



シミュレーションが 未来をひらく

スーパーコンピュータと アプリケーションの性能

2014年4月

独立行政法人理化学研究所
計算科学研究機構 運用技術部門
ソフトウェア技術チーム チームヘッド

南 一生

minami_kaz@riken.jp



RIKEN ADVANCED INSTITUTE FOR COMPUTATIONAL SCIENCE

講義の概要

- スーパーコンピュータとアプリケーションの性能
- アプリケーションの性能最適化1 (高並列性能最適化)
- アプリケーションの性能最適化2 (高並列性能最適化)
- アプリケーションの性能最適化の実例1
- アプリケーションの性能最適化の実例2

内容

- スーパーコンピュータとは？
- アプリケーションの性能とは？
- 高並列化のための重要点
- 単体性能向上のための重要点

スーパーコンピュータとは？

スーパーコンピュータとは・・・

- **スーパーコンピュータ=卓越した計算能力**

その時代の一般的なコンピュータよりも、極めて高速（浮動小数点演算性能で1000倍以上）な計算機

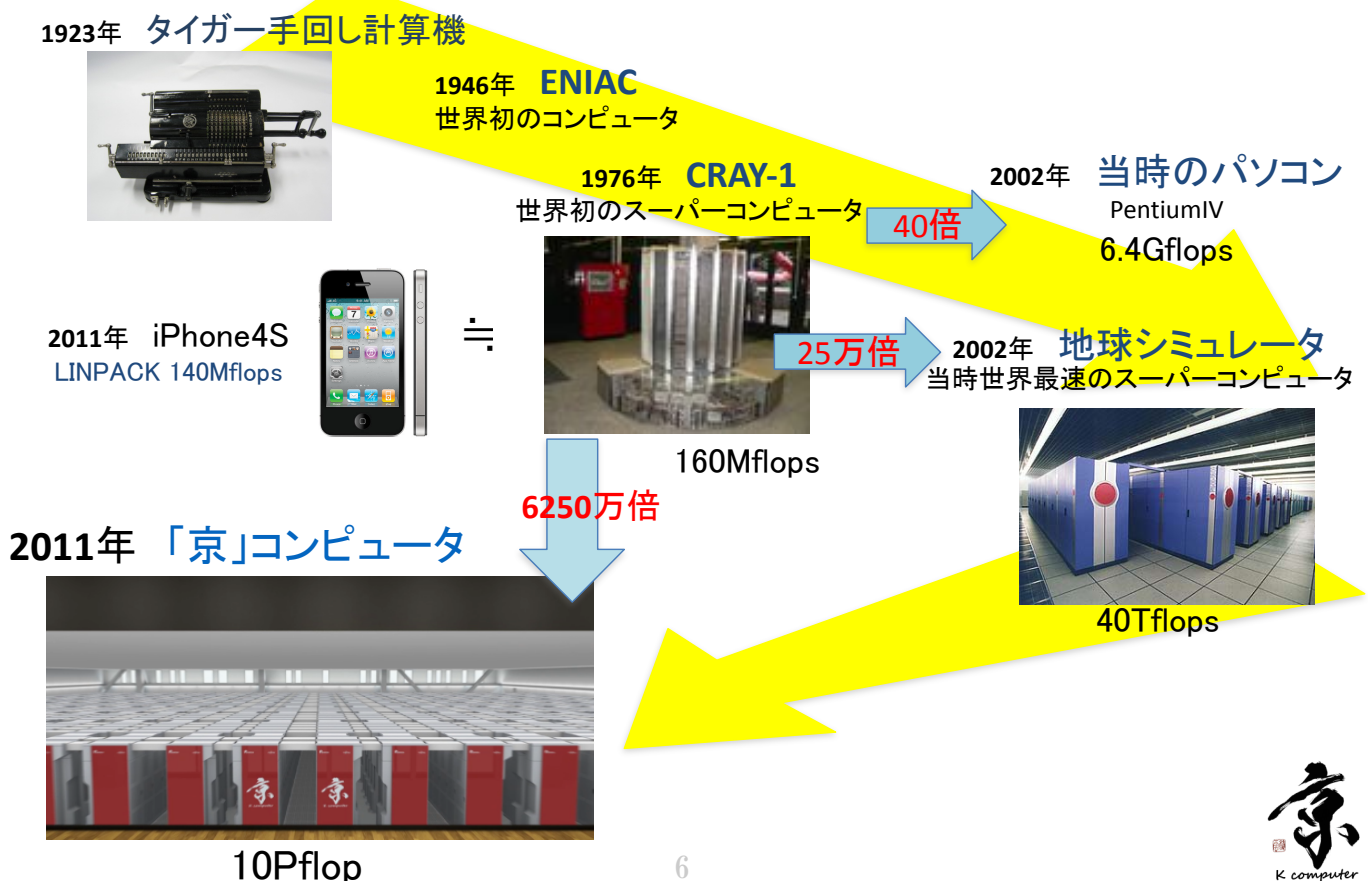
- **演算性能 1.5 TFlops 以上 (*)**

・・・現在の政府調達における「スーパーコンピュータ」の定義
(*) : 1秒間に1.5兆回以上の浮動小数点演算が可能



5

スーパーコンピュータの発展



6

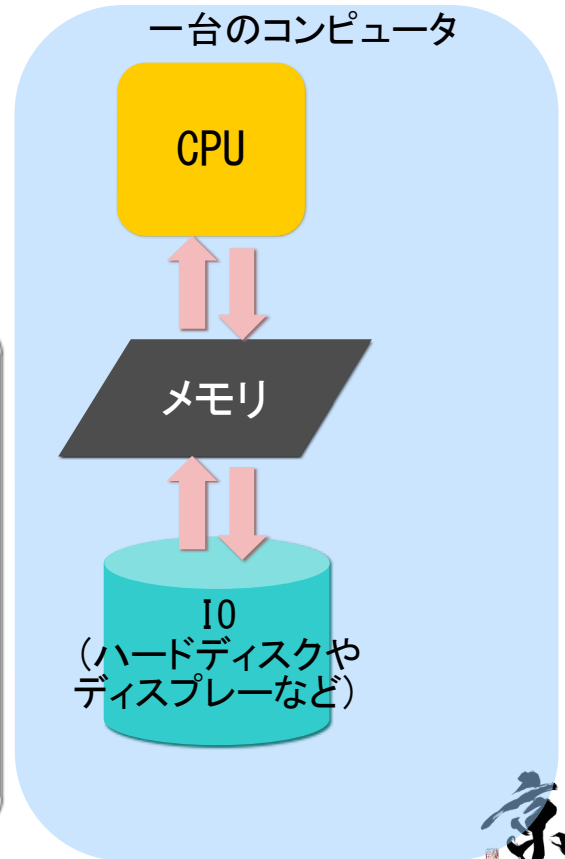


コンピュータとは？

ハードウェア ソフトウェア(進化とともに分化)

