INFORMATION TECHNOLOGY CENTER
THE UNIVERSITY OF TOKYO

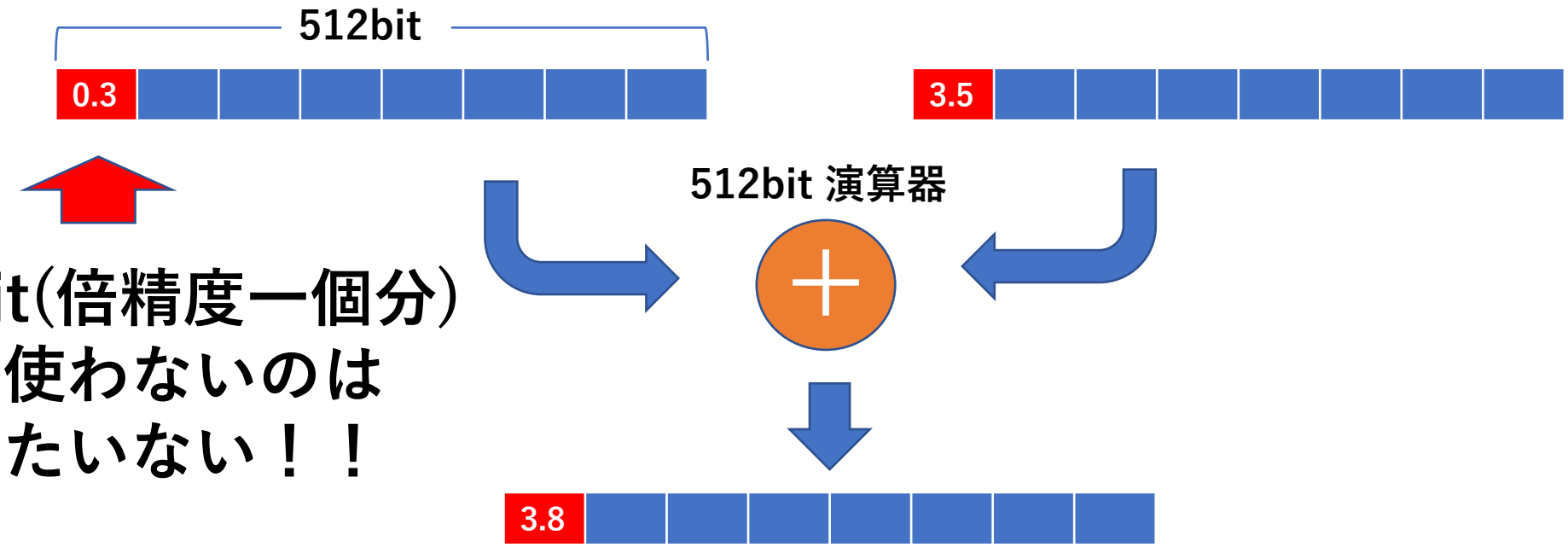
「絶対にSIMD化させる手法」を用いたフレームワークの設計

星野哲也（東京大学・情報基盤センター）

THE UNIVERSITY OF TOKYO

SIMDは#pragma omp simd だけじゃちゃんと使われない！

- 最近のCPUのSIMD長は大きい
  - e.g. Intel Cascade Lake: 512bit 演算器



64bit(倍精度一個分)しか使わないのはもったいない！！

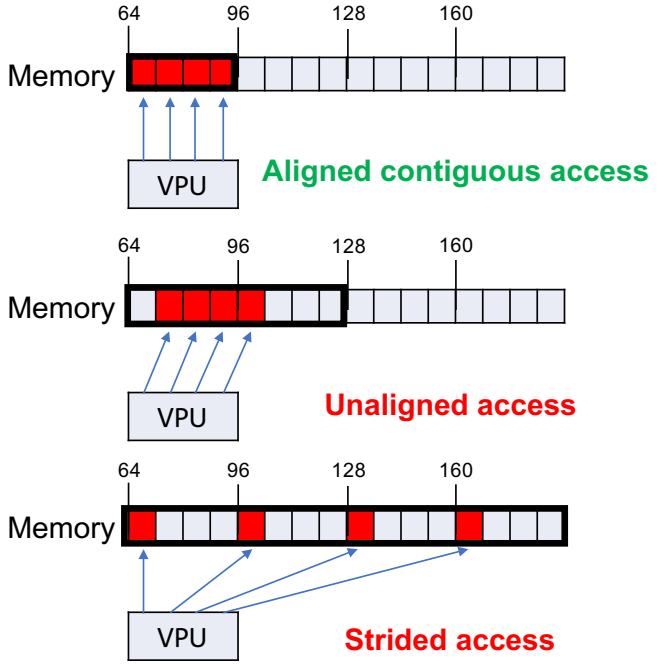
SIMDは#pragma omp simd だけじゃちゃんと使われない！

- 最近のCPUのSIMD長は大きい
  - e.g. Intel Cascade Lake: 512bit 演算器

- SIMDをちゃんと使うのはめんどくさい
  - OMP SIMDは効かないことが多い
    - SIMD化阻害要因例:
      - ループ内分岐
      - ループ内関数呼び出し
      - ループ伝搬依存がある。または**あるように見える**（コンパイラからみて）。

SIMDは#pragma omp simd だけじゃちゃんと使われない！

- 最近のCPUのSIMD長は大きい
  - e.g. Intel Cascade Lake: 512bit 演算器



- SIMDをちゃんと使うのはめんどくさい
  - OMP SIMDは効かないことが多い
    - SIMD化阻害要因例：
      - ループ内分岐
      - ループ内関数呼び出し
      - ループ伝搬依存がある。または**あるように見える**（コンパイラからみて）。
  - Memory Alignment を考えるとよりめんどう

SIMDは#pragma omp simd だけじゃちゃんと使われない！

- 最近のCPUのSIMD長は大きい
