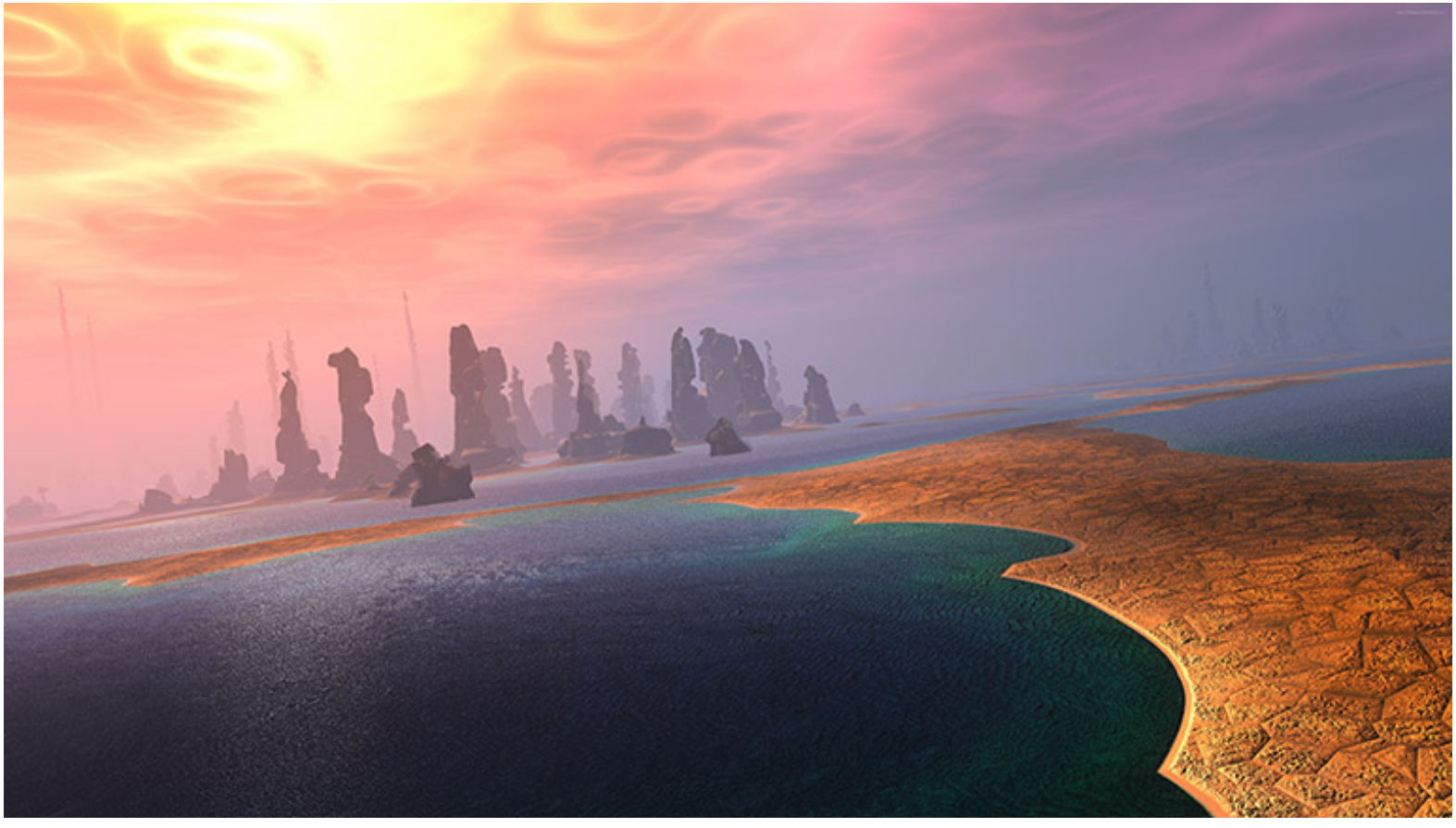
使用した画像

- 8k 画像 (768×4320 pixel)