- CPU ■ アプリケーション
- メモリ ■ ミドルウェア
- IO (入出力) ■ OS

CPUの性能向上
≒コンピュータの性能向上
であった時代



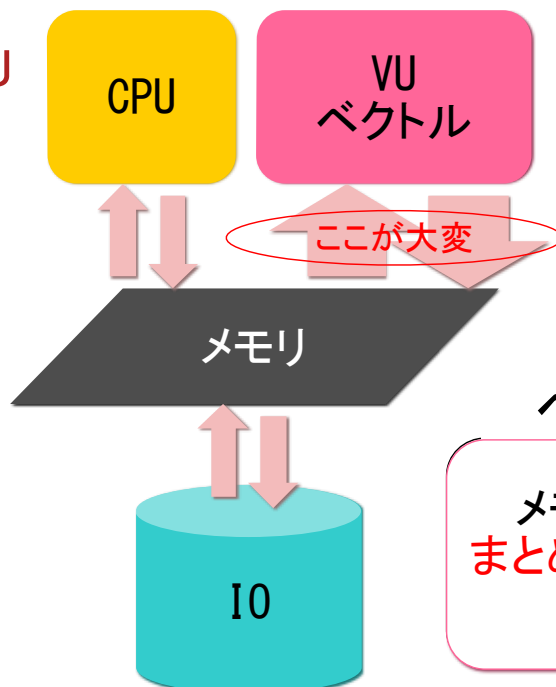
7



昔のスーパーコンピュータ

昔は、ベクトル機構などによって、1台のコンピュータを高速化

- CPU(SU)+VU
- メモリ
- IO(入出力)



一つのOSで制御
される一つの
コンピュータ

ベクトル機構

メモリからデータを
まとめて持ってくるのは
とても大変

今もこの方式が無くなったわけではありませんが、

8

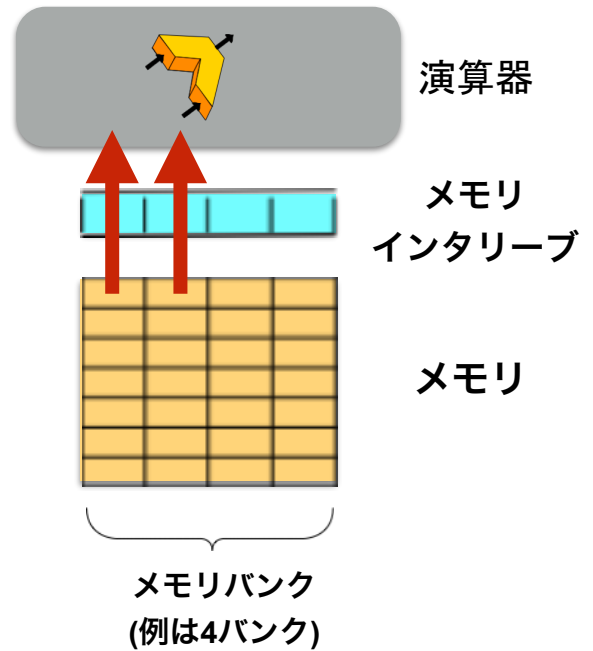


シングルプロセッサの問題

動作周波数が低い場合(昔)

メモリウォール問題

- メモリは一定時間を経過しないと同一メモリバンクにアクセスできない
- 昔は20サイクルくらい待っていた
- しかし動作周波数が低かったため演算器も遅く演算速度とメモリ転送性能は釣り合っていた

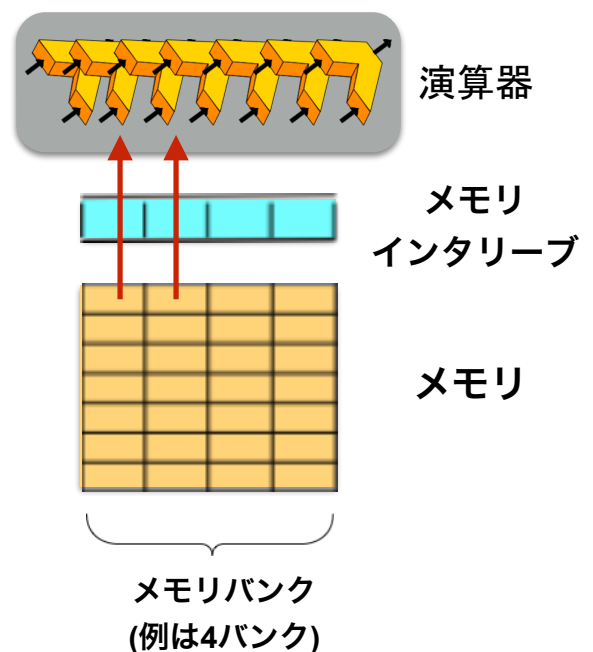


シングルプロセッサの問題

動作周波数が高い場合(現在)

メモリウォール問題

- メモリは動作周波数が高くなってくるともっと待つ事となる(100-200サイクル)
- メモリに比べて演算器は動作周波数が高くなると高速になった
- さらに半導体プロセスの微細化により演算器はCPUにたくさん搭載可能となった
- 結果的に演算器に比べメモリのデータ転送能力が低くなった