  - e.g. Intel Cascade Lake: 512bit 演算器

- SIMDをちゃんと使うのはめんどくさい
  - OMP SIMDは効かないことが多い
    - SIMD化阻害要因例:
      - ループ内分岐
      - ループ内関数呼び出し
      - ループ伝搬依存がある。または**あるように見える**（コンパイラからみて）。
  - Memory Alignment を考えるとよりめんどう
  - intrinsic はめんどくさいし Fortranでは使えない

SIMDは#pragma omp simd だけじゃちゃんと使われない！

- 最近のCPUのSIMD長は大きい

```
#pragma omp parallel for private(x,y,z,c,w,e,n,s,b,t)
      for (z = 0; z < nz; z++) {
        for (y = 0; y < ny; y++) {
          for (x = 0; x < nx; x++) {
            c =  x + y * nx + z * nx * ny;
            w = (x == 0)    ? c : c - 1;
            e = (x == nx-1) ? c : c + 1;
            n = (y == 0)    ? c : c - nx;
            s = (y == ny-1) ? c : c + nx;
            b = (z == 0)    ? c : c - nx * ny;
            t = (z == nz-1) ? c : c + nx * ny;
            f2_t[c] = cc * f1_t[c] + cw * f1_t[w] + ce * f1_t[e]
              + cs * f1_t[s] + cn * f1_t[n] + cb * f1_t[b] + ct * f1_t[t];
          }
        }
      }
```

