SIMD化とは何か

2020/04/28

慶應義塾大学理工学部物理情報工学科
渡辺

SIMDとは何か

Single Instruction Multiple Dataの略

直訳すると「一つの命令、複数のデータ」(※)

1サイクルで複数の計算を
同時に行うための工夫の一つ

※フリンの分類(Flynn's taxonomy)の一つだが、気にしなくて良い

SIMDとは何か

科学計算に使われる汎用CPUは、ほぼSIMDを採用している

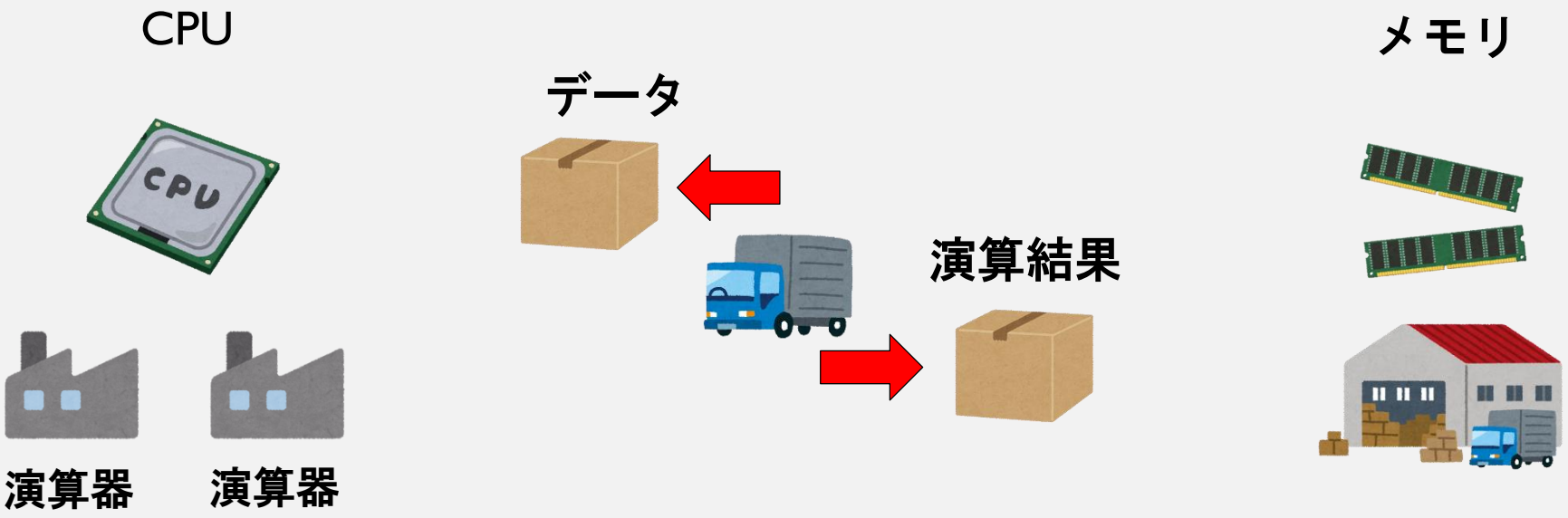
なぜSIMDが必要か？
SIMD化とは何か？
どうやってSIMD化するか？

計算機の仕組み

計算機とは

メモリからデータと命令を取ってきて
演算機に投げ
演算結果をメモリに書き戻す

装置のこと

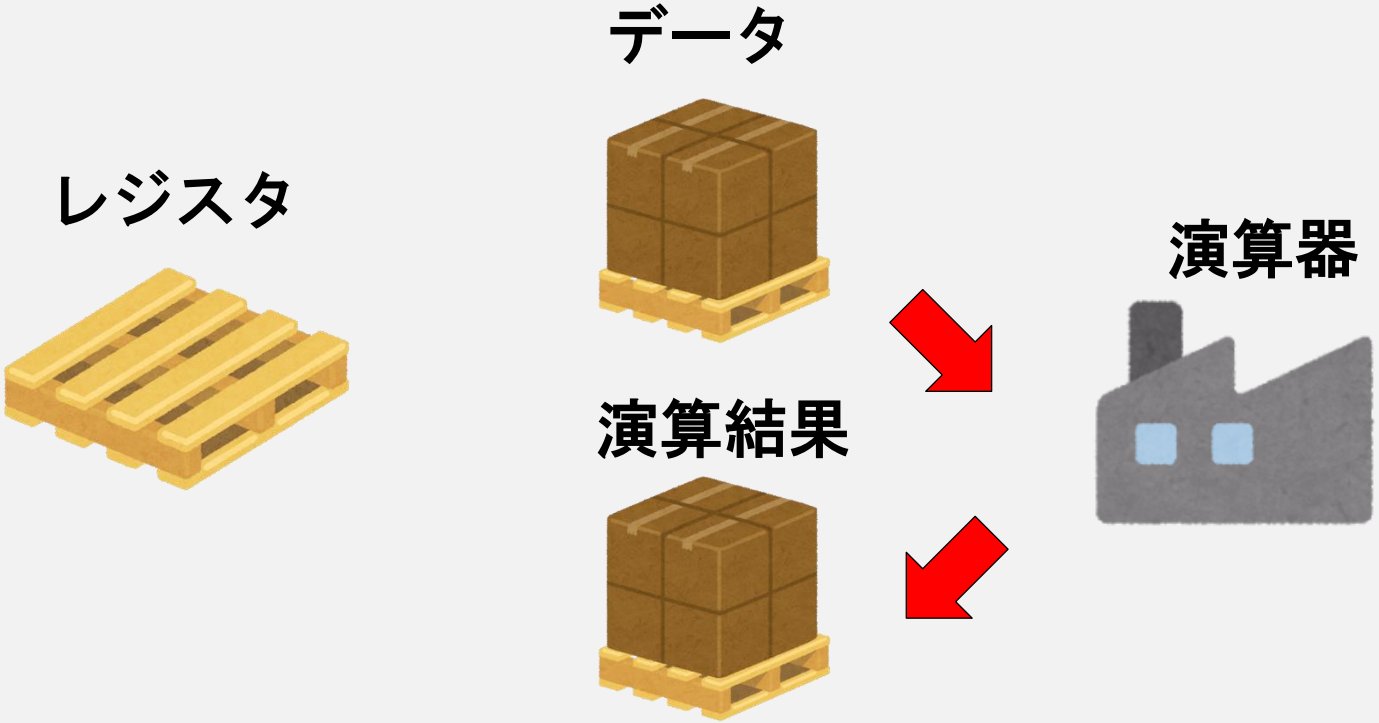


計算機の仕組み

計算機は

データをレジスタに載せて演算器に投げる

ことで計算する



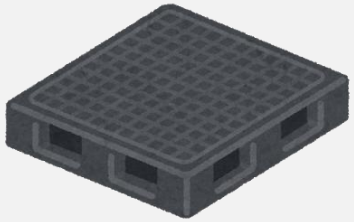
データをレジスタに載せて計算し、結果もレジスタに帰ってくる

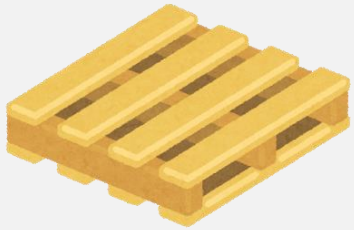