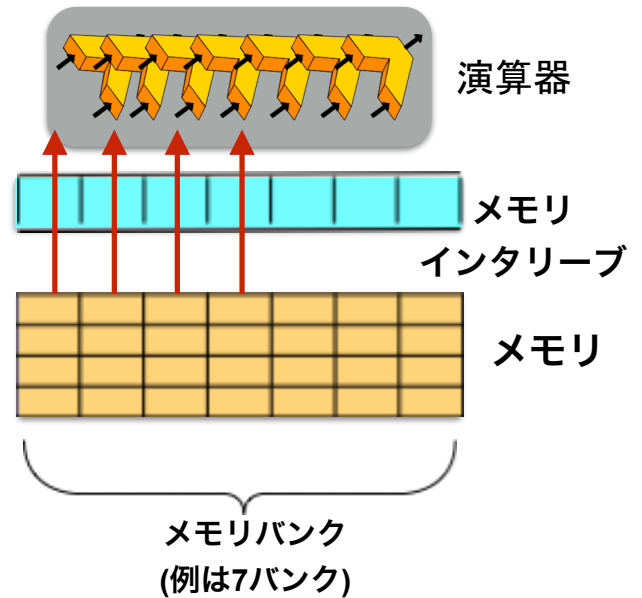


シングルプロセッサの問題

対策1

- メモリバンクを増やし演算器に供給するデータ量を増大させた
- ベクトル機では数百のバンクを搭載した
- しかしこの方式は機構が複雑になり電力が増大する
- またコスト・価格も高くなる

動作周波数が高い場合



11

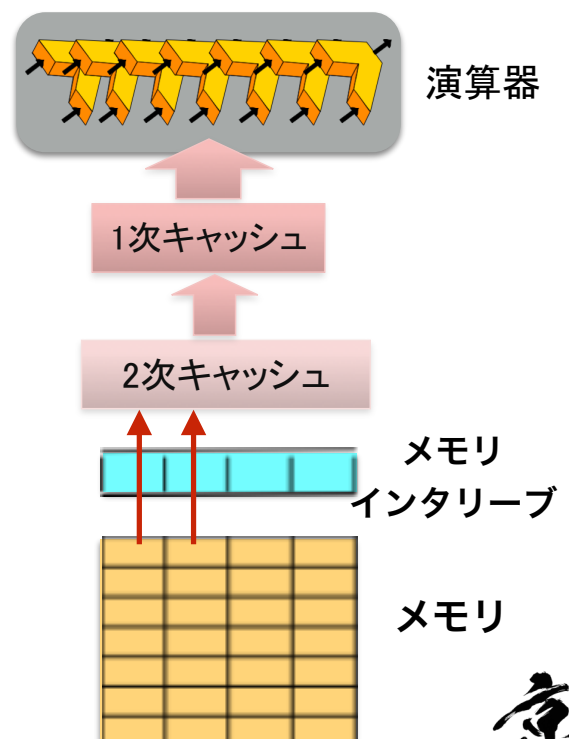


シングルプロセッサの問題

対策2

- メモリのバンクは増やさない
- データ供給能力の高いキャッシュをメモリと演算器間に設ける
- データをなるべくキャッシュに置きデータを再利用する事でメモリのデータ供給能力の不足を補う
- こうすることで演算器の能力を使い切る

動作周波数が高い場合



12



シングルプロセッサの問題

一台のコンピュータの処理限界

- 動作周波数を上げる事で電力が吹き上がる
- 動作周波数の限界を迎えている
- メモリウォール問題もあり一台のコンピュータの演算能力を上げててもメモリの能力が追いつかない