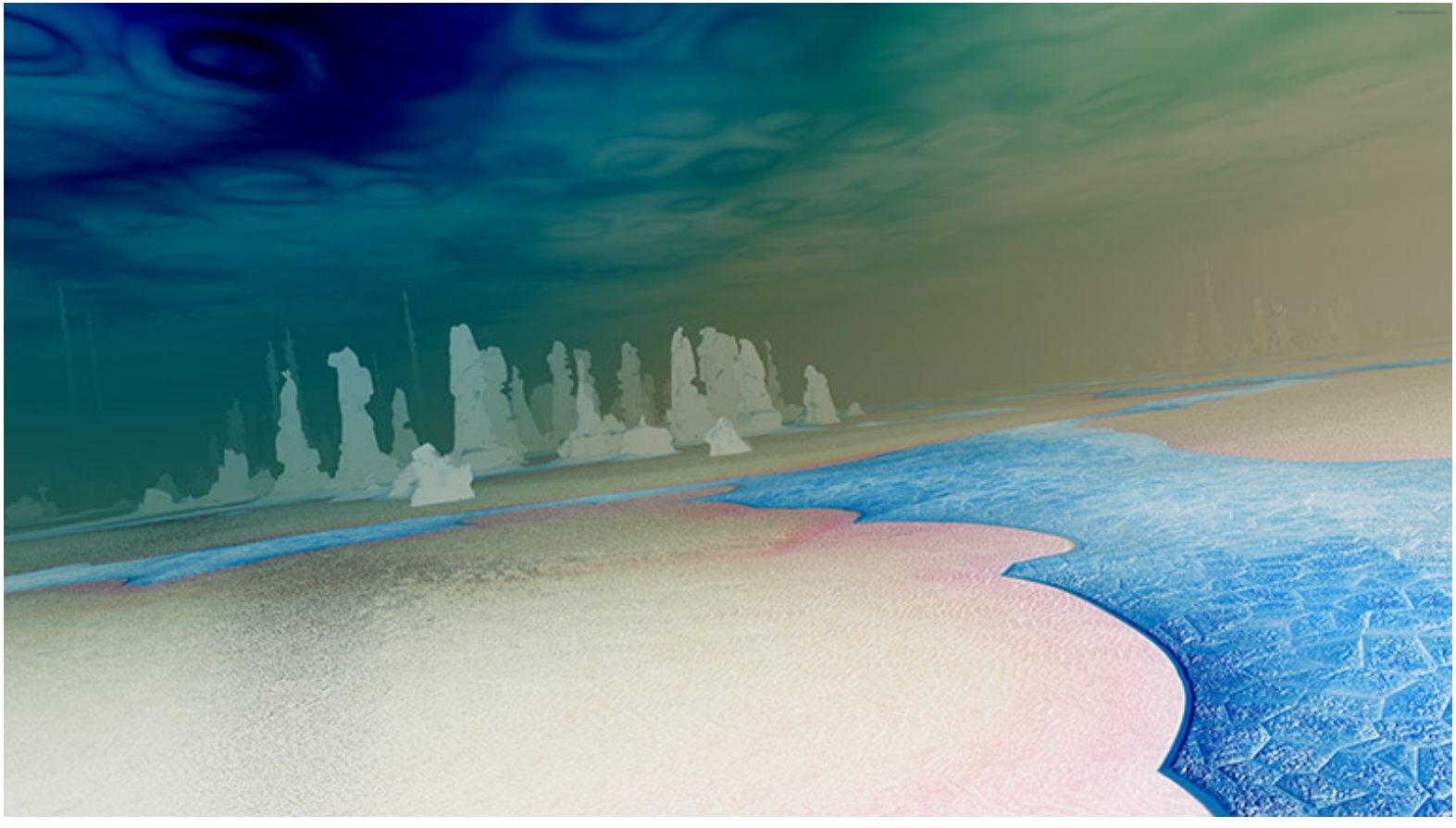
マシンスペック

- CPU: Intel Core i7-4790K
- コア数: 4
- スレッド数: 8
- ベース動作周波数: 4.0GHz
- L1 キャッシュサイズ: 256KB
- L2 キャッシュサイズ: 1024KB
- L3 キャッシュサイズ: 8192KB
- メモリ: DDR3 16GB 1600MHz

