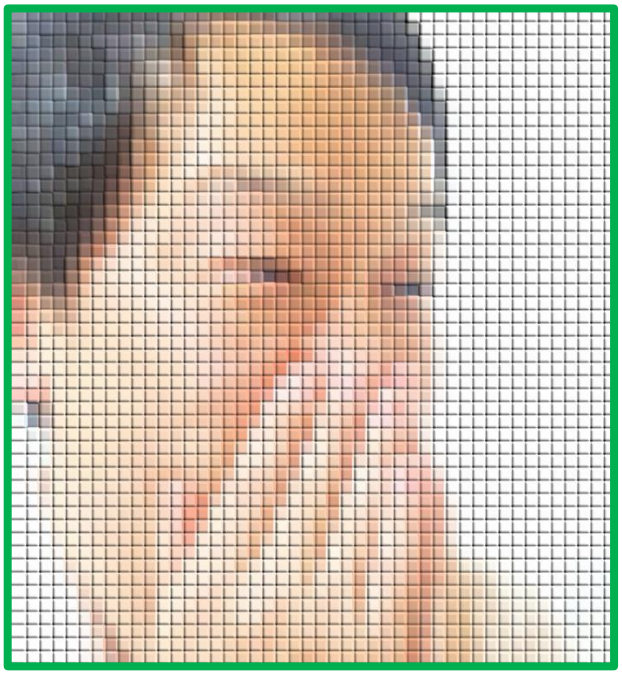
うわっ・・・私のプログラム、性能低すぎ・・・？



東北大学大学院情報科学研究科
2016年6月9日　情報談話会

滝沢 寛之 〈takizawa@tohoku.ac.jp〉

アーキテクチャ学分野・准教授

CHECK!

>> あなたのプログラムの適正性能は？

自己紹介

- **滝沢寛之 (たきざわひろゆき)**
  - 東北大学大学院情報科学研究科
    - 情報基礎科学専攻アーキテクチャ学分野 准教授
  - 東北大学工学部機械・航空工学科(兼担)
  - 東北大学サイバーサイエンスセンター(兼担)

専門分野は高性能計算
ハイパフォーマンスコンピューティング(HPC)

発表の構成

- 背景：性能をまじめに考える重要性
- 課題：スパコン開発に立ちはだかる壁
- 研究紹介：我々の取り組み
- 将来：スパコンが切り開く未来

性能をまじめに考える重要性

The free lunch is over（Sutter@2005）

- **よくあるQ&A**

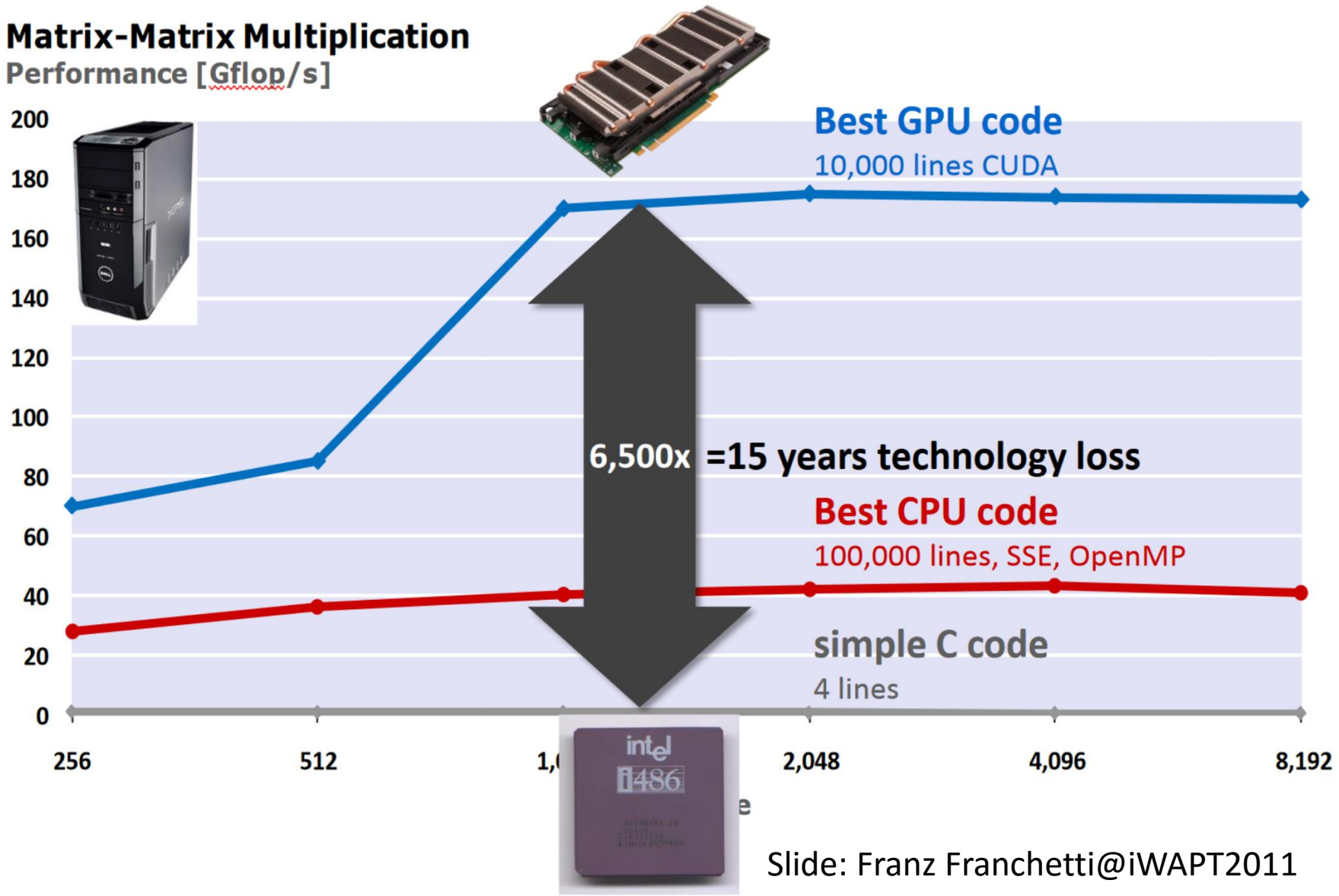
  Q: **コンピュータの性能**が年々向上するから、**プログラムの性能**(and/or 性能向上)なんてまじめに考えなくても問題ないのでは？

  A: そんなただ飯のサービス期間は終わりますよ？

- **現代に生きる我々にとって**
  - **コンピュータの性能向上 ≠ あなたのプログラムの性能向上**
    - プロセッサの動作周波数はもはやほとんど向上しない
    - カタログ性能向上はその他の「工夫」によって達成されている

  → プログラムの書き方によって実際の性能が大きく変化

The Cost Of Portability and Maintainability



Slide: Franz Franchetti@iWAPT2011

「適正性能」と落とし穴

- 例) ○○したら100倍速くなりました！
  - そんな論文がうんざりするほどある
  - しかし・・・
    - それって**基準となるプログラムが遅かっただけ**かも？