そこで！



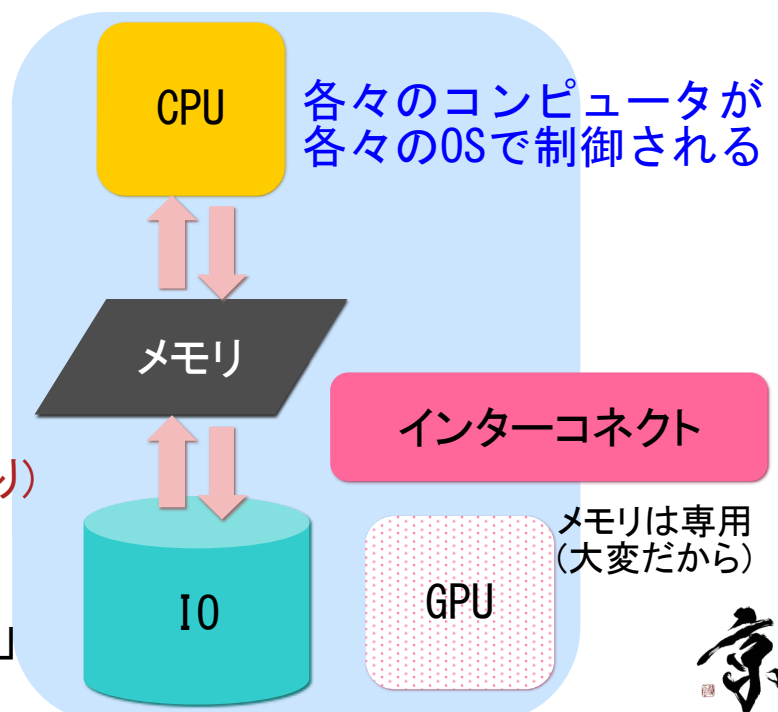
13

最近のスーパーコンピュータ

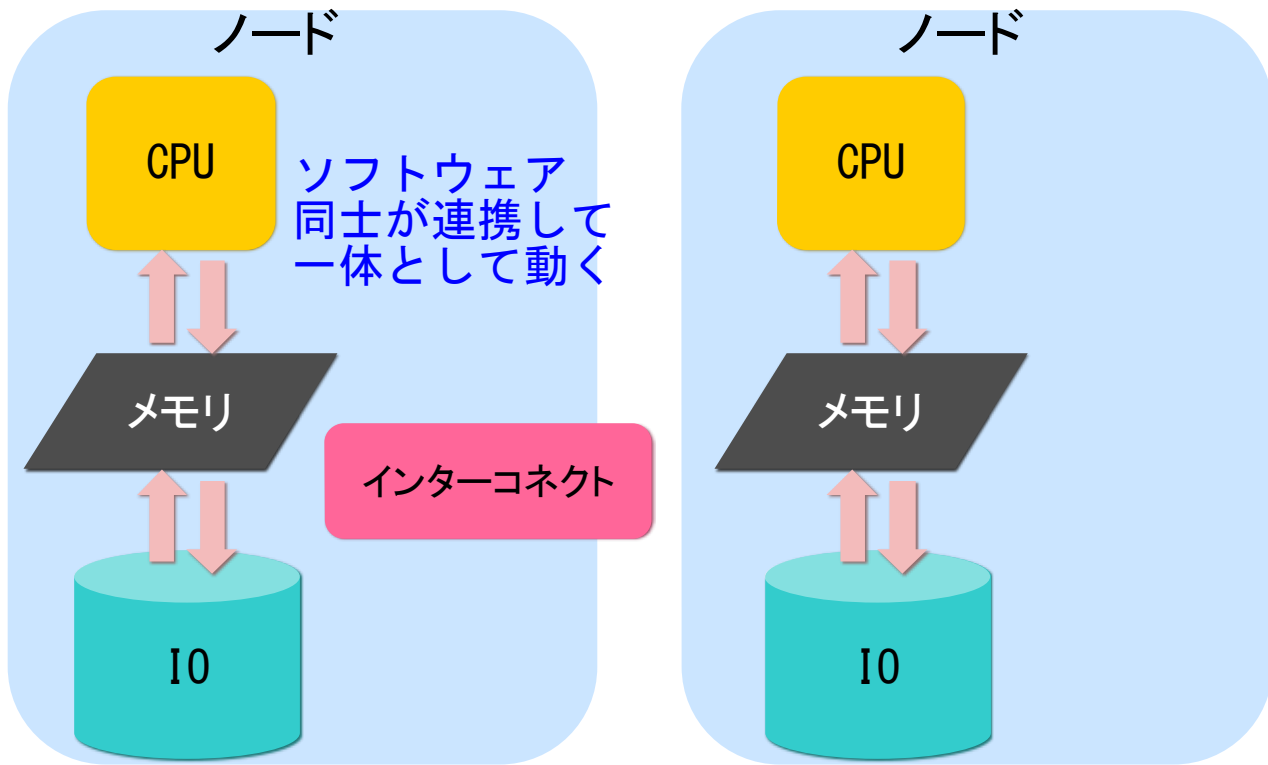
たくさんのコンピュータをつないで、システムとして**一体**で動かす

- CPU
- メモリ
- IO(入出力)
- インターコネク(接続機構)
- GPU/アクセラレータ
(加速器: あったり、なかったり)

各々のコンピュータ=「ノード」



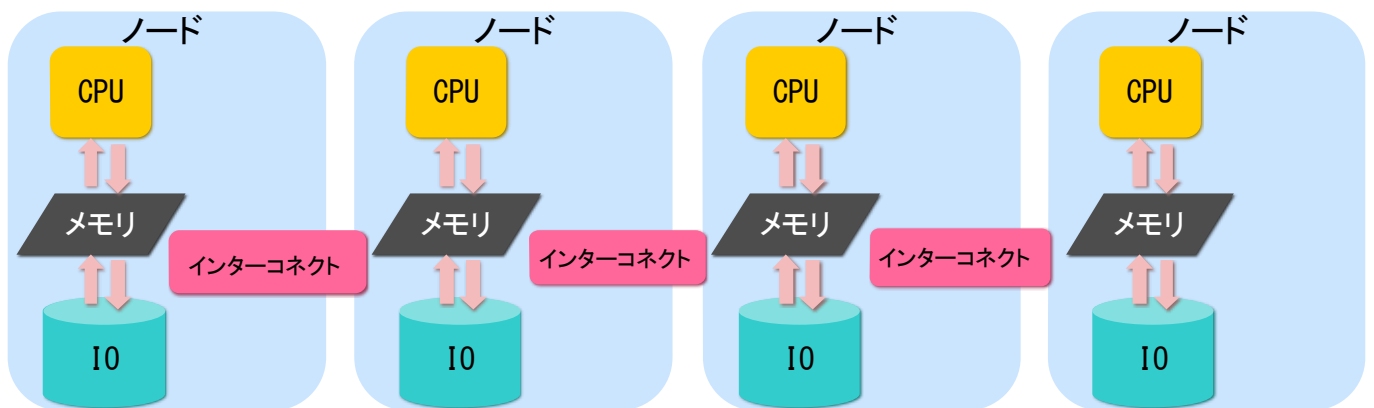
コンピュータ (ノード) どうしをつなぐ



15



コンピュータ (ノード) どうしをつなぐ



システムとして一体で動く

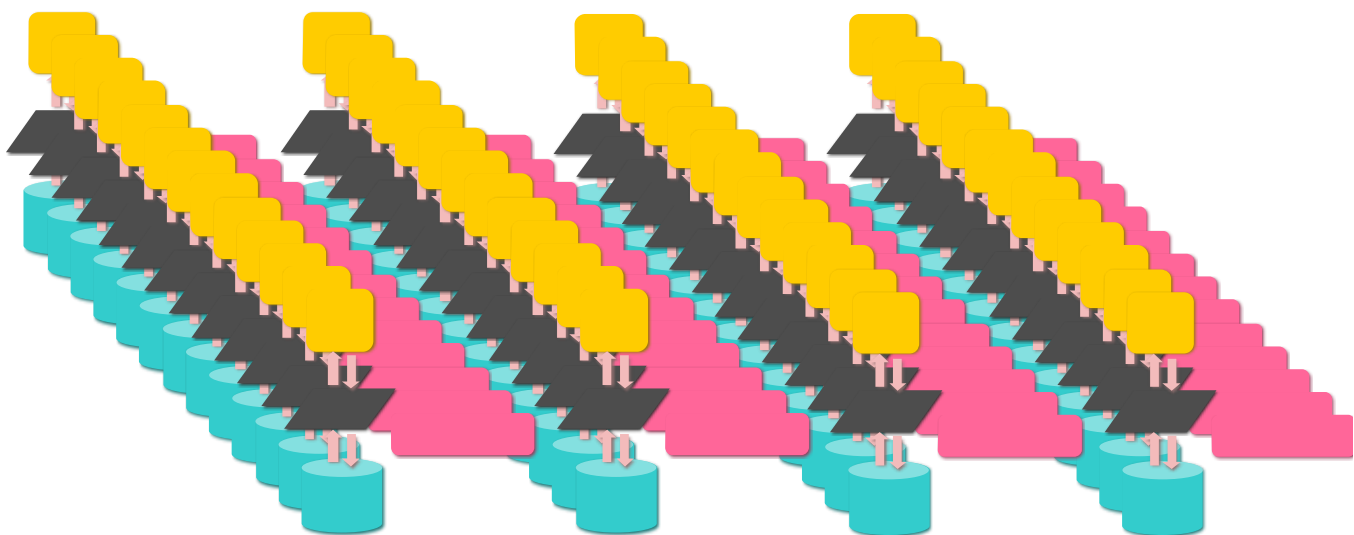
(一つの仕事をやる、たくさんの仕事は互いに連携して分担しあって片づける)