使用した画像



ネガポジ反転した画像



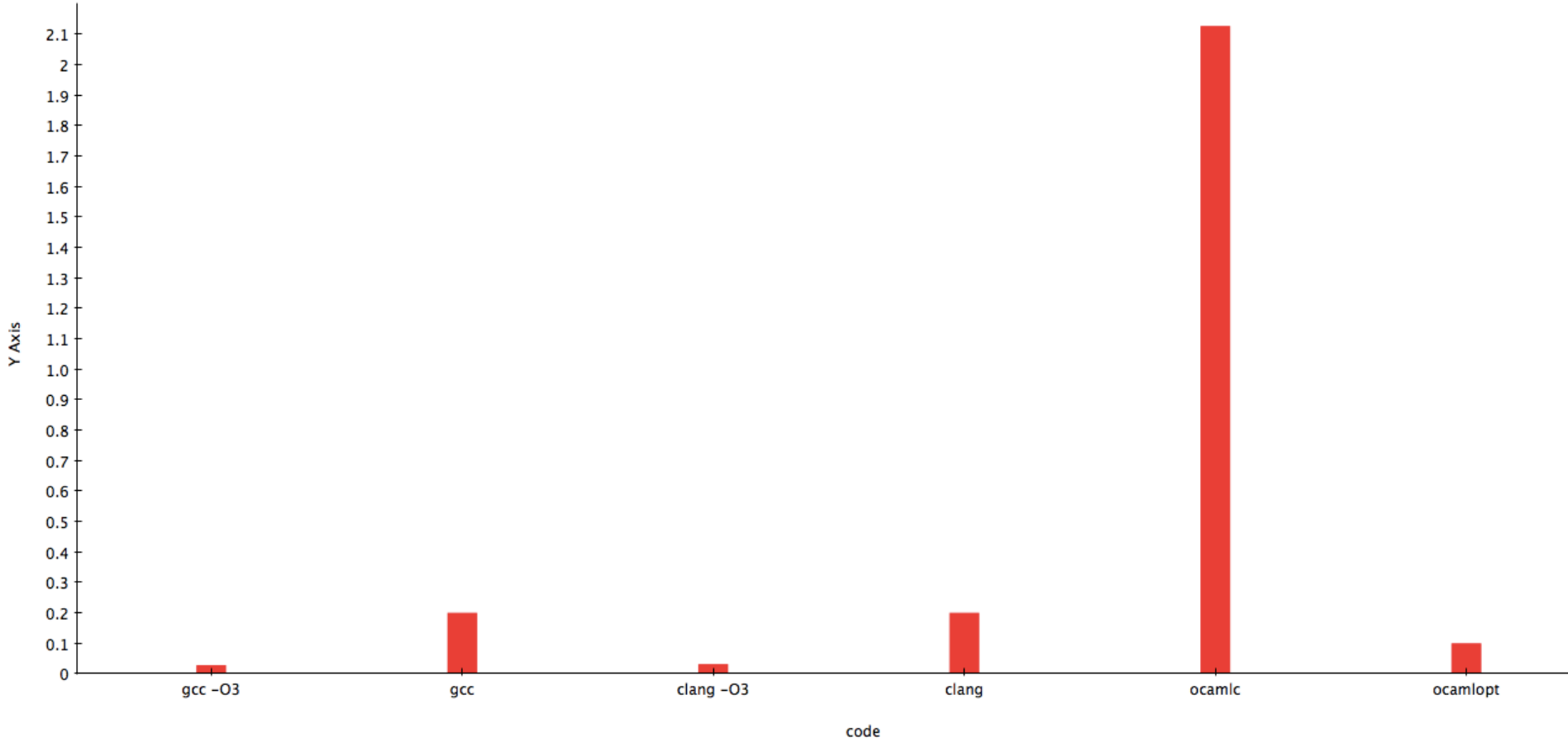
使用したコード

```
.<fun w h inp out ->
  for i = 0 to w*h*3 - 1 do
    out.(i) <- 255 - inp.(i)
  done>.
```