計算機の仕組み

整数と浮動小数点数は異なるレジスタ、異なる演算器を使う

整数レジスタ

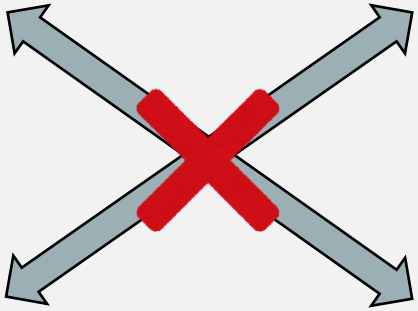
浮動小数点レジスタ





整数レジスタ
しか受け付けない

浮動小数点レジスタ
しか受け付けない



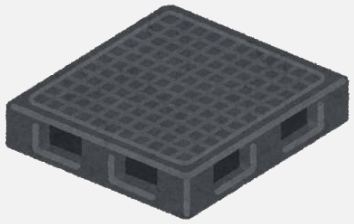
整数演算器

浮動小数演算器

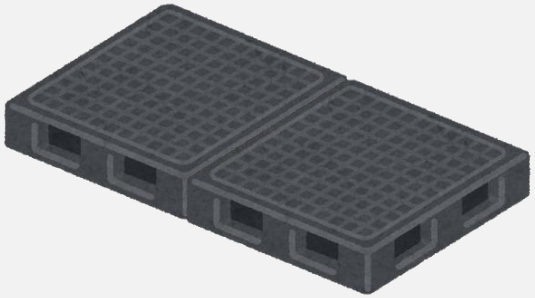
計算機の仕組み

レジスタには**長さ**がある
レジスタの長さは**ビット数**(bit)で表す

32ビットレジスタ



64ビットレジスタ



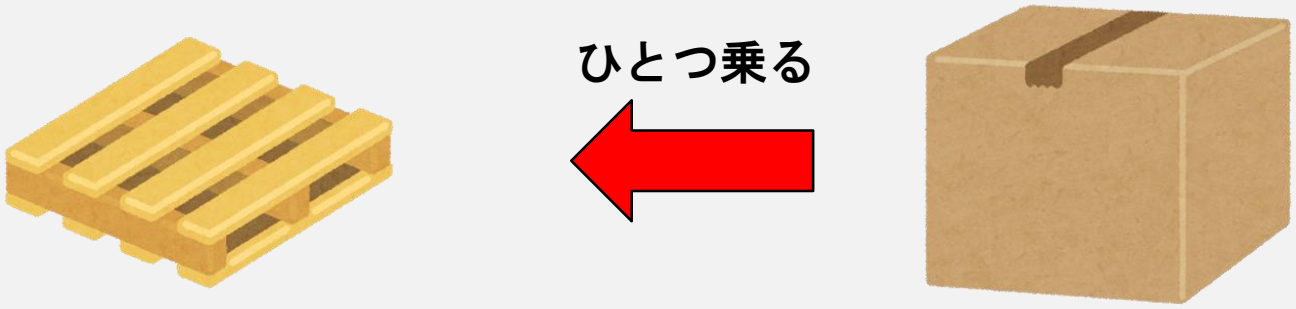
ハードウェアやソフトウェアの「ビット数」は
対応する**整数レジスタ**のビット数で決まる

ファミコン 8ビット → スーファミ 16ビット → プレステ 32ビット → NINTENDO64 64ビット → ・・・

計算機の仕組み

数値計算では、主に倍精度実数を用いる
倍精度実数は64ビットで表現される

64ビット浮動小数点レジスタ　　倍精度実数 (64ビット)