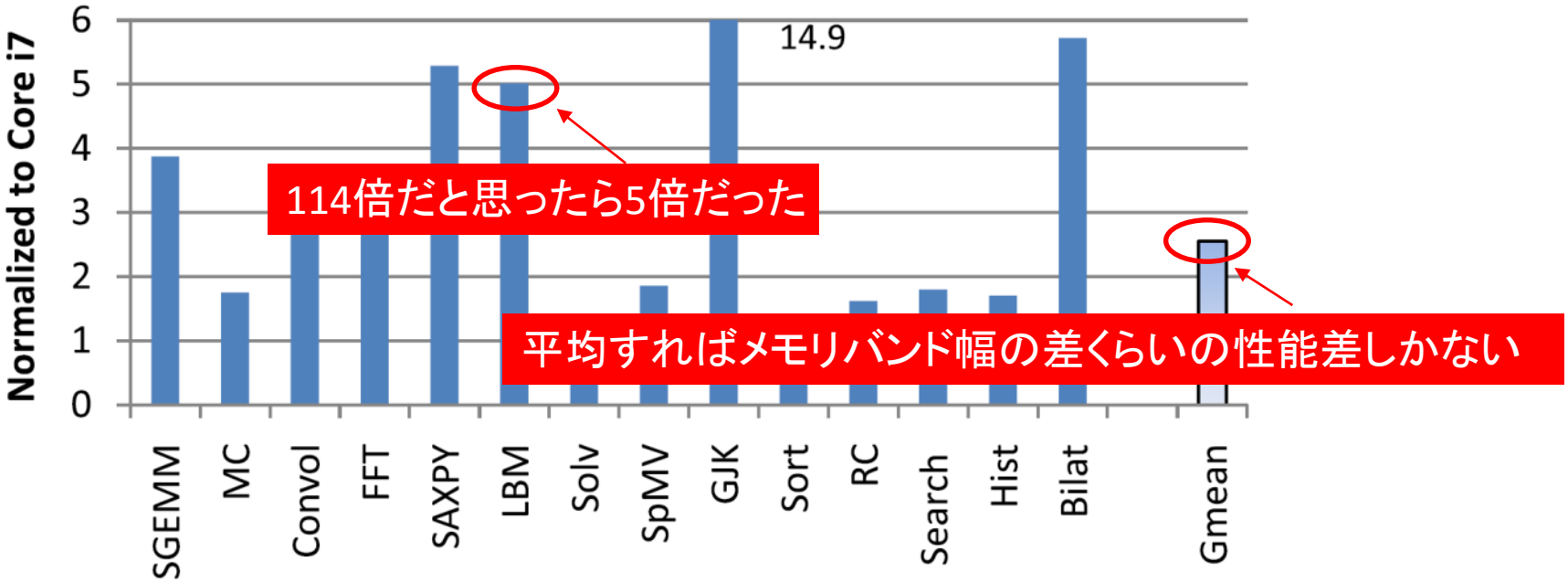


Figure: Lee et al. ISCA'10

あなたのプログラムの適正性能は？

- **適正性能を考えることは極めて重要** CHECK!
  - 利用者：システムの能力を十分活用
  - 開発者：無駄な労力を費やす危険性を回避
  - 研究者：アイディアの価値を正当に評価
- **HPCの重要性・必要性**
  - 現代のコンピュータでは得手不得手による性能差が極端
    - コンピュータ設計の大原則は make the common case fast
  - 理論的によいアルゴリズムの適正性能が高いとは限らない

    → 理論・アルゴリズムに加えて実装方法の検討も必要

    = HPCの知識が求められる

スパコン開発に立ちはだかる壁

スパコンとは？

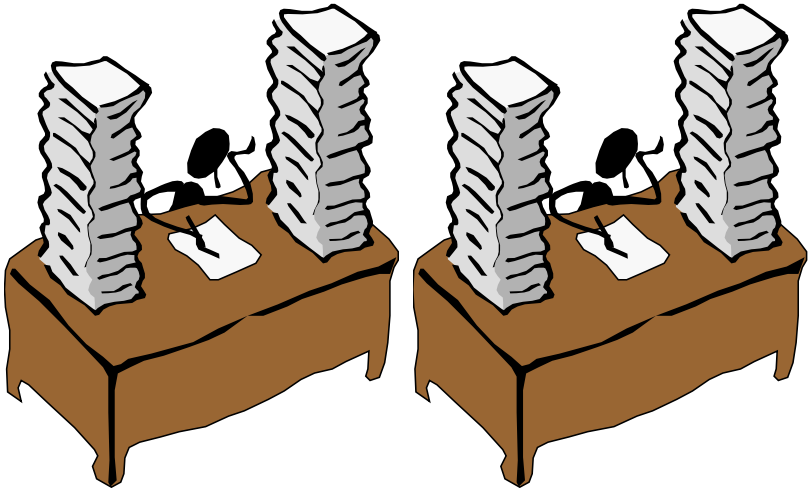
- スーパーコンピュータの定義は時代に沿って大きく変化してきている。一般的にはその時代の最新技術が投入された最高クラスの性能の計算機を指す。現時点では一般的に使用されるサーバ機よりも浮動小数点演算が1,000倍以上の速さのコンピュータを「スーパーコンピュータ」と呼ぶことが多い。(Wikipedia)

普通のコンピュータよりも超(super)速いコンピュータ

どうやって速くする？

並列化！

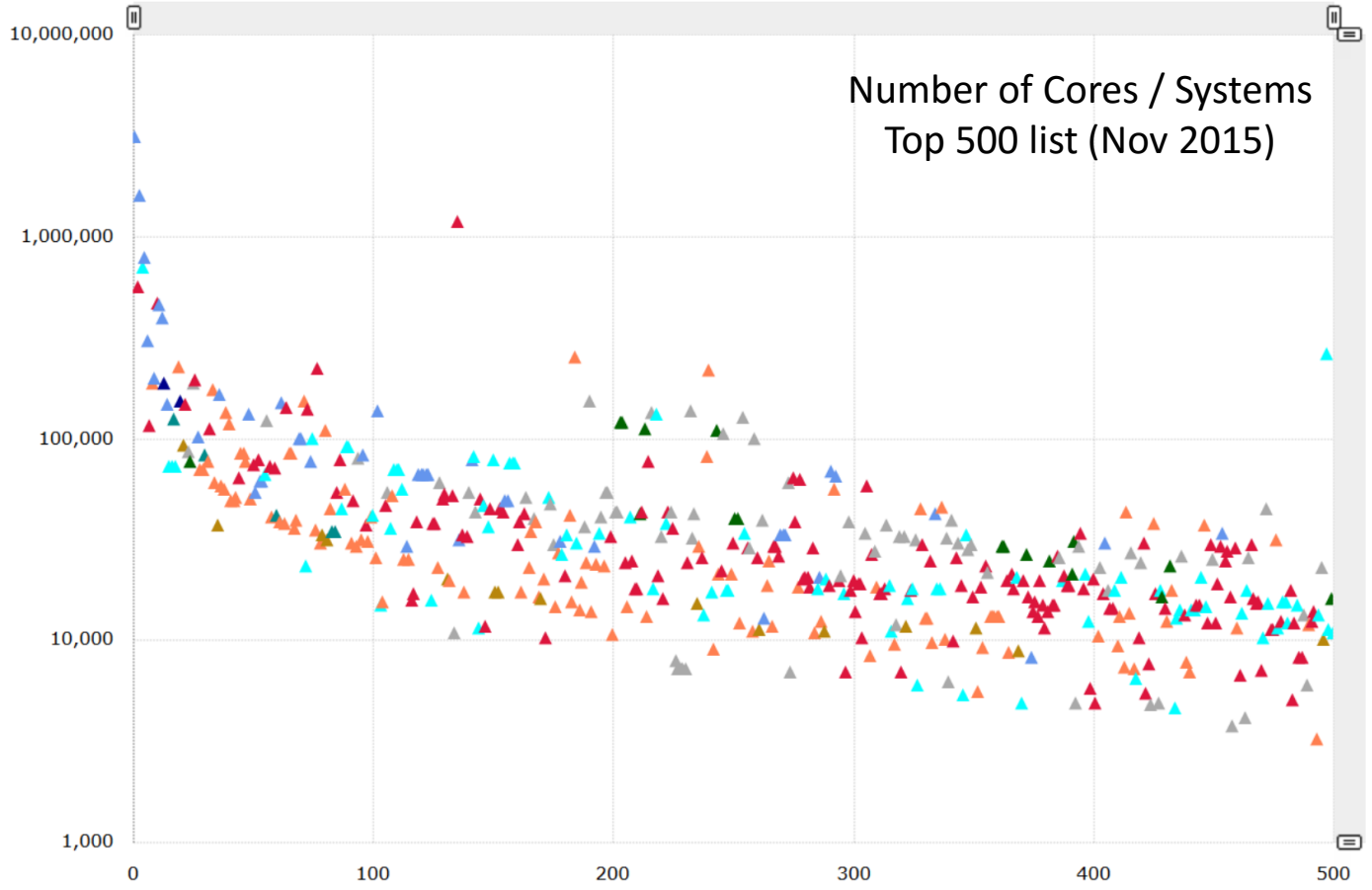
仕事が多すぎて一台のコンピュータでは時間がかかりすぎ…



複数のコンピュータで仕事を分担して時間を短縮しよう！

スパコンの並列度

- すでに10,000コアを超えるシステムが大多数
  - 最大では3,120,000基のコアを搭載 (China, Tianhe-2)