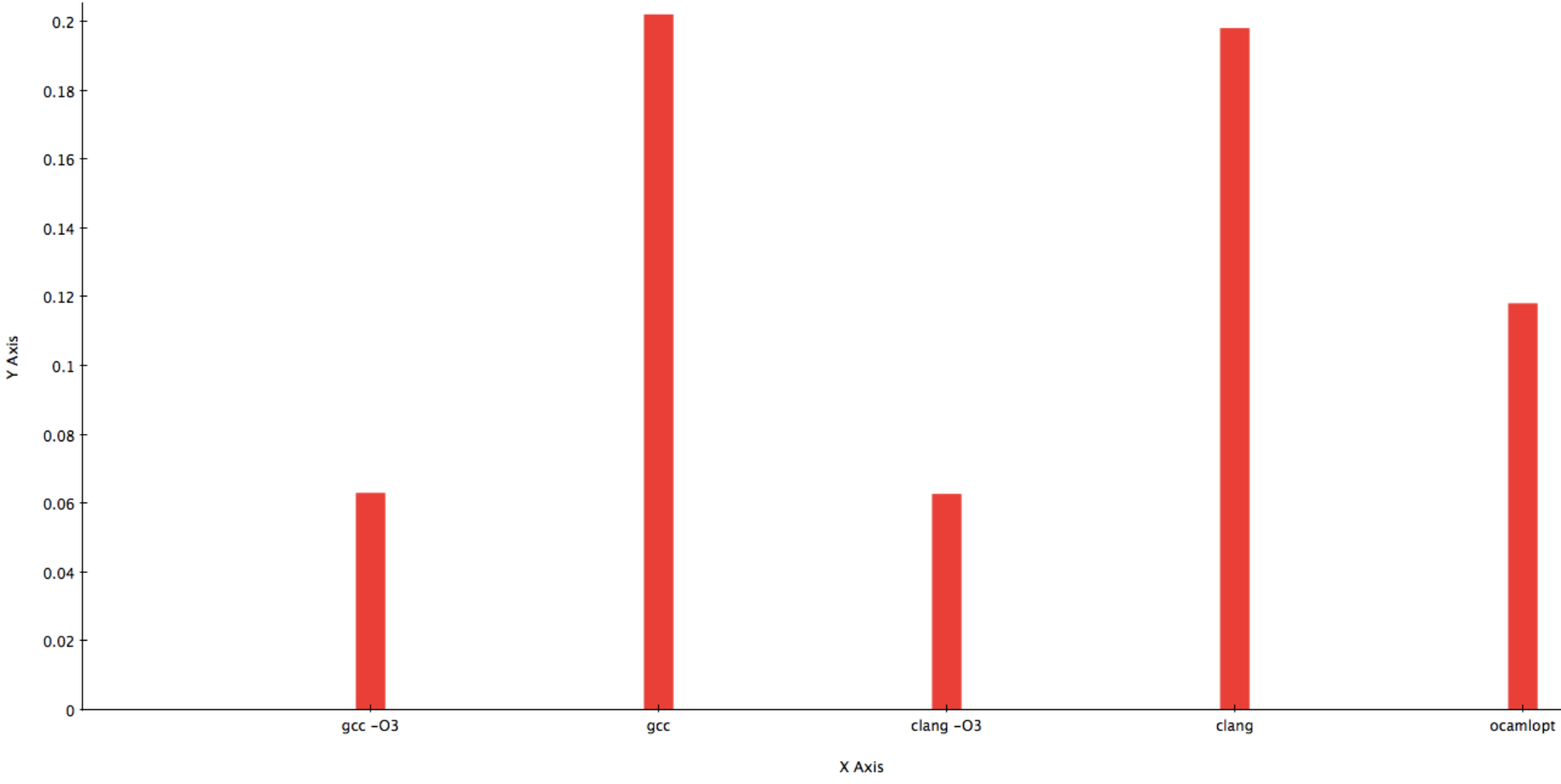
変換したコード

```
void ngps(int w_1, int h_2, int *inp_3, int *out_4)
{
    int i_5;
    for (i_5 = 0; i_5 <= w_1 * h_2 * 3 - 1; i_5++)
        out_4[i_5] = 255 - inp_3[i_5];
}
```

実験結果


Y Axis
2.1
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
gcc -O3
gcc
clang -O3
clang
ocamlc
ocamlopt
code

実験結果


0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
Y Axis
gcc -O3
gcc
clang -O3
clang
ocamlopt
X Axis

実験結果

- gcc -O3 : 0.0629 [s]
- gcc : 0.202 [s]
- clang -O3 : 0.0626 [s]
- clang : 0.198 [s]
- ocamlc : 2.161 [s]
- ocamlopt : 0.118 [s]

考察

- ocamlc でのコンパイルによって生成された実行ファイルの実行時間は一番遅い.
- ocamlopt は健闘しているが，offshore された C 言語の-O3 オプションを付けて生成した実行ファイルの実行時間は，それに比べて 2 倍ほど速い.