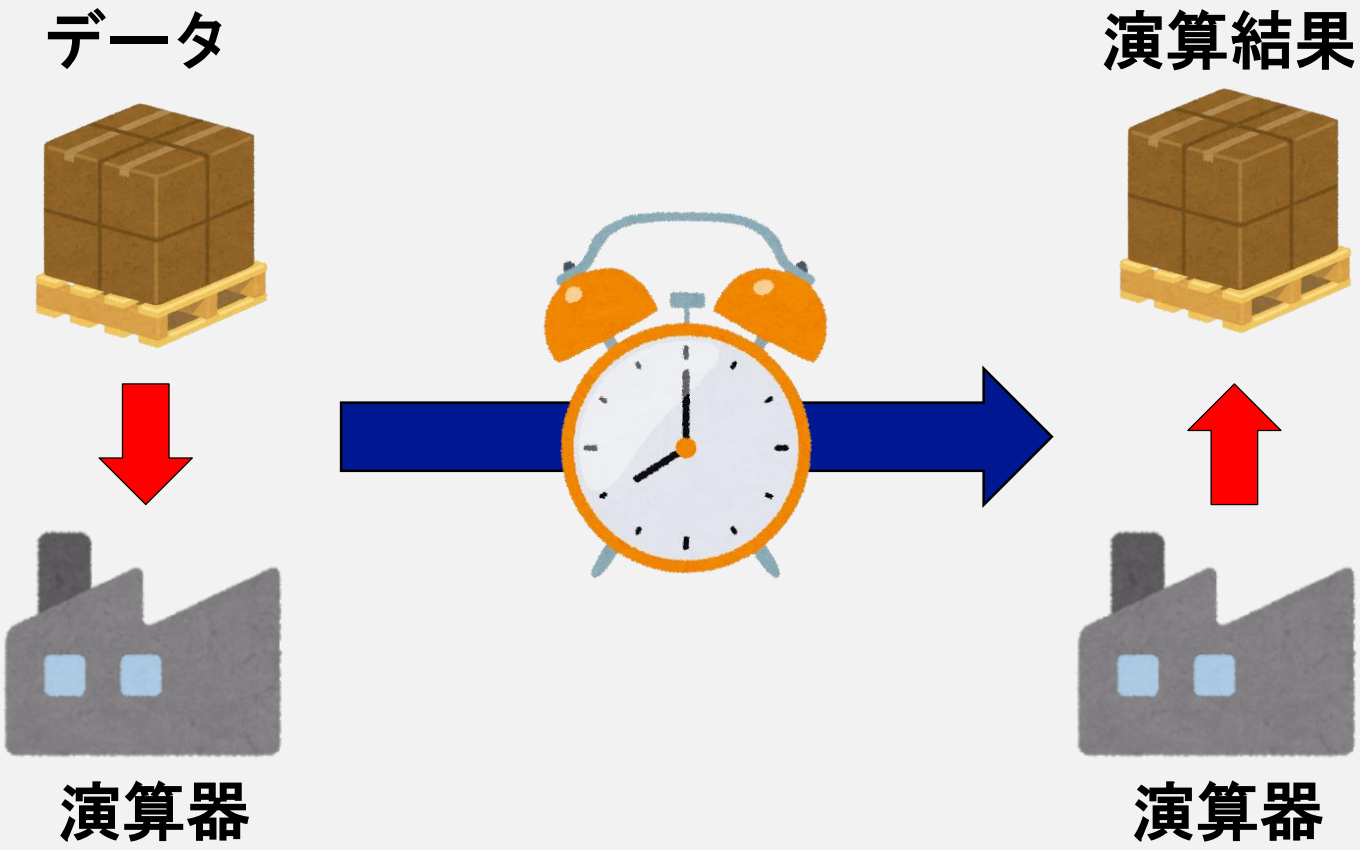


数値計算に用いるCPUの多くは64ビット整数レジスタと
64ビット浮動小数点レジスタを持つ

ただし、x86系のCPUは歴史的事情により64ビット浮動小数点
レジスタを持たず、128ビットSIMDレジスタを使う

パイプライン処理

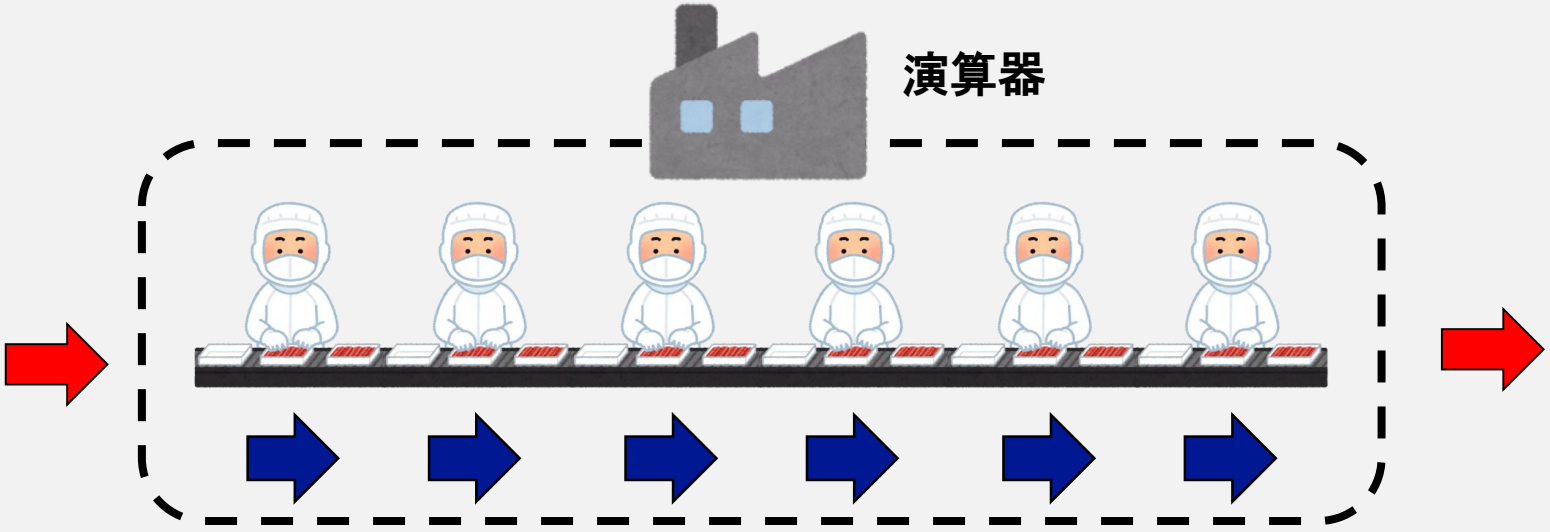
演算器に計算を投げてから結果が返ってくるまで時間がかかる
この時間をレイテンシと呼び、サイクル数で測る