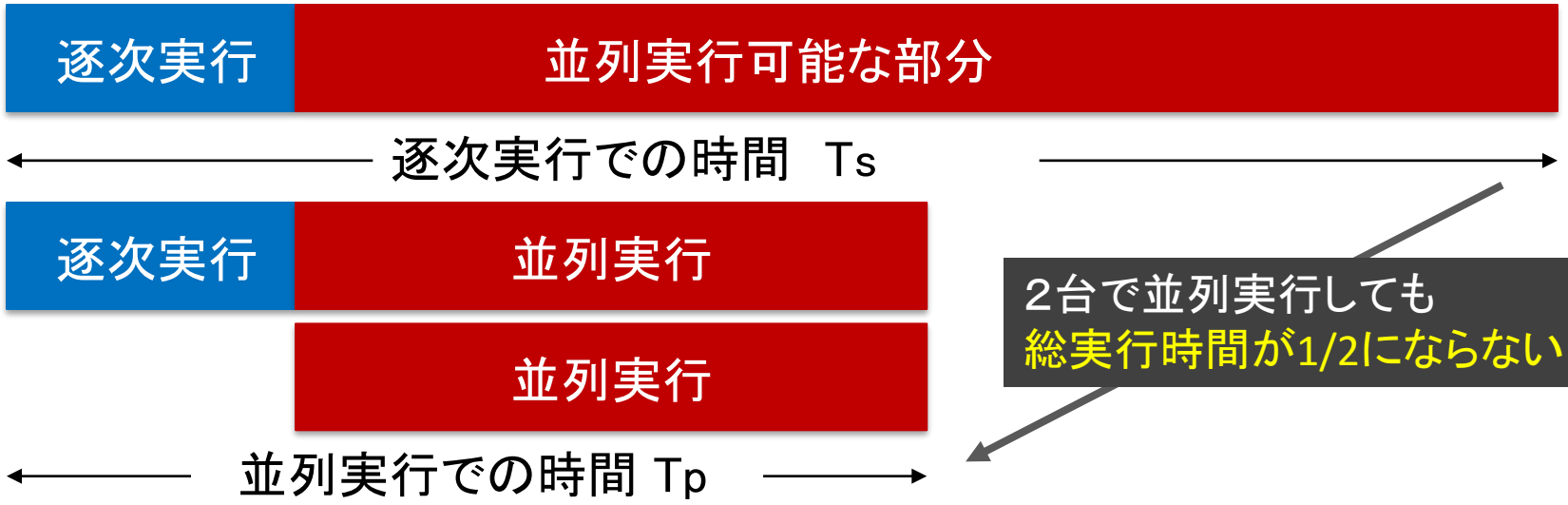


アムダールの法則

- まじめに性能を考えれば「並べるだけ」ではダメなことがわかる
  - プログラムには並列化できる部分とできない部分がある
    - 並列化率 α ：並列実行可能な部分の割合

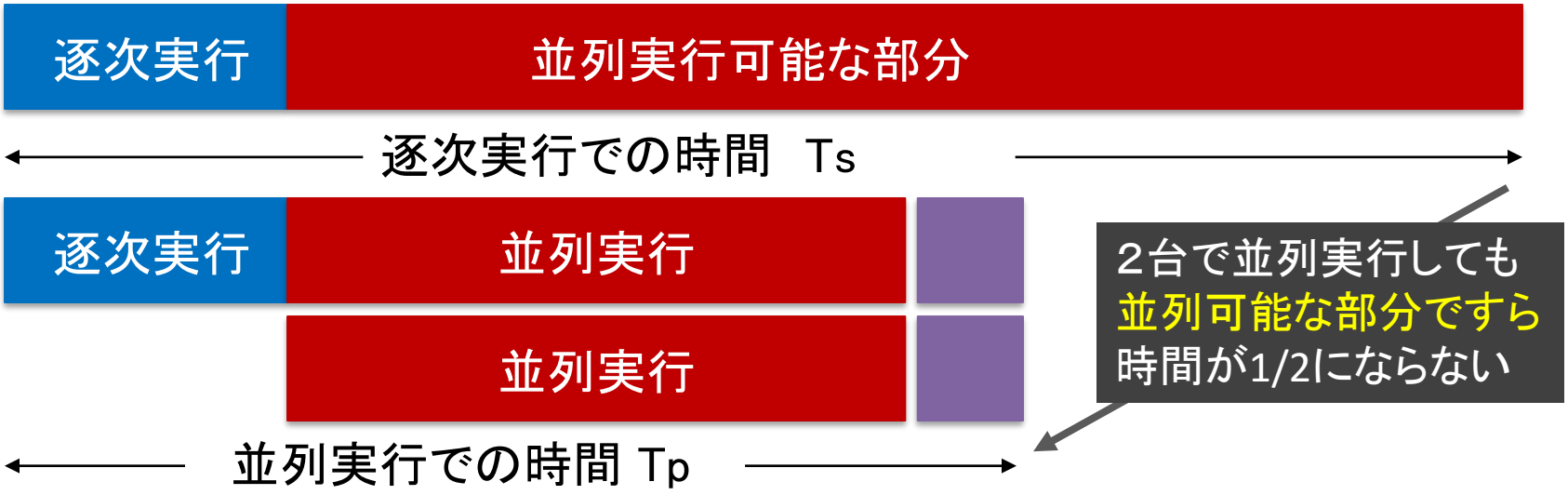
$$\text{性能向上比} = \frac{Ts}{Tp} = \frac{1}{1-\alpha+\alpha/n}$$