16



もともっと、たくさんのコンピュータ（ノード）どうしをつなぐ

どれだけたくさんつなげるか？⇒システムとして高性能
一つのシステムとして動く

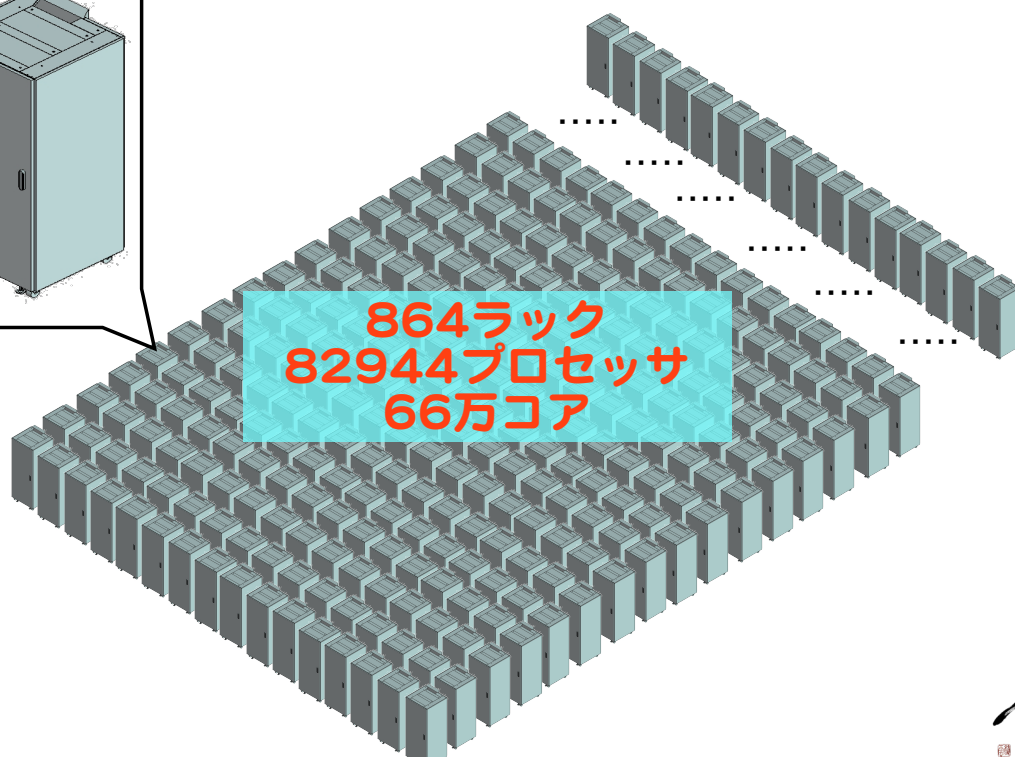
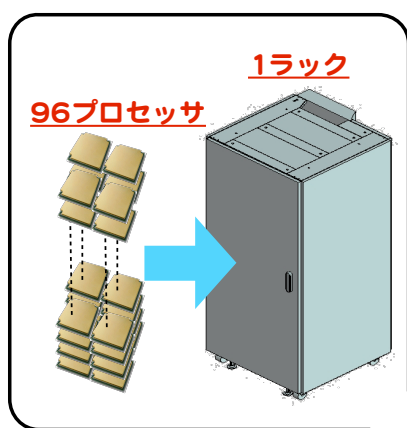


実際はハードディスクは、まとめる



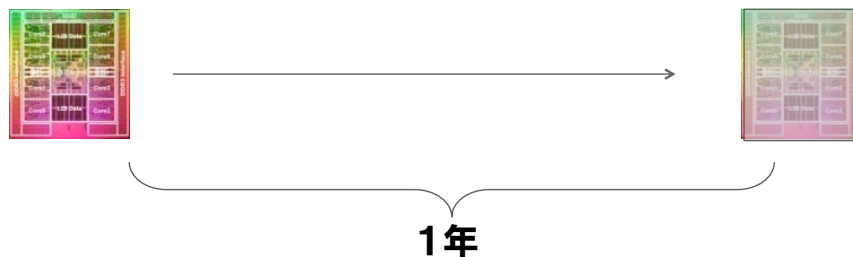
ちなみに「京」の場合は？

50m×60mの部屋に

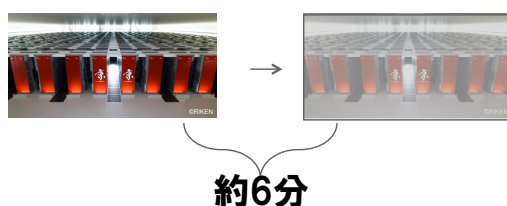


これだけ巨大システムでCPUが頻繁に故障すると・・・

- ✓ 仮にひとつのCPUが、1年に1回故障したとします



- ✓ システム全体で見ると、**約6分に1回**の故障・・・



CPUの故障率を抑えることは極めて重要

19



スーパーコンピュータについてまとめると

- 1940年代半ばのENIACの登場
- 最初はシングルプロセッサの時代
- ベクトル/CISC/RISC/スーパースカラ等色々なアーキテクチャが登場
- その時代は演算器を増やすことにより高速処理を実現.
- メモリウォール問題及び電力、動作周波数を上げられない問題によりシングルプロセッサの限界となった.
- それらによりスーパーコンピュータが並列プロセッサアーキテクチャへと変化