浮動小数点演算なら、加減乗算で3〜6サイクル程度。除算は遅い(10〜20サイクル)

パイプライン処理

全部で6工程ある作業を6人で分担すれば
1サイクルに1つ製品を作ることができる



1サイクルに1段右に動くベルトコンベア

演算器に入ってから出てくるまでは6サイクル(レイテンシ)
演算器から毎サイクル結果が出てくる(スループット)

CPUの動作周波数

パイプライン処理により、1サイクルに1回計算できるようになった

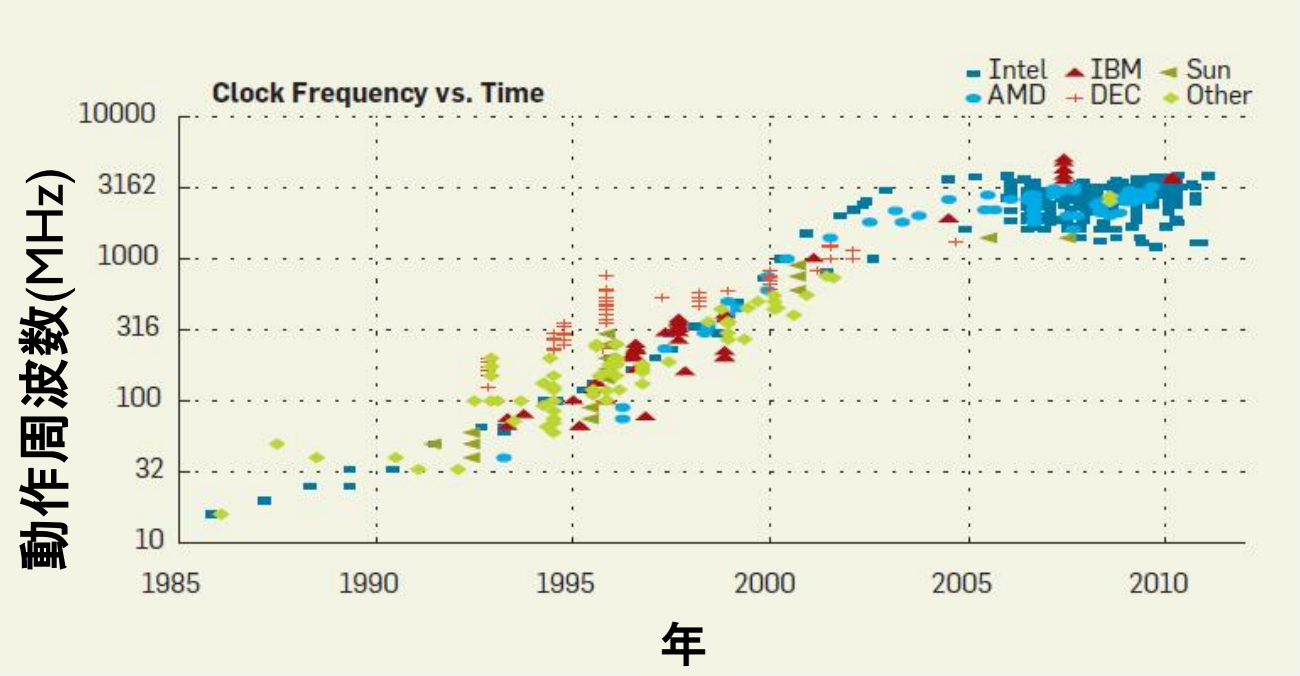
性能 ＝ **動作周波数**

あとは動作周波数を上げれば上げるだけ性能があがる

・・・はずだった

CPUの動作周波数

CPUの動作周波数向上は2000年頃から頭打ちに



http://cacm.acm.org/magazines/2012/4/147359-cpu-db-recording-microprocessor-history/fulltext