intrinsic

```
const __m512d cc_vec = _mm512_set1_pd(cc);
const __m512d cw_vec = _mm512_set1_pd(cw);
const __m512d ce_vec = _mm512_set1_pd(ce);
const __m512d cs_vec = _mm512_set1_pd(cs);
const __m512d cn_vec = _mm512_set1_pd(cn);
const __m512d cb_vec = _mm512_set1_pd(cb);
const __m512d ct_vec = _mm512_set1_pd(ct);
do {
  int yy;
  for (yy = idy; yy < idy+1; yy++) {
    for (z = idz; z < idz + zchunk; z++) {
      b = (z == 0)    ? 0 : - nx * ny;
      t = (z == nz-1) ? 0 :   nx * ny;
      for (y = yy; y < (yy+YBF < ny ? yy+YBF:ny); y++) {
        n = (y == 0)    ? 0 : - nx;
        s = (y == ny-1) ? 0 :   nx;
        c =  y * nx + z * nx * ny;
        __m512d fc_vec   = _mm512_load_pd(f1_t+c);
        __m512d fcp1_vec = _mm512_load_pd(f1_t+c+8);
        __m512d fcm1_vec = _mm512_alignr_epi64(fc_vec,fc_vec,1);

        __m512d fcw_vec = _mm512_alignr_epi64(fc_vec,fcm1_vec,7);
        __m512d fce_vec = _mm512_alignr_epi64(fcp1_vec,fc_vec,1);
        __m512d fcs_vec = _mm512_load_pd(f1_t+c+s);
        __m512d fcn_vec = _mm512_load_pd(f1_t+c+n);
        __m512d fcb_vec = _mm512_load_pd(f1_t+c+b);
        __m512d fct_vec = _mm512_load_pd(f1_t+c+t);
        __m512d tmp = _mm512_mul_pd(cc_vec,fc_vec);
        tmp = _mm512_fmadd_pd(cw_vec,fcw_vec,tmp);
        tmp = _mm512_fmadd_pd(ce_vec,fce_vec,tmp);
        tmp = _mm512_fmadd_pd(cs_vec,fcs_vec,tmp);
        tmp = _mm512_fmadd_pd(cn_vec,fcn_vec,tmp);
        tmp = _mm512_fmadd_pd(cb_vec,fcb_vec,tmp);
        tmp = _mm512_fmadd_pd(ct_vec,fct_vec,tmp);
        _mm512_store_pd(f2_t+c,tmp);
#pragma unroll
        for (x = 8; x < nx-8; x+=8) {
          fcm1_vec = fc_vec;
          fc_vec   = fcp1_vec;
          fcp1_vec = _mm512_load_pd(f1_t+c+8+x);

          fcw_vec = _mm512_alignr_epi64(fc_vec,fcm1_vec,7);
          fce_vec = _mm512_alignr_epi64(fcp1_vec,fc_vec,1);
          fcs_vec = _mm512_load_pd(f1_t+c+x+s);
          fcn_vec = _mm512_load_pd(f1_t+c+x+n);
          fcb_vec = _mm512_load_pd(f1_t+c+x+b);
          fct_vec = _mm512_load_pd(f1_t+c+x+t);
          tmp = _mm512_mul_pd(cc_vec,fc_vec);
          tmp = _mm512_fmadd_pd(cw_vec,fcw_vec,tmp);
          tmp = _mm512_fmadd_pd(ce_vec,fce_vec,tmp);
          tmp = _mm512_fmadd_pd(cs_vec,fcs_vec,tmp);
          tmp = _mm512_fmadd_pd(cn_vec,fcn_vec,tmp);
          tmp = _mm512_fmadd_pd(cb_vec,fcb_vec,tmp);
          tmp = _mm512_fmadd_pd(ct_vec,fct_vec,tmp);

          _mm512_store_pd(f2_t+c+x,tmp);
        }
        fcm1_vec = fc_vec;
        fc_vec   = fcp1_vec;
        fcp1_vec = _mm512_alignr_epi64(fc_vec,fc_vec,7);

        fcw_vec = _mm512_alignr_epi64(fc_vec,fcm1_vec,7);
        fce_vec = _mm512_alignr_epi64(fcp1_vec,fc_vec,1);
        fcs_vec = _mm512_load_pd(f1_t+c+x+s);
        fcn_vec = _mm512_load_pd(f1_t+c+x+n);
        fcb_vec = _mm512_load_pd(f1_t+c+x+b);
        fct_vec = _mm512_load_pd(f1_t+c+x+t);
        tmp = _mm512_mul_pd(cc_vec,fc_vec);
        tmp = _mm512_fmadd_pd(cw_vec,fcw_vec,tmp);
        tmp = _mm512_fmadd_pd(ce_vec,fce_vec,tmp);
        tmp = _mm512_fmadd_pd(cs_vec,fcs_vec,tmp);
        tmp = _mm512_fmadd_pd(cn_vec,fcn_vec,tmp);
        tmp = _mm512_fmadd_pd(cb_vec,fcb_vec,tmp);
        tmp = _mm512_fmadd_pd(ct_vec,fct_vec,tmp);
        _mm512_store_pd(f2_t+c+x,tmp);
      }
    }
  }
```