n → ∞のとき
性能向上比は1/(1-α)に漸近



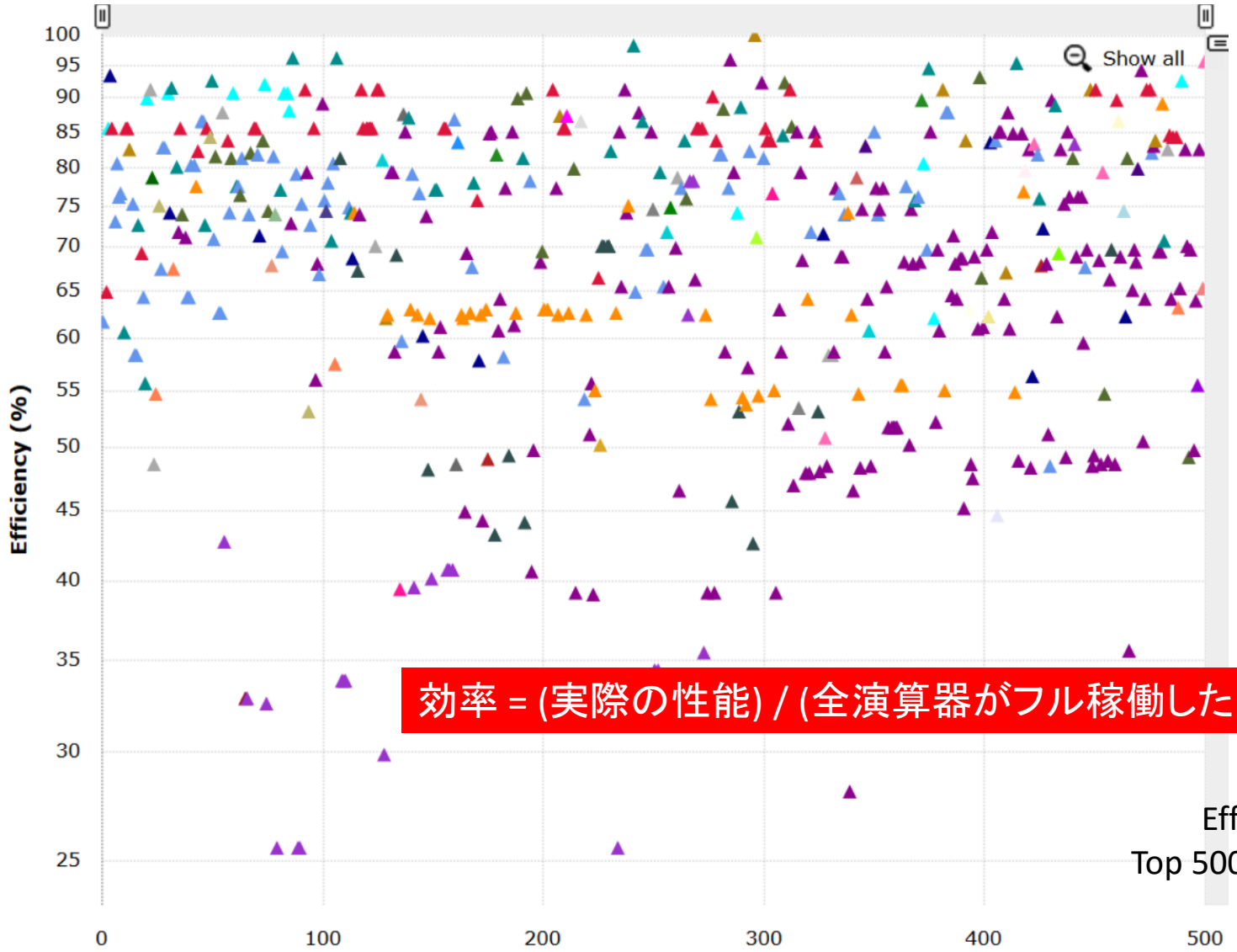
並列化のオーバヘッド

- 並列化自体にもさまざまなオーバヘッドがある
  - データ通信・同期
  - スレッドの生成・削除
  - 冗長計算　　…などなど



システムの大規模化に伴ってオーバヘッドも増加する傾向

HPLは特殊なベンチマーク



効率 = (実際の性能) / (全演算器がフル稼働した時の性能)

Efficiency (%)
Top 500 list (Nov 2015)

実際の用途での性能は？

TOHOKU UNIVERSITY

November 2015 HPCG Results

実用途に近い条件での性能を測る新ベンチマーク

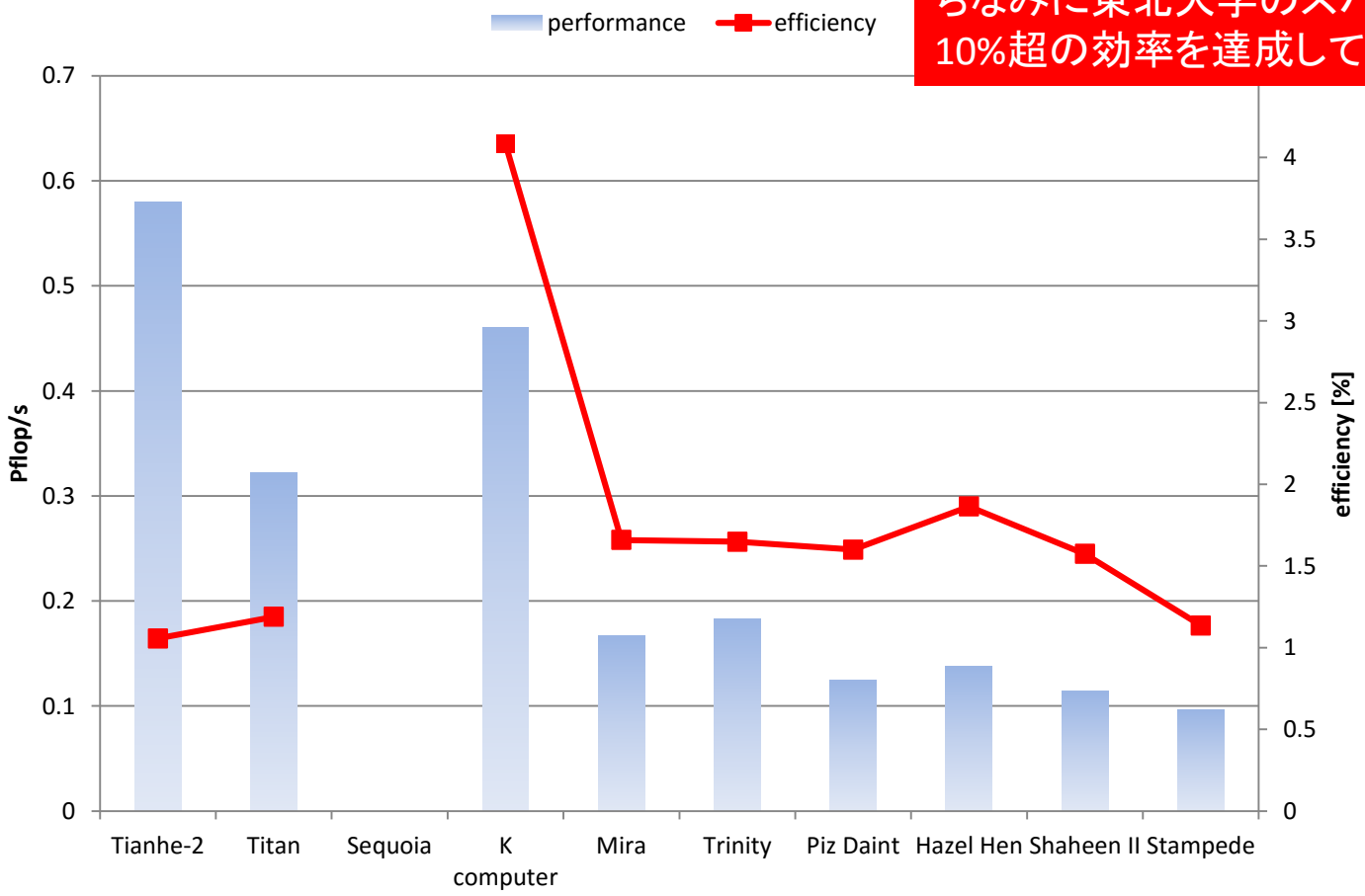
November 2015 HPCG Results

Rank	Site	Computer	Cores	HPL Rmax [Pflop/s]	TOP500 Rank	HPCG [Pflop/s]
1	NSCC / Guangzhou	Tianhe-2 NUDT, Xeon 12C 2.2GHz + Intel Xeon Phi 57C + Custom	3120000	33.863	1	0.5800
2	RIKEN Advanced Institute for Computational Science	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	705024	10.510	4	0.4608
3	DOE/SC/Oak Ridge Nat Lab	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	560640	17.590	2	0.3223
4	DOE/NNSA/LANL/SNL	Trinity - Cray XC40, Intel E5-2698v3, Aries custom	301056	8.101	6	0.1826
5	DOE/SC/Argonne National Laboratory	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom	786432	8.587	5	0.1670
6	HLRS/University of Stuttgart	Hazel Hen - Cray XC40, Intel	185088	5.640	8	0.1380

実際の用途での性能は？

- HPCGにおける効率はわずか数%

ちなみに東北大学のスパコン(SX-ACE)は10%超の効率を達成しています！



並べたくても並べられない

- 京コンピュータ
  - 演算能力：約10PFLOPS
  - 消費電力：約12MW
- 100倍の演算能力(エクサ)がほしい
  - 京を100個並べよう！⇒無理
    - 12MW × 100 = 1200MW = 1.2GW！