主に発熱が原因

動作周波数を上げずに演算性能を上げたい

➡ 1サイクルに複数の命令を実行するしかない

解決案1:スーパースカラ

ハードウェアにがんばらせる

演算機

データフェッチ　　依存関係チェック

演算機

データと命令を複数持ってきて
複数の生産ラインに振り分ける

この人が過労死する

命令の後方互換性を保てる

実行ユニットが増えると命令振り分けで死ぬ

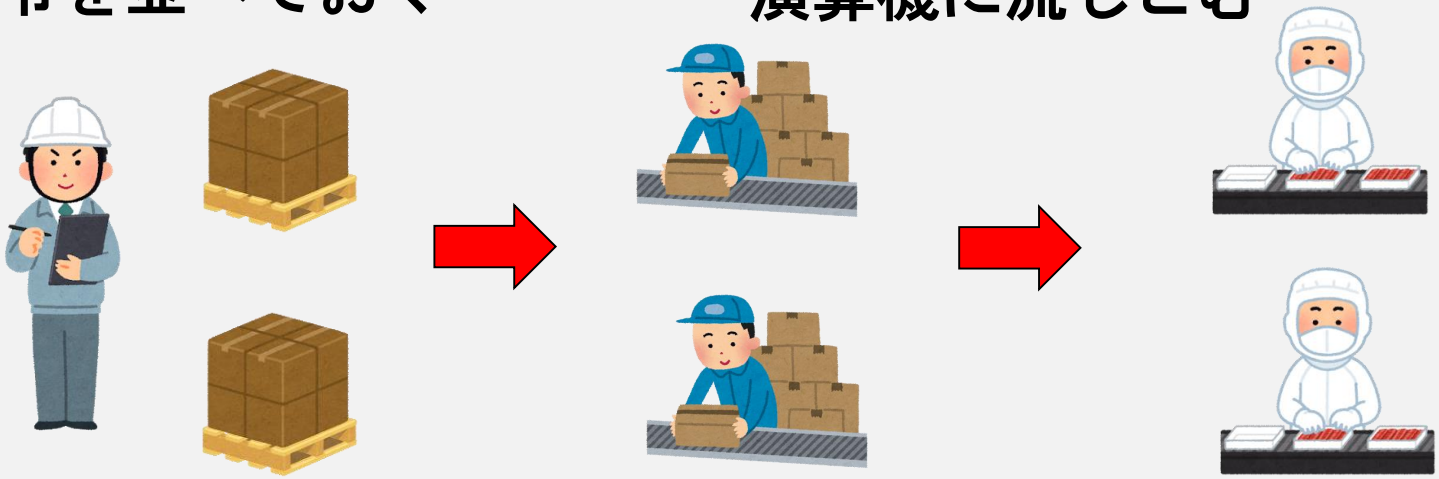
解決案2：VLIW

※Very Long Instruction Word

ソフトウェアにがんばらせる

コンパイラがデータと
命令を並べておく

それをノーチェックで
演算機に流しこむ



依存関係チェックが不要→ハードウェアが簡単に

神のように賢いコンパイラが必要
後方互換性を失う

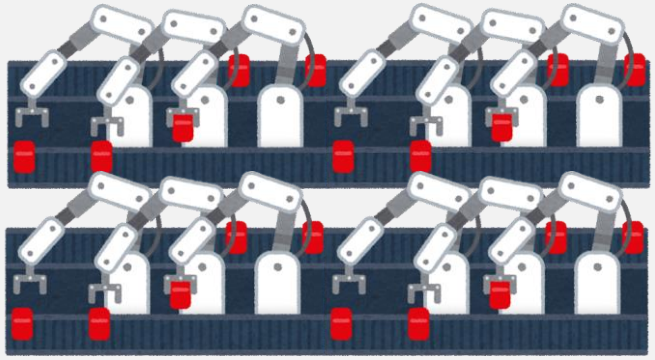
組み込み向けでは人気も
HPC向けとしてはほぼ絶滅

解決案3：SIMD

プログラマにがんばらせる

プログラマが
データを並べておく

それをノーチェックで
演算機に流しこむ



一度に2～8演算を行う

ハードウェアは簡単
後方互換性も保てる

コンパイラによる自動SIMD化には限界がある
プログラムが大変

「なぜSIMDが必要か」のまとめ

パイプライン処理により、1サイクルに1命令実行できる
CPUの動作周波数は限界に達しており、これ以上あがらない

1サイクルに複数の命令を実行するしかない

ハードやソフトにがんばらせる方法も限界

人間ががんばるしかない ←イマココ

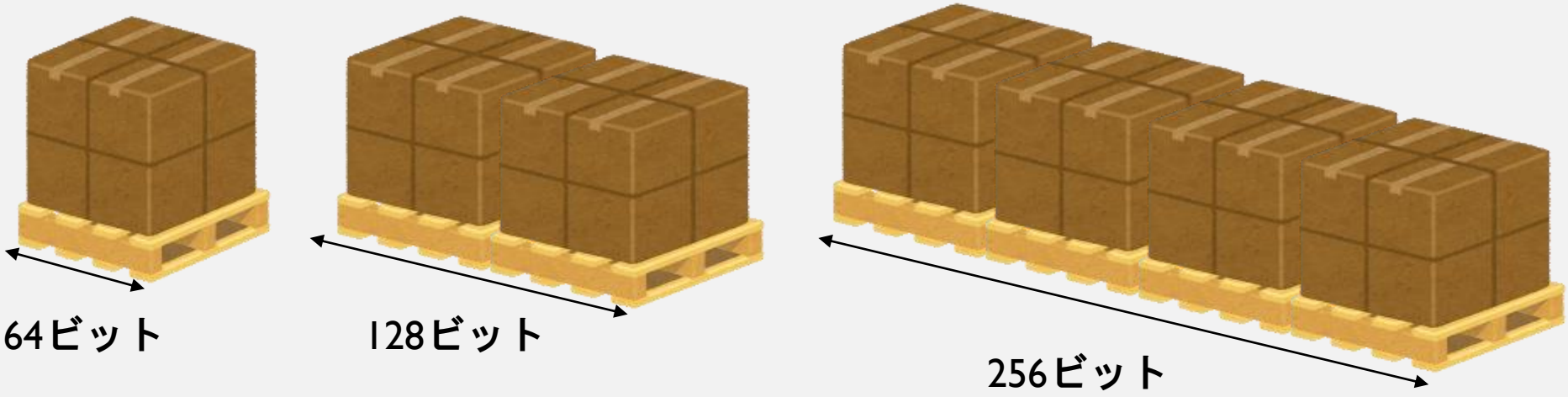
SIMDの仕組み

1時間に1個製品ができる製造ラインがある
ただし、コンベアの速度はもう上がらない

じゃあ製造ラインの**幅**を倍にすれば良いじゃん

SIMDの仕組み

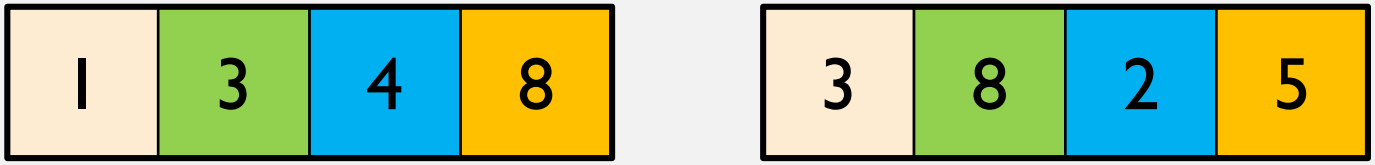
- 64ビットレジスタは倍精度実数を一つ載せることができる
- 128ビットレジスタなら、二つ載せることができる
- 256ビットレジスタなら、四つ載せることができる