実験結果

回数	gcc -O3 [s]	gcc [s]	clang -O3 [s]	clang [s]	ocamlc [s]	ocamlopt [s]
1	0.084628	0.248957	0.083537	0.245865	2.161121	2.161121
2	0.057468	0.190805	0.057313	0.186081	2.160990	2.160990
3	0.057403	0.190846	0.057324	0.185934	2.159268	2.159268
4	0.057429	0.190796	0.057357	0.185931	2.160954	2.160954
5	0.057414	0.190914	0.057307	0.185922	2.159235	2.159235

考察

- C のコンパイラによって生成された実行ファイルは総じて 1 回目の実行速度が遅い.
- 1 回目は直接メモリから読み込むから少し遅く,
- 2 回目以降はキャッシュから読み込むから速いのだろう.

実験結果

gcc -O3 -S によって出力されたアセンブリコードの一部を示す．xmm というものがベクトルレジスターと呼ばれるものである．

```
.L4:
 movdqu (%rdx,%rax), %xmm0
 addl $1, %esi
 movdqa %xmm1, %xmm2
 psubd %xmm0, %xmm2
 movdqu %xmm2, (%rcx,%rax)
 addq $16, %rax
 cmpl %esi, %r8d
 ja .L4
 cmpl %r9d, %edi
 movl %r9d, %esi
```

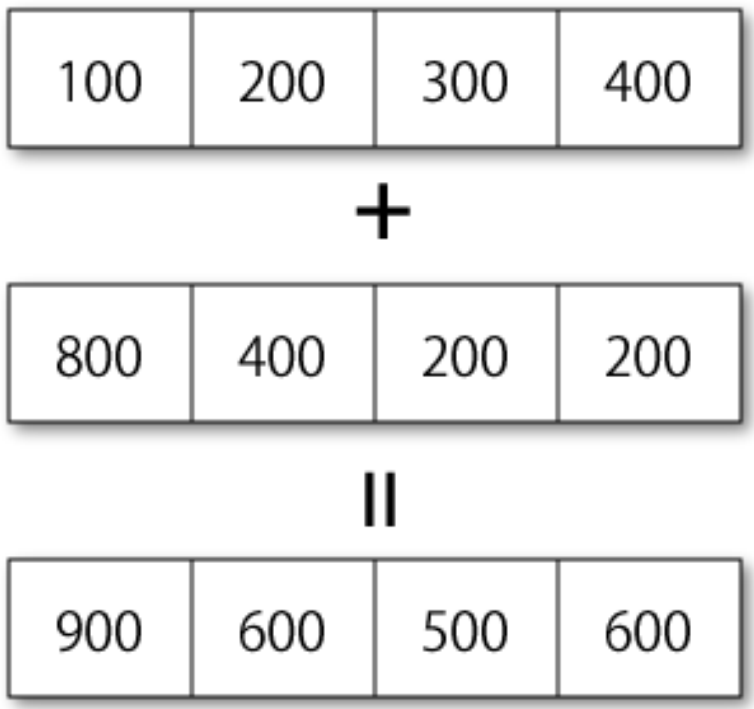
考察

- gcc -O3 でコンパイルすることにより，SIMD 命令が出力された.
- clang -O3 でも同様に SIMD 命令が出力された.
- 配列に対して連続アクセスをするようなよくあるパターンだったため，gcc の自動ベクトル化機構が有効に働いたと考えられる.

前提知識

SIMD 命令

同じ演算を複数のデータに対して 1 サイクルで適用する命令



前提知識

自動ベクトル化

コードを SIMD 命令を利用したコードに自動的に変換すること