    → スパコンのためだけに原発を作るのは非現実的・・・

スパコン利用技術の重要性

- **電力効率＆並列度 (+耐障害性 etc)の向上**
  - アムダールの法則
  - 電力・設置面積

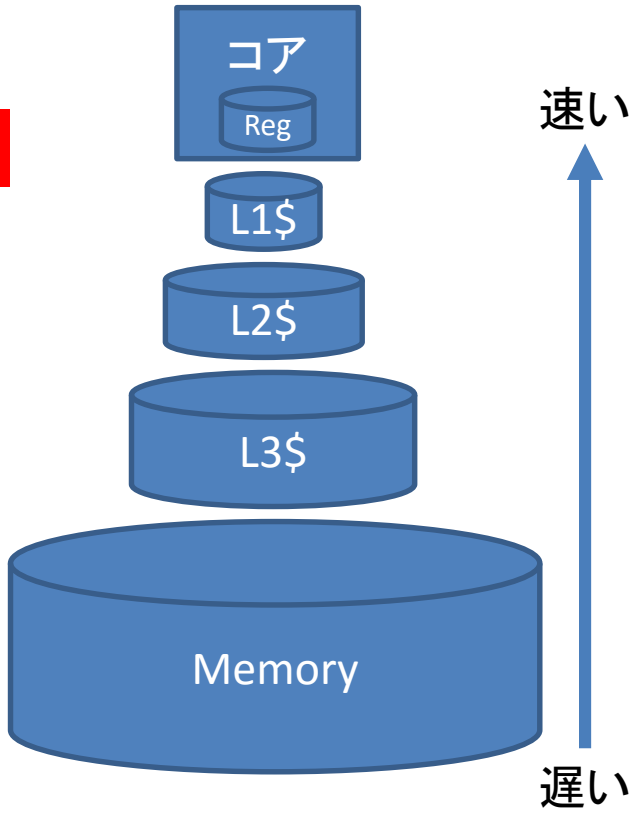
  →システムの大規模化・複雑化が不可避

  = **カタログ性能**だけがよくて、**実際の性能**は低いスパコン？

→ 複雑なスパコンを使いこなす技術がますます重要に！

なぜ難しいのか？

- **局所化と並列化の両立の難しさ**
  - 局所化
    - 計算やデータをなるべく一か所に集中
  - 並列化
    - 計算やデータをなるべく均等に分散
- **コンパイラを介した指示**
  - 機械語での指示はほぼ不可能
  - コンパイラの挙動を予想する必要
    - コンパイラが解釈しやすいコードに修正
    - 期待通り動かないコンパイラの肩代わり



性能を引き出すための労力

- **正しく動くプログラム≠正しく速く動くプログラム**
  - 正しく動くプログラムを作るだけでも十分大変だが、**正しく速く動くプログラムを書くためにはさらなる労力が必要**
  - コンパイラによる最適化に期待したいが、現実には職人プログラマによる最適化に性能面で大きく劣る(ことがある)

- **例）東北大学サイバーサイエンスセンター**
  - 長年の高速化支援実績

    このようなサービスを提供できる機関は多くない

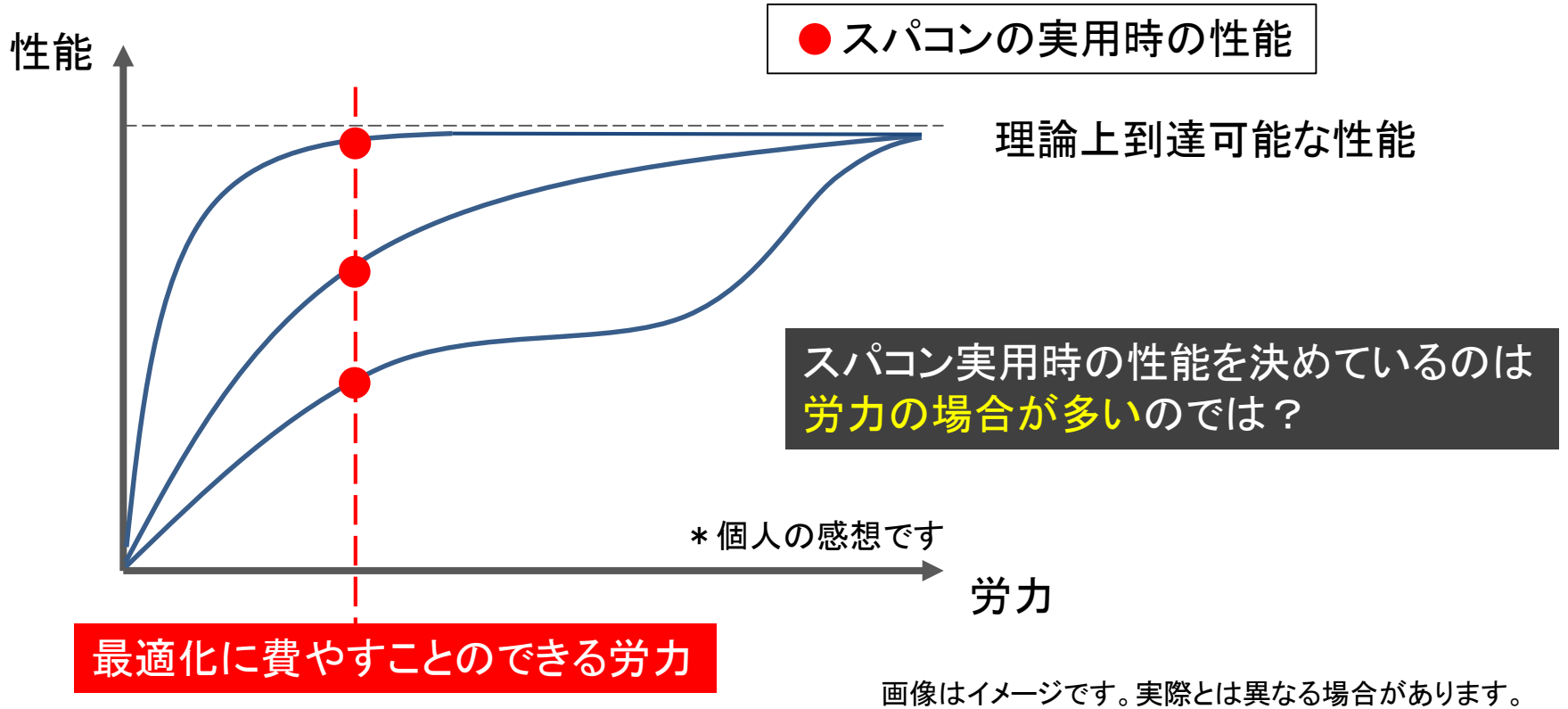
年	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
件数	2	8	8	9	10	7	18	20	8	16	10	15	8	8	13	6	11	9
平均ベクトル化性能向上比	1.9	46.7	4.5	2.5	1.6	2.2	6.7	2.9	1.3	2.9	33	9.4	381	47	16.2	19.7	16.7	10.3
平均並列化性能向上比	11.1	18.4	31.7	8.6	4.9	2.8	18.6	4.5	5.3	8.1	1.9	5.1	3.6	48	17.2	15.3	12.9	7.9

TOHOKU
UNIVERSITY

我々の取り組みのご紹介

性能と労力の関係

- スパコンが複雑化する＝勾配が小さくなる
  → 労力に何らかの前提をおいた時の適正性能が低下する



画像はイメージです。実際とは異なる場合があります。

「私のプログラム、性能低すぎ？」を解決するためには最適化の労力の低減が効果的

性能最適化のためのコード修正

- **Bad News**
  - システム特有/アプリ特有の最適化→ケースバイケースでの対応
  - 修正箇所はアプリコード全体に散在→保守性の著しい低下
- **Good News**
  - 特定のシステム/アプリに限定して考えれば、同じ (or 似たような) コード修正が何度も求められる傾向