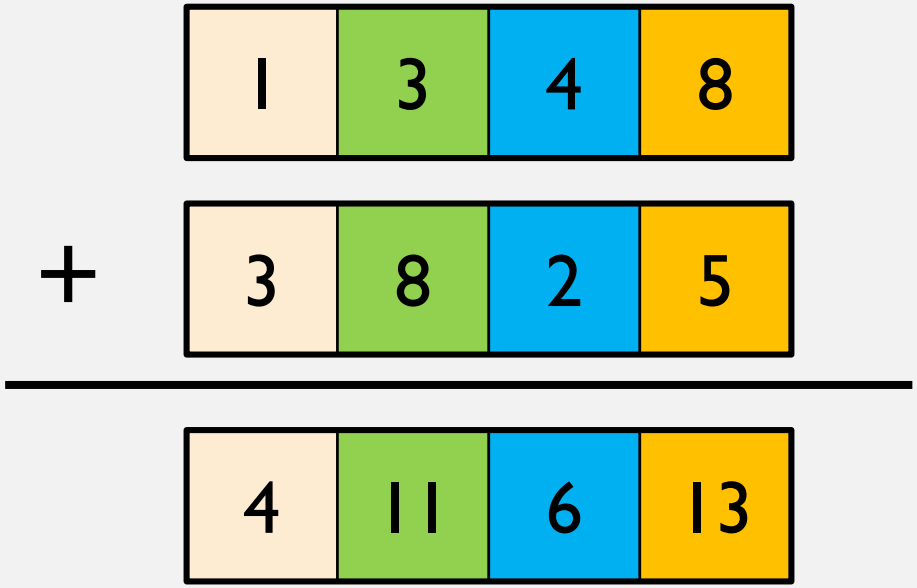
ビット幅が広く、データを一度に複数載せることができるレジスタをSIMDレジスタと呼ぶ

SIMDの仕組み

2つのレジスタに4つずつ値を載せる(256ビットの場合)