(コンパ

- Memory Alignment を考えるとよりめんどう
- intrinsic はめんどくさいし Fortranでは使えない

OMP SIMDで絶対にSIMD化させる手法

- 要は、コンパイラから見てSIMD阻害要因がなければいい
- OMP DECLARE SIMD 指示文を効果的に使うと、コンパイラからみて明らかにSIMD化可能なコードが作れる
  - この手法は、よく探すとIntel Developer Zoneで”Explicit Vector Programming in Fortran”として紹介されてる
    - https://software.intel.com/content/www/us/en/develop/articles/explicit-vector-programming-in-fortran.html

OMP SIMDで絶対にSIMD化させる手法

```
module mymod
contains
  subroutine vec(a,b,c,alpha,beta)
!$omp declare simd(vec) linear(a,b,c) uniform(alpha,beta)
      double precision, intent(in) :: a,b,alpha,beta
      double precision, intent(out) :: c
    c = alpha * a + beta * b
  end subroutine
end module

...
use mymod
double precision :: a(N),b(N),c(N),alpha,beta
...
!$omp simd
do i = 1, N
  call vec(a(i),b(i),c(i),alpha,beta)
end do
```

関数の引数はすべてスカラ

関数中にmoduleなどからの大域アクセスや、save属性を持つ変数などを持たない新たな配列を生成しない
さらに関数を呼び出す場合、その関数もdeclare simdで並列化されている必要がある

やりたい計算

エイリアスを持たない配列のi番目要素またはスカラ

ループボディを関数呼び出しに置き換えてしまう

確実にSIMD化されるFortran疑似コード

絶対にSIMD化させる手法の
フレームワークへの応用

```
!$omp parallel do
   do j = 1, N
    !$omp simd
    do i = 1, N
      a(i,j) = user_func(i,j,st_bemv)
    end do
   end do
!$omp end do
```

ユーザ定義の関数（境界要素法の積分核を計算する）

ユーザ定義の構造体

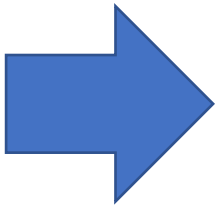
境界要素法のためのフレームワーク
ppOpen-BEMの疑似コード

このコードはSIMD化されない

絶対にSIMD化させる手法のフレームワークへの応用

```
!$omp parallel do
   do j = 1, N
     !$omp simd
     do i = 1, N
       a(i,j) = user_func(i,j,st_bemv)
     end do
   end do
!$omp end do
```

境界要素法のためのフレームワーク
ppOpen-BEMの疑似コード



```
real(8),dimension(SIMDLENGTH) :: ans
real(8),dimension(SIMDLENGTH) :: arg1,arg2,...
do j = 1, N
  do i = 1, N, SIMDLENGTH
    ii = 1
    do jj = i, min(i+SIMDLENGTH-1, N)
      arg1(ii) = st_bemv%x(i,j)
      arg2(ii) = st_bemv%y(i,j)
      ...
      ii = ii+1
    end do
    !$omp simd
    do ii = 1, SIMDLENGTH
      call vectorize_func(arg1(ii),arg2(ii),...,ans(ii))
    end do
    ii = 1
    do jj=i,min(i+SIMDLENGTH-1, N)
      a(i,j) = ans(ii)
      ii = ii+1
    end do
  end do
end do
```