人手によるコード修正を比較的少ない種類の機械的なコード変換で代行できる可能性

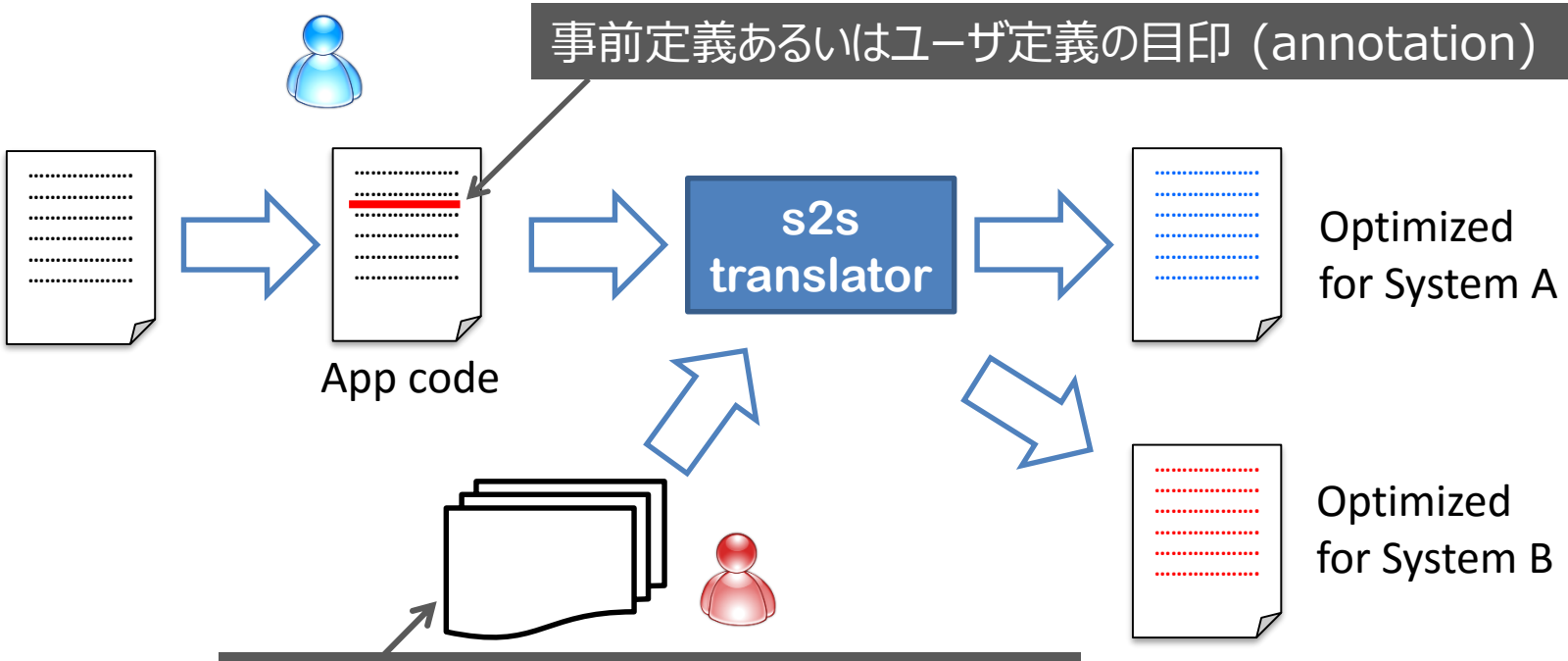
→ システム特有/アプリ特有の**コード修正**を機械的な**コード変換**として表現

＝手作業を減らすことで労力削減、知識や経験の蓄積・共有・再利用

Xevolver フレームワーク

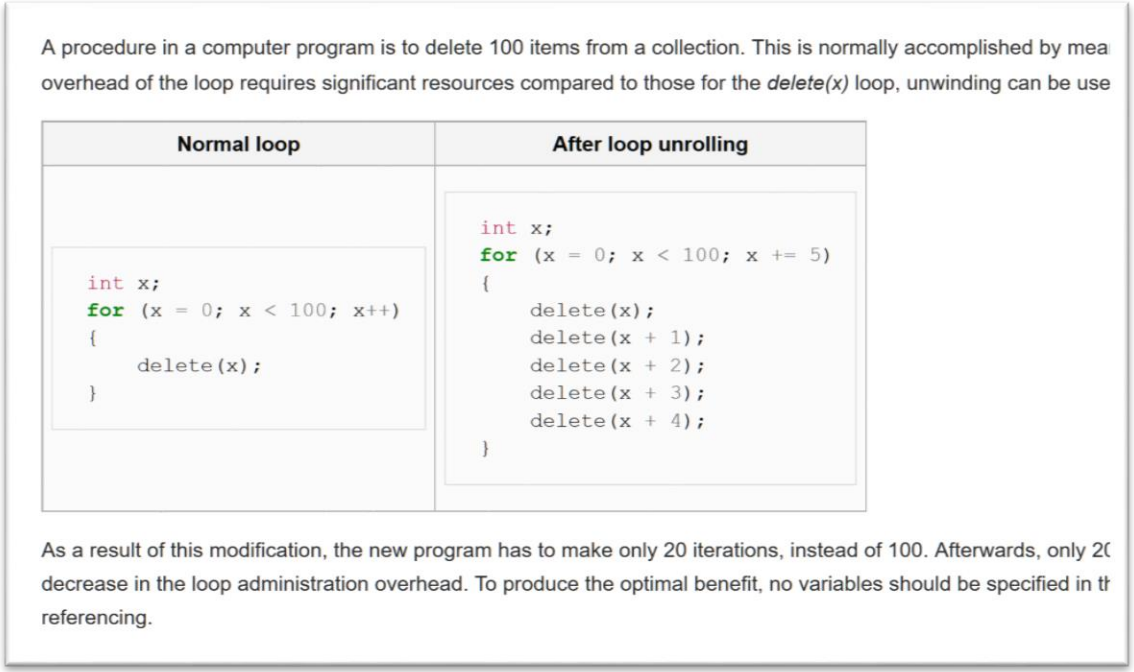
ケースバイケースのコード修正を代行するためには多種多様なコード変換が必要
＝いくつかの変換ルールを事前に準備して組合せるだけでは対応不可

→ Xevolver：ユーザ定義コード変換のためのフレームワーク



「コード変換」の表現方法

- **変換前後のコードの提示**
  - **e.g. Loop unrolling の説明(en.wikipedia.org)**



A procedure in a computer program is to delete 100 items from a collection. This is normally accomplished by mea
overhead of the loop requires significant resources compared to those for the *delete(x)* loop, unwinding can be use

Normal loop	After loop unrolling

`int x;`
`for (x = 0; x < 100; x++)`
`{`
`  delete(x);`
`}`
	`int x;`
	`for (x = 0; x < 100; x += 5)`
	`{`
	`  delete(x);`
	`  delete(x + 1);`
	`  delete(x + 2);`
	`  delete(x + 3);`
	`  delete(x + 4);`
	`}`

As a result of this modification, the new program has to make only 20 iterations, instead of 100. Afterwards, only 2(
decrease in the loop administration overhead. To produce the optimal benefit, no variables should be specified in th
referencing.

特別な知識 (e.g. XML and AST) がなくてもユーザ定義のコード変換を表現可能

ルールの一般化

- **ループの中に具体的な文を記述 = 一般性のないルール**
  - 変換前のコードを具体的に書いたルールでは、入力コードが変換前のコードと正確に一致する必要

before

```
int x;
for (x=0;x<100;x++) {
  delete (x);
}
```

after

```
int x;
for (x=0;x<100;x+=3) {
  delete (x);
  delete (x+1);
  delete (x+2);
}
```

- **一般化されたルール**
  - 任意の文を表現する特別な変数の利用

before

```
int x;
for (x=0;x<100;x++) {
  $STMT;
}
```

after

```
int x;
for (x=0;x<100;x++) {
  $STMT; x++;
  $STMT; x++;
  $STMT;
}
```

変換ルールの自動生成

須田(東京大)


```
program loop_reversal

!$xev tgen var(i_, i0_, i1_) exp
!$xev tgen list(l_) stmt

!$xev tgen src begin
!$xev loop rev
  do i_ = i0_, i1_
     !$xev tgen  stmt(l_)
  end do
!$xev tgen src end

!$xev tgen dst begin
  do i_ = i1_, i0_, -1
     $!xev tgen stmt(l_)
  end do
!$xev tgen dst end

end program loop_reversal
```

Tgen variables that match any Fortran variables

A list variable that matches any number of statements

Directive used as a mark for transformation

The code pattern before transformation

Special directive that matches arbitrary statement(s)