- スーパーコンピュータは一つのノードの構成としては普通のコンピュータと基本的には変わらないが..
- トータルとしての計算能力・演算性能がきわめて高い
- そのためには高速なインターコネクトが要求される
- またシステムトータルとしての高い省電力性能・高信頼性が高いレベルで要求される

20



アプリケーションの性能とは？

21



スパコンはシミュレーションに使う

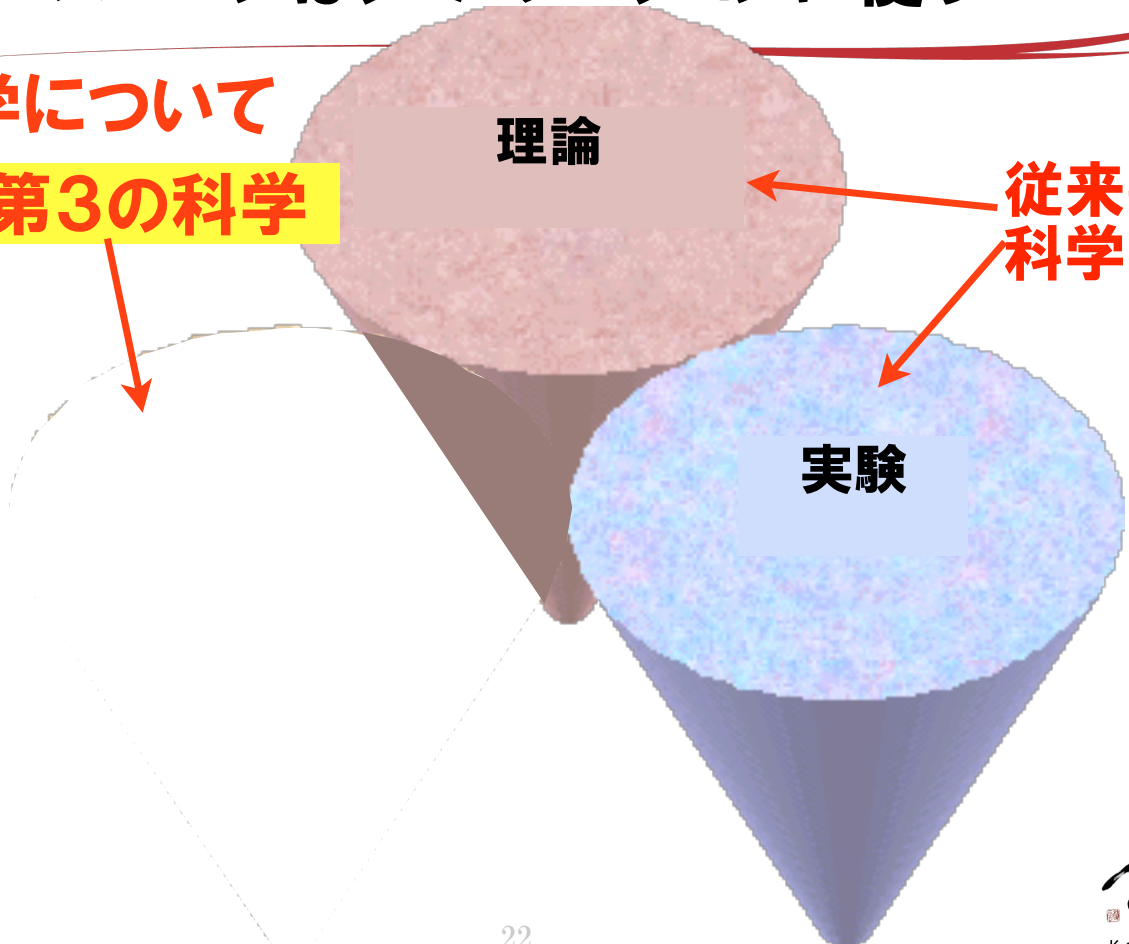
科学について

第3の科学

理論

従来の科学は

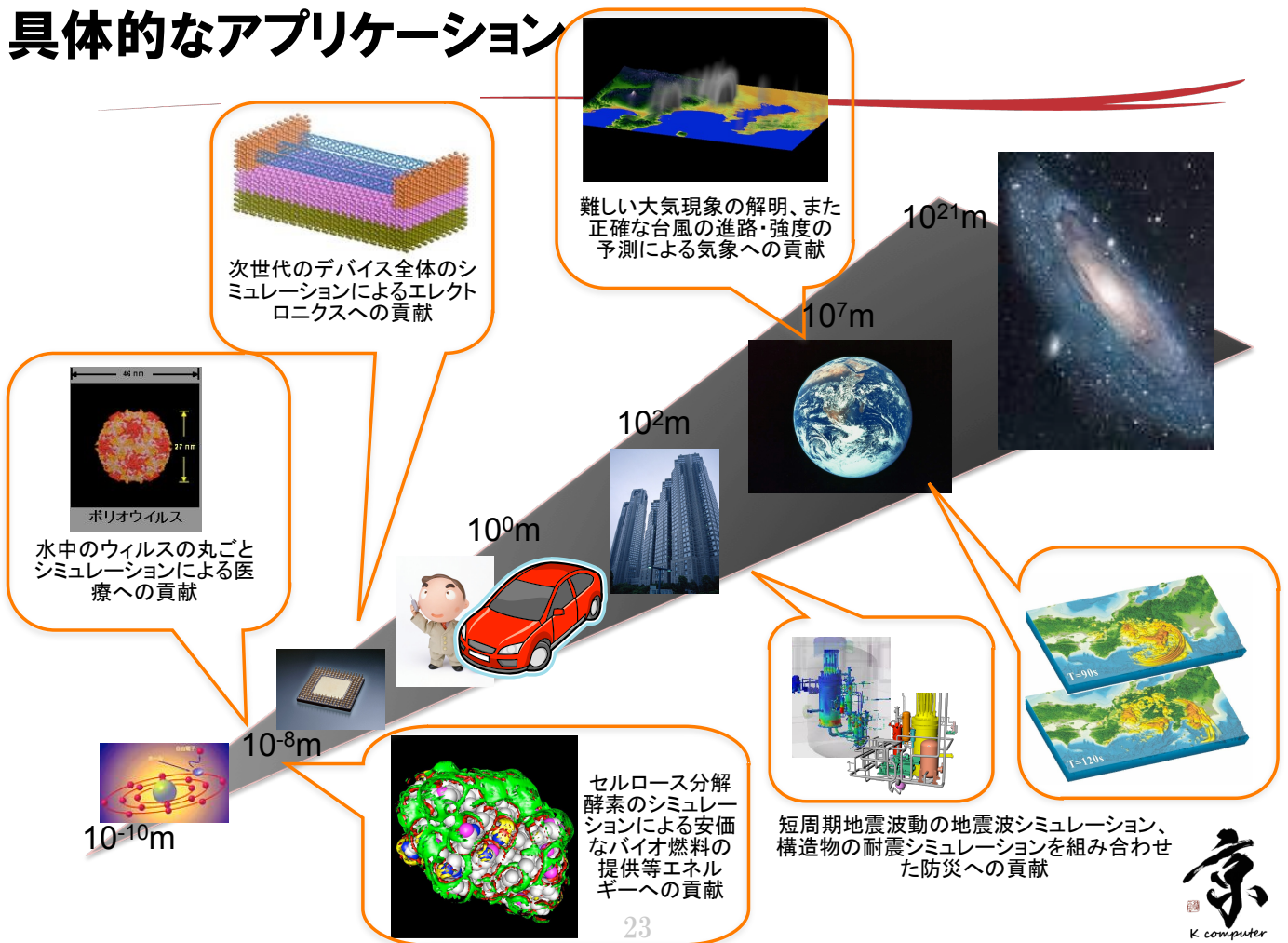
実験



22