新たに宣言する
SIMDLENGTH長の配列

データ読み
込み部

絶対にSIMD化される計算部

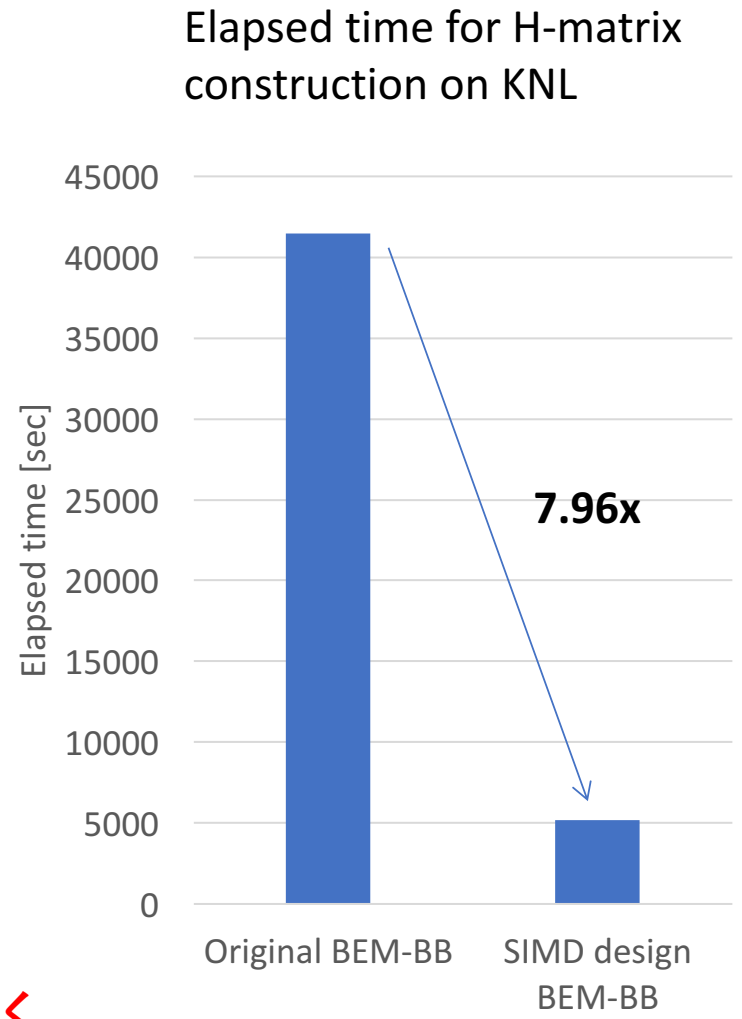
データ書き
込み部

SIMD対応版フレームワーク

どんなコードに有用？

- 計算＞メモリ読み書きなコード
  - メモリ読み書き部分は効率化されないため

```
      dvc (3, 1, 2) = (((((((-((( y2 * temp62 )*(((-((2* a )* y2 ))/
. temp181 )-( y2 / temp171 ))))/ rb ))-(( temp62 * temp138 )/ rb
. ))+((( temp3 * temp62 )* temp138 )/ temp63 ))+( temp49
. *((((((-(((1+ temp64 )* temp3 )/ temp171 ))+((1+ temp64 )/ temp33
    . ))-((  a * temp3 )/ temp149 ))+(( temp106 * temp3 )/ temp158 ))-(
. temp106 / temp38 ))+((( a  *  cb )* temp3 )/ temp159 ))))-((( temp7
. * temp62 )* temp139 )/( temp16 * temp157 )))+((( cb * temp62 )*
. temp139 )/ temp39 ))-((( temp7 * temp62 )* temp139 )/ temp159
    . ))+((((  cb * y2 )* temp62 )*(((((-(((2* a )* y2 )* yb3 ))/
. temp181 )-((( temp78 * y2 )* temp59 )/ temp158 ))+(( temp106 * y2
. )/ temp39 ))-((( a * y2 )* temp59 )/ temp159 ))))/ temp39 ))
```



とある地震動シミュレーションのコードの一部これで１行。100行以上続く。

Numerical Evaluations

T. Hoshino et al. “Design of Parallel BEM Analyses Framework for SIMD Processors” (ICCS 2018)

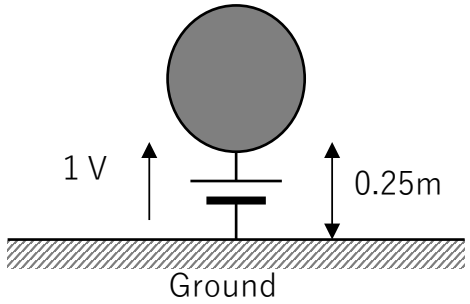
■ Test model of electrostatic field analysis