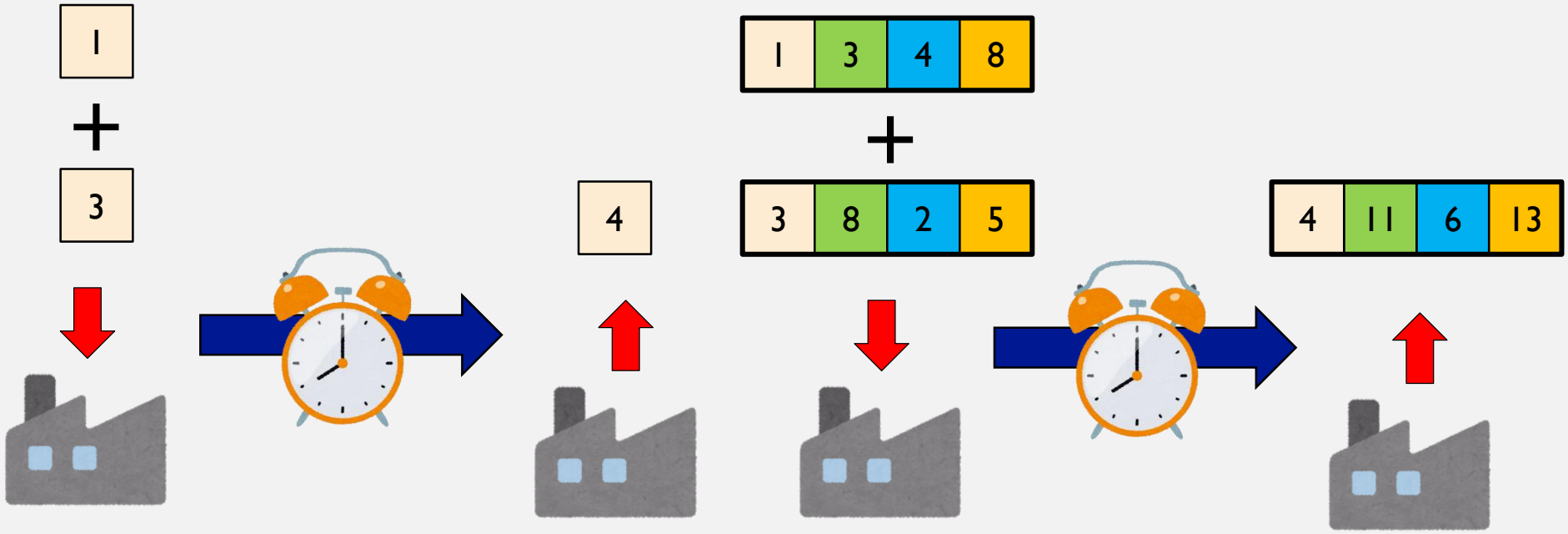


「同じ位置」同士で同時に独立な演算をする



SIMDの仕組み

一つの計算をするのと同じ時間で
複数の計算を同時に実行できる

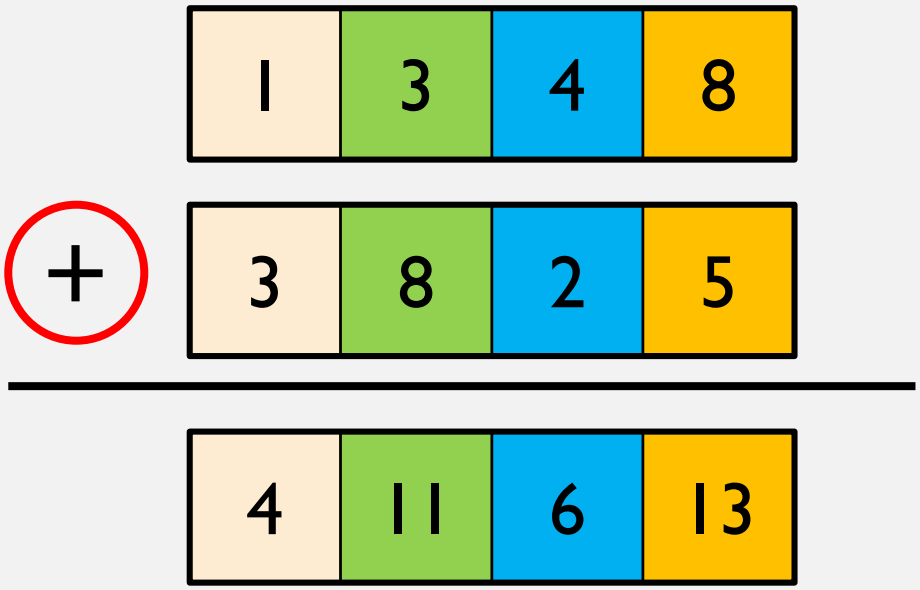


CPUの理論ピーク性能は「SIMD幅を使い切った時」の値

➡ SIMDが使えていなければ、数分の一の性能しか出せない

SIMDの注意

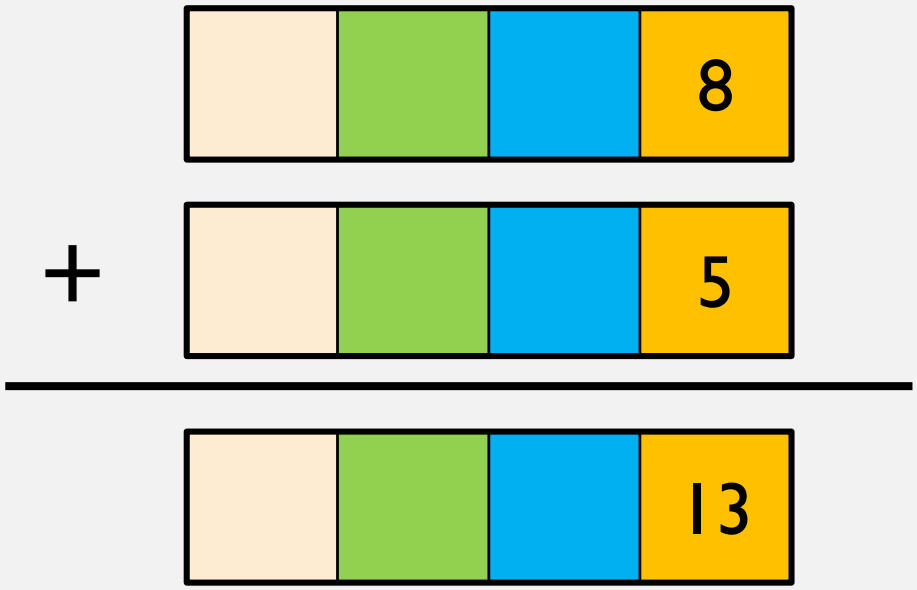
各位置ごとに異なる演算はできない



複数のデータ(Multiple Data)に
単一の演算 (Single Instruction)を実行するから
SIMD (Single Instruction Multiple Data)

SIMDの注意

使っていない位置は無駄になる



なるべくSIMDレジスタにデータを詰め込んで一度に計算したい

SIMD化とは

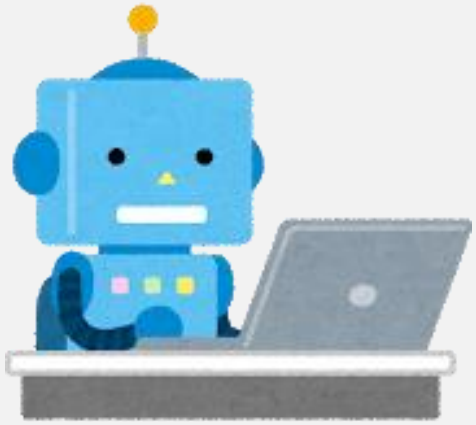
SIMDレジスタをうまく使えていないプログラムを
SIMDレジスタを活用するように修正し
性能を向上させること

SIMDベクトル化 (SIMD Vectorization)とも

SIMD化の方法

1. コンパイラにSIMD化してもらう
2. 自分でSIMD化する

基本的にこの二択



コンパイラにSIMD化してもらう

最近のコンパイラは簡単なコードなら勝手にSIMD化してくれる

たとえばこんなファイルを用意する

test.cpp

```
const int N = 10000;
double a[N], b[N];

void func(void){
  for(int i=0;i<N;i++){
    a[i] += b[i];
  }
}
```

コンパイラにSIMD化してもらう

Intelコンパイラに食わせて、最適化レポートを出力

```
icpc -O3 -qopt-report=2 -c test.cpp
```

test.cpp

```
const int N = 10000;
double a[N], b[N];

void func(void){
  for(int i=0;i<N;i++){
    a[i] += b[i];
  }
}
```

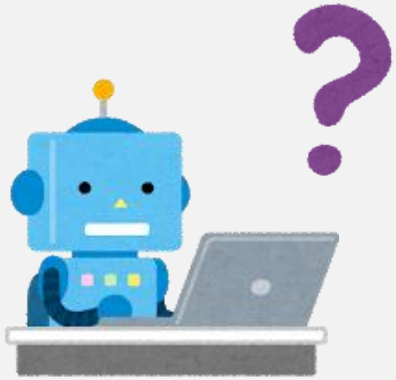
test.optrpt

```
LOOP BEGIN at test.cpp(5,3)
   remark #15300: LOOP WAS VECTORIZED
LOOP END
```