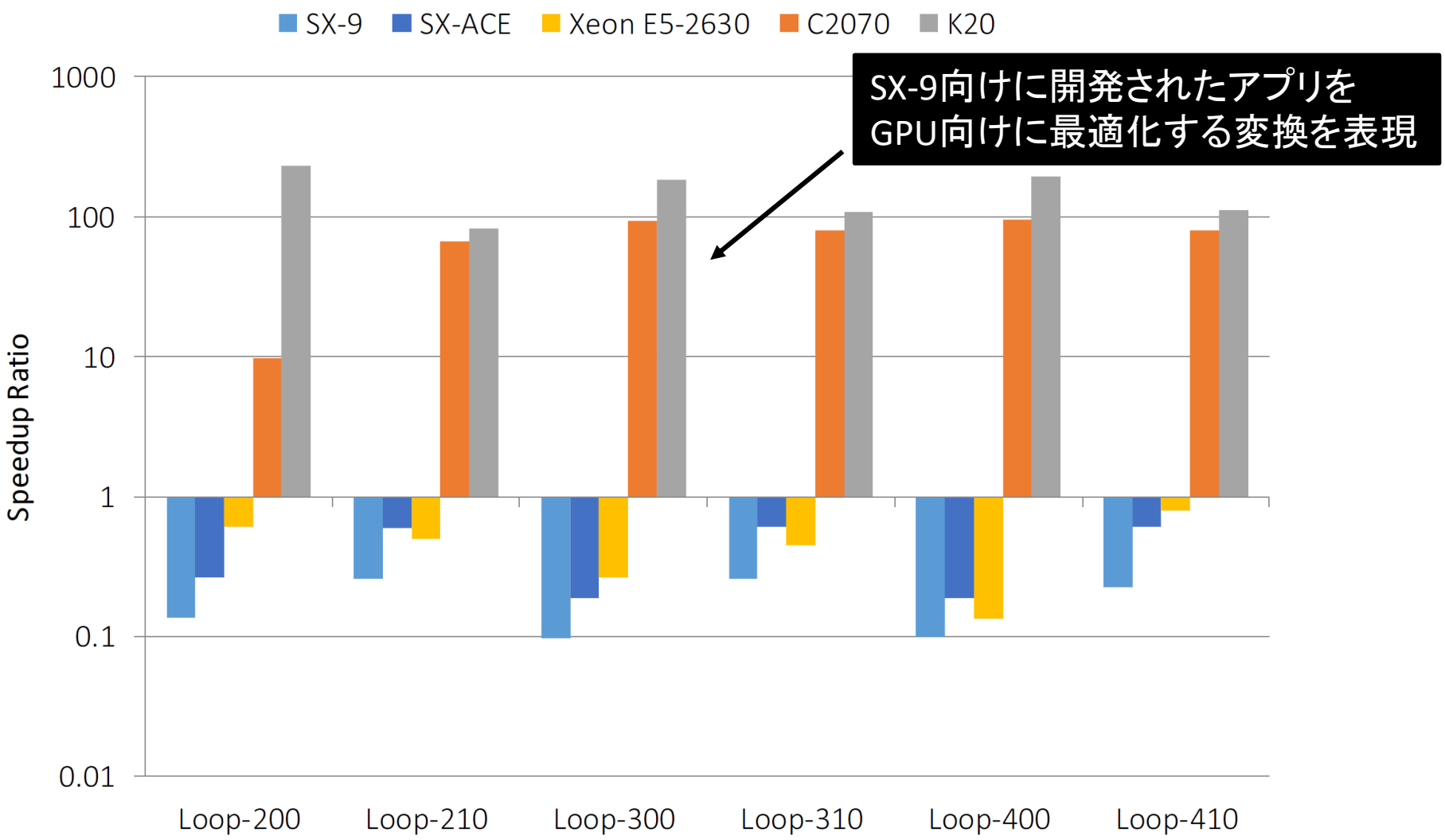
Loop is reversed

The code pattern after transformation

The loop body is copied to the dst code.

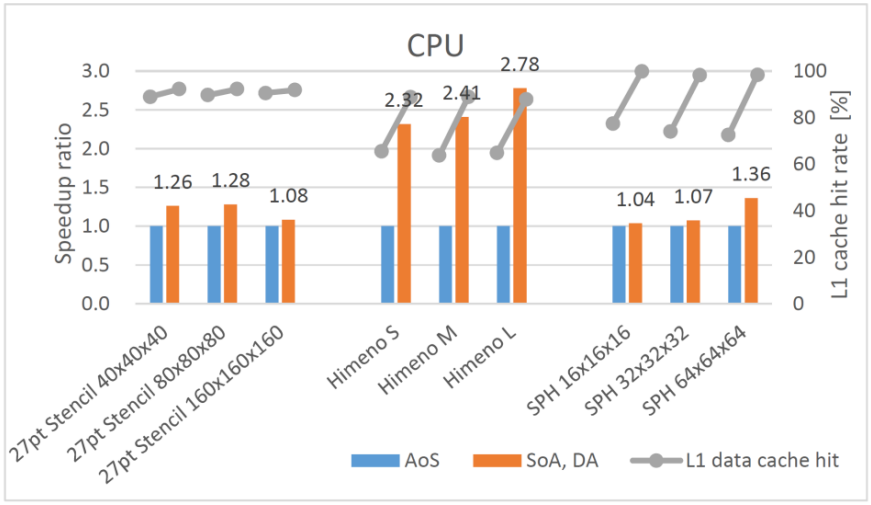
須田(東京大)

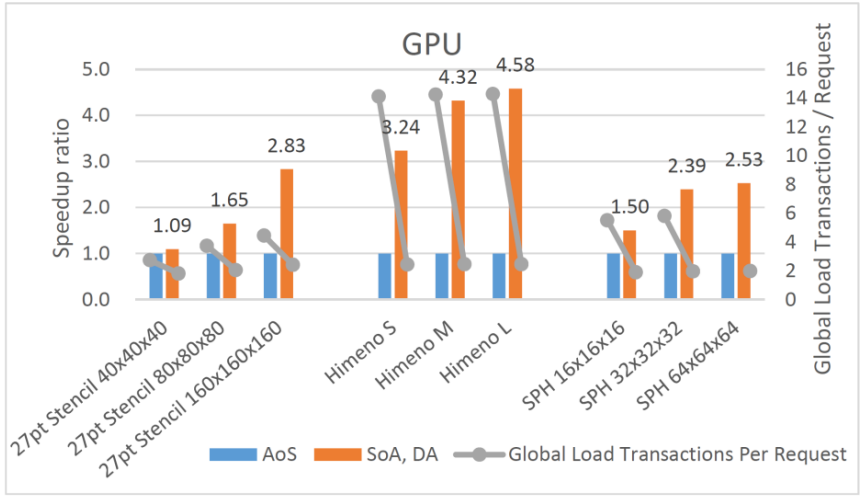
ユーザ定義変換による最適化の効果



山田ら(東北大)

データレイアウト最適化への適用



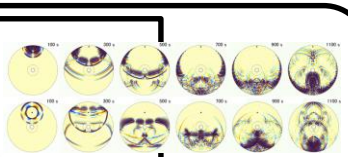


- **データレイアウト最適化によってCPUもGPUも性能向上**
  - GPUの性能はデータレイアウトにより敏感
  - キャッシュ容量よりもデータサイズが大きい場合にはCPUの性能向上にも寄与
  - 変換ルールをカスタマイズすることにより、他のアプリにも流用可能となる可能性

江川ら(東北大)