- ▸Perfect conducting sphere
- ▸Dielectric sphere
  - including branch divergence

$$P[u](x) := \int_\Omega \frac{1}{4\pi\|x-y\|} u(y)\mathrm{d}y, x \in \Omega$$

$$D[u](x) := \int_\Omega \frac{\langle x-y, n(y)\rangle}{4\pi\|x-y\|^3} u(y)\mathrm{d}y, x \in \Omega$$

User-defined functions depend on these integral equations

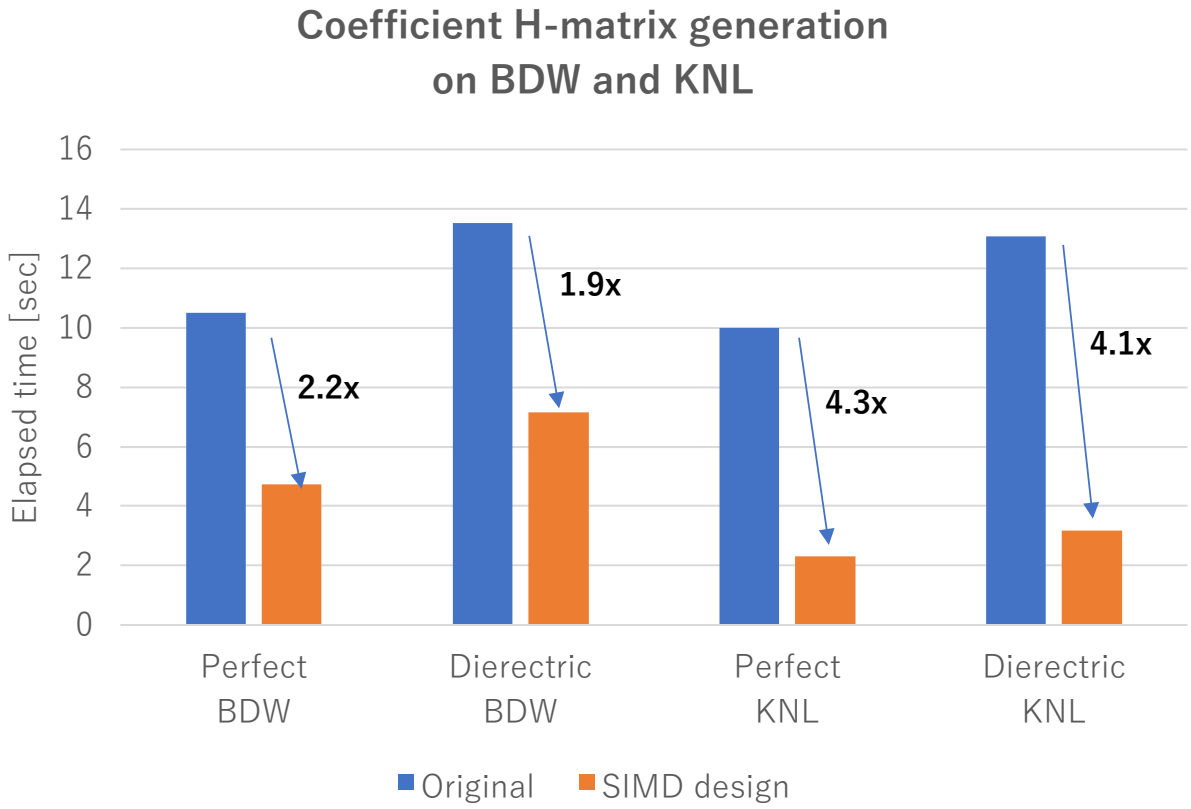


■ Evaluation Environments

- ▸BDW : Intel Xeon E5-2695 v4, 18 core
- ▸KNL : Intel Xeon Phi 7250, 68 core
- ▸Compiler : Intel compiler 18.0.1
  - -qopenmp -O3 -ipo -align array64byte -xAVX2 (BDW) –xMIC-AVX512 (KNL)

■ Performance comparison

- ▸BDW : approximately **2x** speedup
- ▸KNL : over **4x** speedup
  - In the case of dense matrix generation, new design achieved at most **6.6x** speedup



SIMD化挑戦してみてください

Thank you for watching