具体的なアプリケーション



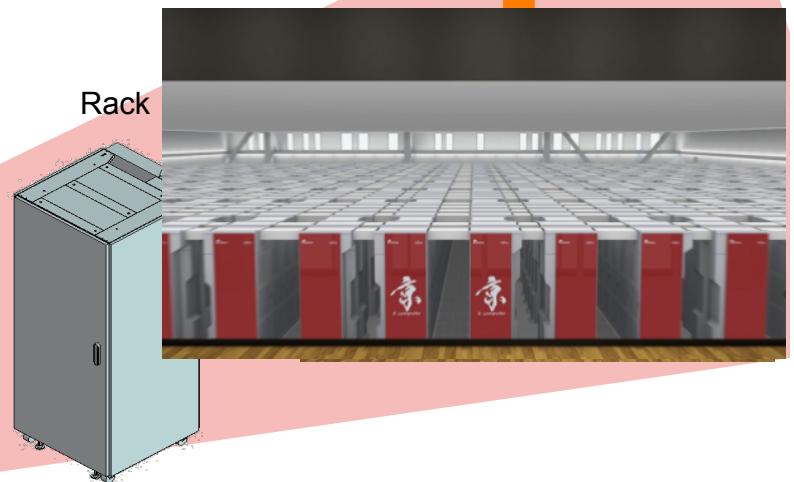
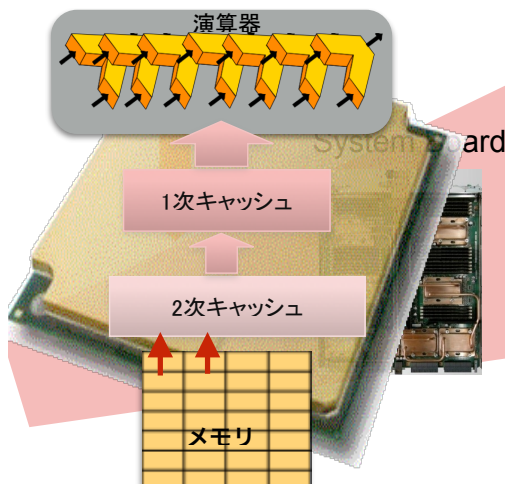
現代のスパコン利用の難しさ

アプリケーションの超並列性を引き出す

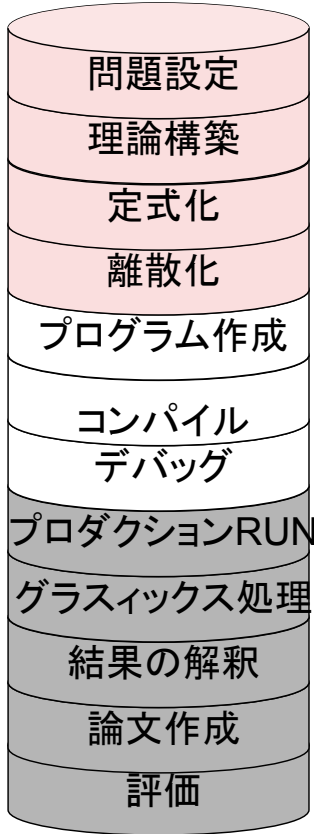
現代のスーパーコンピュータシステム

プロセッサの単体性能を引き出す

現代のプロセッサ



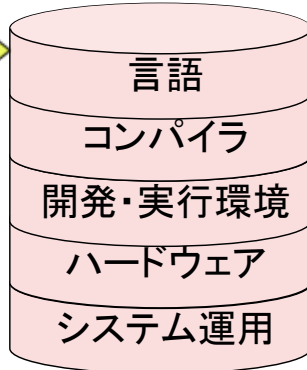
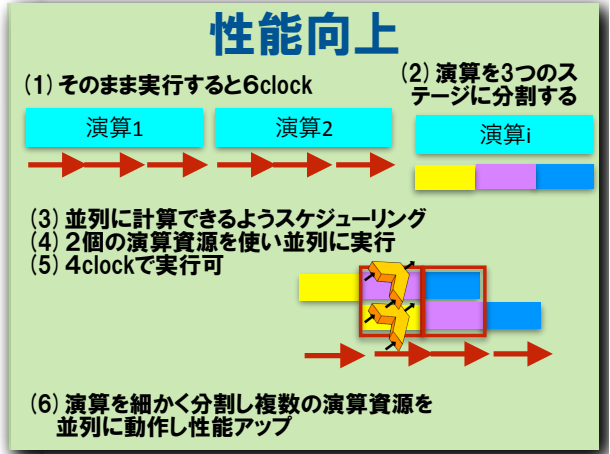
計算科学