知識と経験の蓄積と共有

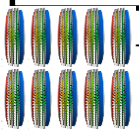
HPC codes



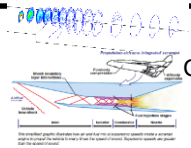
code A | code B | code C | ... | code N

a seismic wave Propagation
X 109

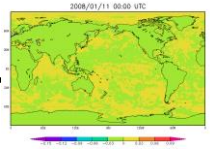
optimizing & parallelizing & porting



Turbine
x11.91



Scramjet Combustion
x20.6

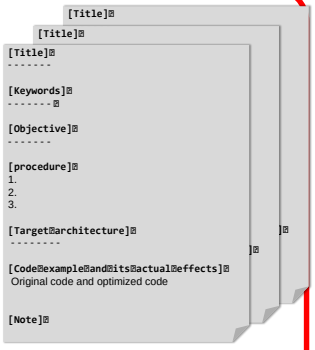


Global Barotropic Ocean Modeling
x8.31

Analyze

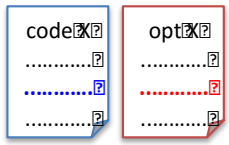
Accumulating Examples

- Code Characteristics
- Aim of optimization
- Plat home
- Process of Optimization
- Effects of optimizations
- Executable codes (Kernels)

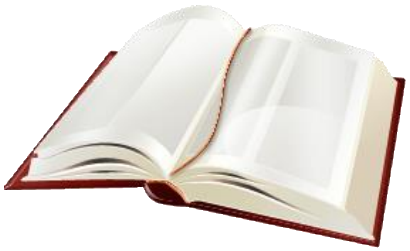


Evaluate

・performance evaluation using several HPC systems



Clarifying the conditions to keep a high performance portability



Refactoring Catalog

TOHOKU UNIVERSITY

~~スパコンが切り開く未来~~

スパコンを使って切り開いてほしい未来

人工知能？

- **強いAIと弱いAI (Searle, 1980)**
  - 強いAI：意識や知能を持つ機械
  - 弱いAI：画像認識や推論などを行う機械
- **人間と区別がつかないAIの実現？**
  - Eugene Goostman (2014)
    - Turing Test 2014で史上初めてチューリングテストをパス
    - 人物像設定が成功のカギ？
  - Microsoft Tay (2016)
    - 不適切発言を学習してしまって公開中止に
  - Microsoft 人工知能女子高生「りんな」(2015)

スパコンでひとつだけ夢をかなえるなら

- **強い人工知能には夢がある！**
  - 人間の頭脳で行われている処理方式を真似する？
    - 人工ニューラルネットワーク
      - 実は学生のころ(ちょっとだけ)携わっていました
      - 頭脳の処理方式の大半は未解明だから強いAIの実現は遠い
  - 人間の頭脳も所詮は物理現象
    - 脳内で起こっているすべての物理現象をシミュレーションできれば強い人工知能も実現できるかも・・・

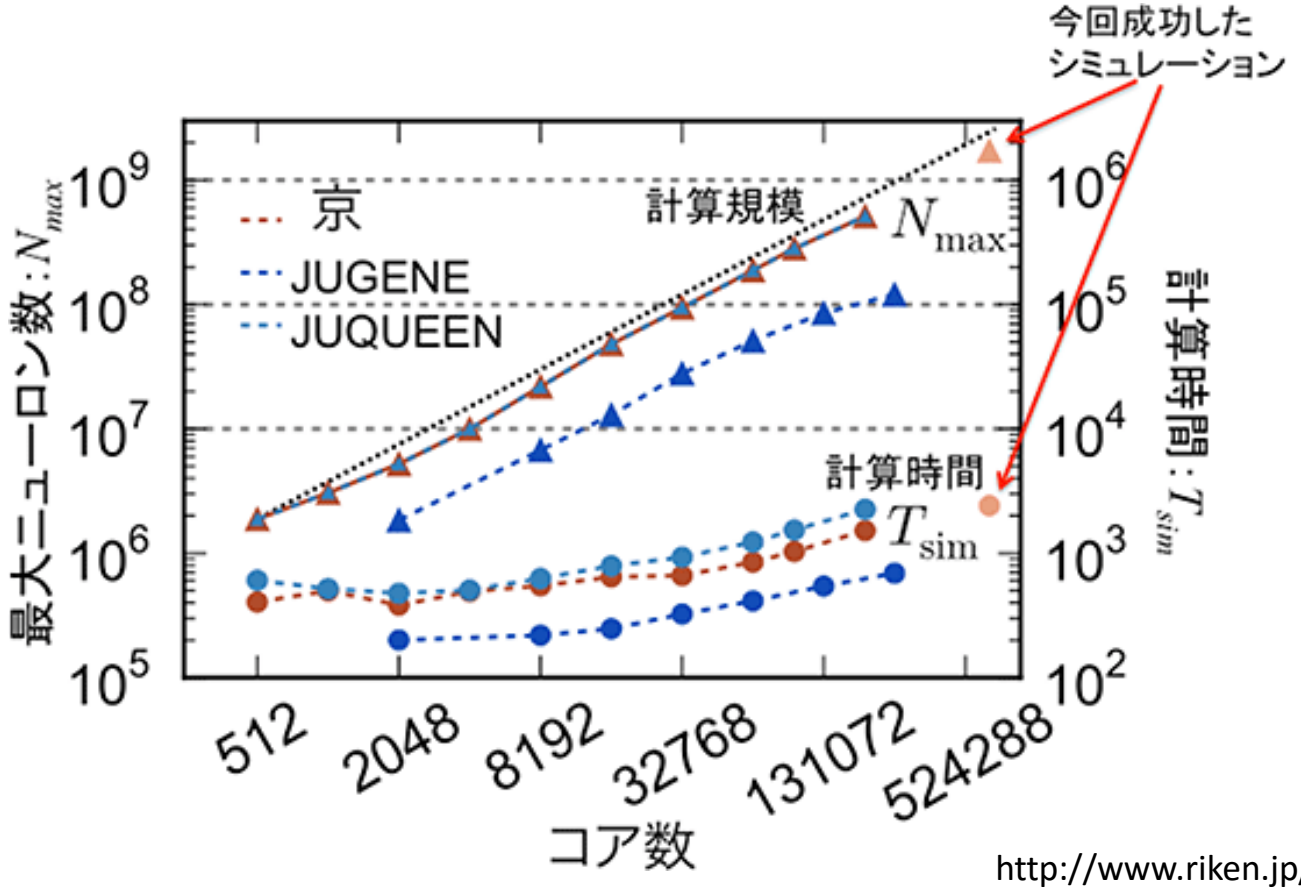
⇒ ものすご～く高性能なコンピュータが必要

2023年ごろに10^{18}FLOPS (Exaflop/s)あればできる!

-- Henry Markram@ISC2012

ヒトの全脳のシミュレーションは遠い

- 「京」を使い10兆個の結合の神経回路のシミュレーションに成功 (2013年8月2日)



シミュレーション
脳細胞　: 17億個
シナプス: 10兆個

ヒトの全脳
脳細胞　: 1000億個
シナプス: 100兆個

http://www.riken.jp/pr/topics/2013/20130802_2/

脳のシミュレーションに必要な計算量

項目	ヒトの全脳（理論値）	カイコガ全脳（理論値）	カイコガ嗅覚・運動系（実測値）
細胞数	10^{11}	10^{6}	10^{4}
必要計算量（FLOPS）	10^{21}	10^{16}	10^{15}

計算科学研究ロードマップ白書
http://open-supercomputer.org/wp-content/uploads/2012/03/science-roadmap.pdf

$$10^{21} = \overset{\text{ゼッタ}}{1},\overset{\text{エクサ}}{000},\overset{\text{ペタ}}{000},\overset{\text{テラ}}{000},\overset{\text{ギガ}}{000},\overset{\text{メガ}}{000},\overset{\text{キロ}}{000}$$

$$10^{16} = 1,0000,0000,0000,0000$$

京　兆　億　万　一

FLOPS（フロップス, Floating-point Operations Per Second）
コンピュータの性能をあらわす. 1秒間に浮動小数点数演算が何回できるかを表す値.

まとめ

- **「適正性能」を考えることは重要**
  - 現代のコンピュータは得手不得手が極端
  - 情報科学の研究成果が真に実用的であるためには性能に無関心ではいられない

    → 適正性能を意識しよう！
    - 性能を測る
    - 測った性能を鵜呑みしない
    - 性能を決めている要因は何かを考える

「労力」の場合が多いのでは？
→ 我々の研究の取り組みの紹介

うわっ・・・私のプログラム、
性能低すぎ・・・？ ← これを取り除けたら一歩前進