test.cppの5行目のループを
ベクトル化したよ

コンパイラにSIMD化してもらう

うまくSIMD化できなかった時、レポートを見ながら
SIMD化のヒントを出したり、コードを修正したりする



少しの修正で性能が出る場合は良いが、これでがんば
るくらいなら自分でSIMD化したほうが早い場合が多い

自分でSIMD化する

自分でSIMD命令を書くことでSIMD化する

SIMD命令はアセンブリなのでアセンブリで
コードを書くことになる

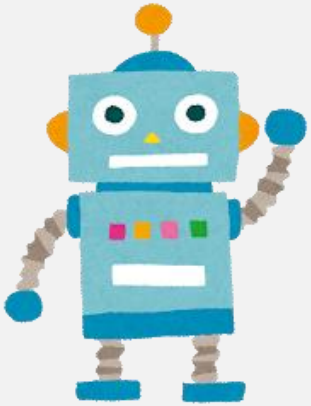
実際にはアセンブリに対応したC言語の関数を呼ぶ

```
v4df vdq_1_b = (vqj_1 - vqi);
v4df vdq_2_b = (vqj_2 - vqi);
v4df vdq_3_b = (vqj_3 - vqi);
v4df vdq_4_b = (vqj_4 - vqi);
tmp0 = _mm256_unpacklo_pd(vdq_1_b, vdq_2_b);
tmp1 = _mm256_unpackhi_pd(vdq_1_b, vdq_2_b);
tmp2 = _mm256_unpacklo_pd(vdq_3_b, vdq_4_b);
tmp3 = _mm256_unpackhi_pd(vdq_3_b, vdq_4_b);
vdx = _mm256_permute2f128_pd(tmp0, tmp2, 0x20);
vdy = _mm256_permute2f128_pd(tmp1, tmp3, 0x20);
vdz = _mm256_permute2f128_pd(tmp0, tmp2, 0x31);
```

SIMD化は面倒くさい

SIMD命令はCPUによって異なる
→CPUごとにプログラムを書き換えないといけない

旧世代CPU



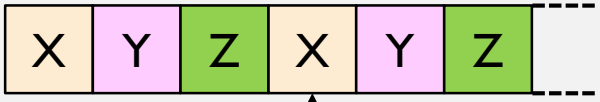
次世代CPU

SIMD化は面倒くさい

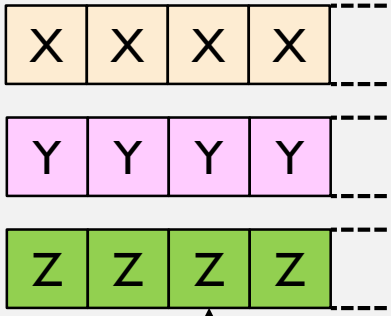
場合によって最適なデータレイアウトが異なる

例：三次元の座標データを持つ原子のまとまりを表現したい

方法1: 順番に並べていく
Array of Structure (AoS)



方法2: 同じ成分をまとめる
Structure of Array (SoA)



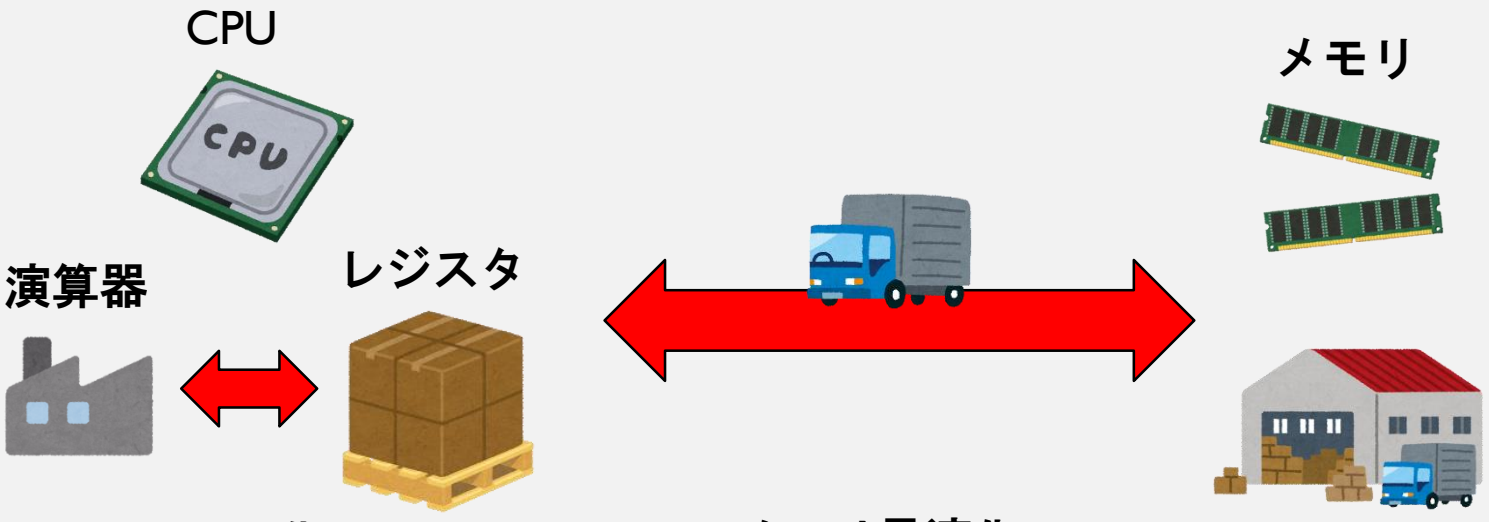
どちらが良いか、CPUやプログラムごとに違う

SoAとAoSの変換は、コードをほぼ全て書き直しに・・・

SIMD化で注意すべきこと

SIMD化の目的はSIMD化率を上げることではなく
実行性能を向上させること

多くの場合、メモリ転送がボトルネック
SIMD化率を上げてもキャッシュ効率が低下したら性能は落ちる



まとめ

- SIMDは1サイクルに複数の命令を実行するための工夫の一つ
- SIMDは幅広レジスタに複数のデータを載せて同時に独立な計算を実行する
- SIMD化の目的はSIMDレジスタを活用することで性能を向上させること
- SIMD化は、コンパイラに任せる方法と、自分で書く方法がある

SIMD化は面倒だが難しくない

まずはやってみよう