理論に忠実な分かりやすいコーディング

プログラム

計算機科学



計算科学



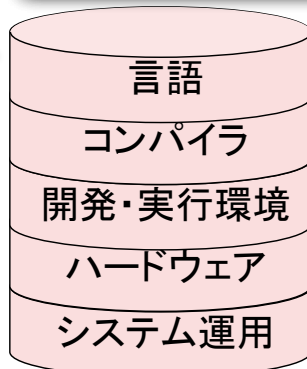
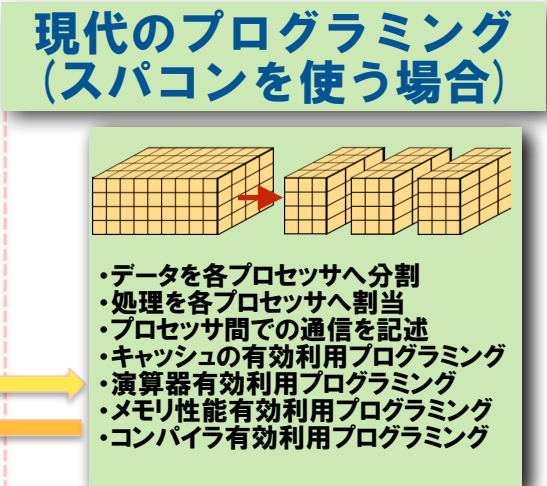
理論に忠実な分かりやすいコーディング

プログラム

プログラム

高並列化・高性能コーディング

計算機科学



アプリケーションの性能についてまとめると

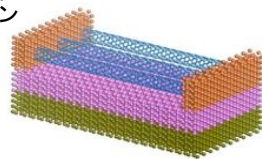
- プログラマーは、プロセス間の並列性を意識して並列化し、またプロセス毎のデータの分散を意識してプログラミングすることが必要となった。
- そうすることにより数千から数万に及ぶ**プロセス間の並列性**を最大限利用した超高速計算が実現可能となる
- また**シングルプロセッサの性能を最大限使いきる**プログラムのチューニングも必要となる
- もしシングルプロセッサの性能の1%しか出せなければ、並列化されていてもシステムトータルでも1%の性能

27

京
K computer

スーパーコンピュータの威力

例えば次世代のデバイス
全体のシミュレーション



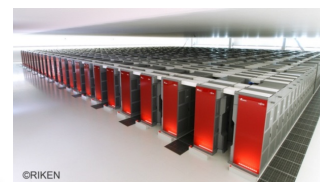
たくさんの原子を用いたシ
ミュレーション



パソコンで100年かかる計算



京を使って
73000倍
の速度で実行



©RIKEN

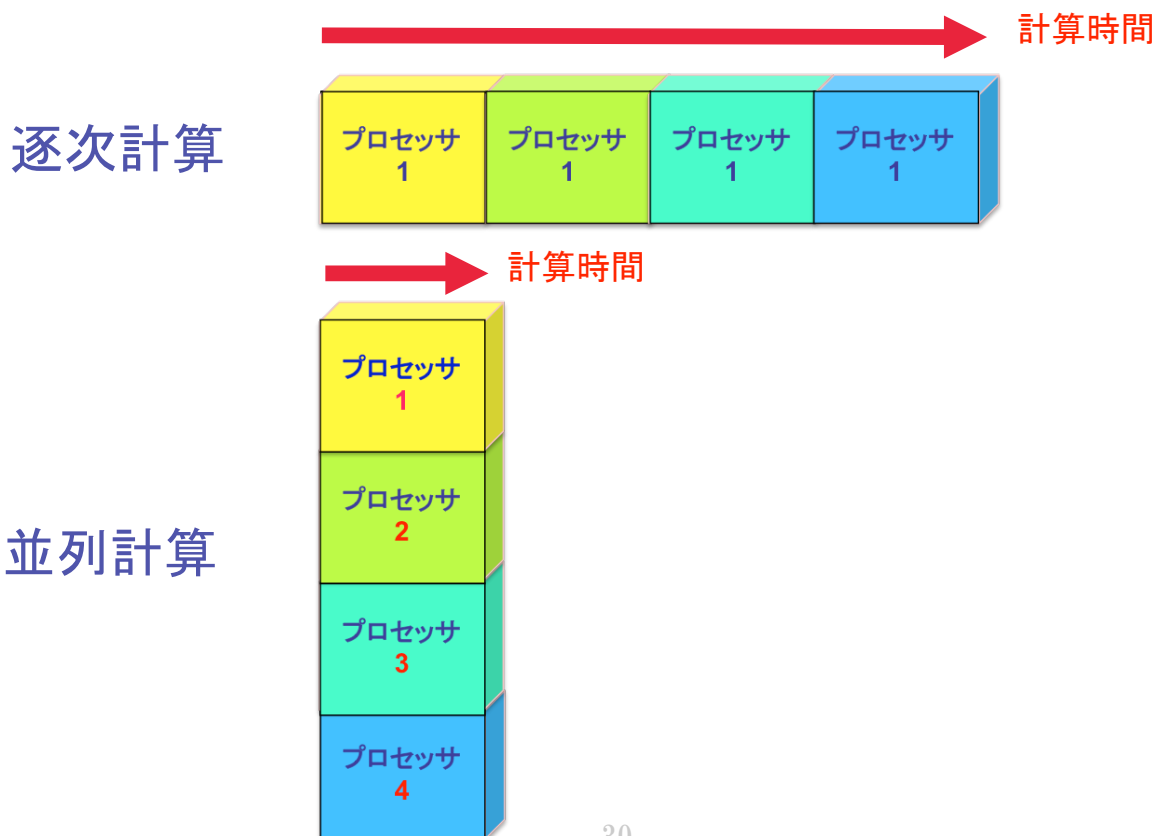


半日で終わる

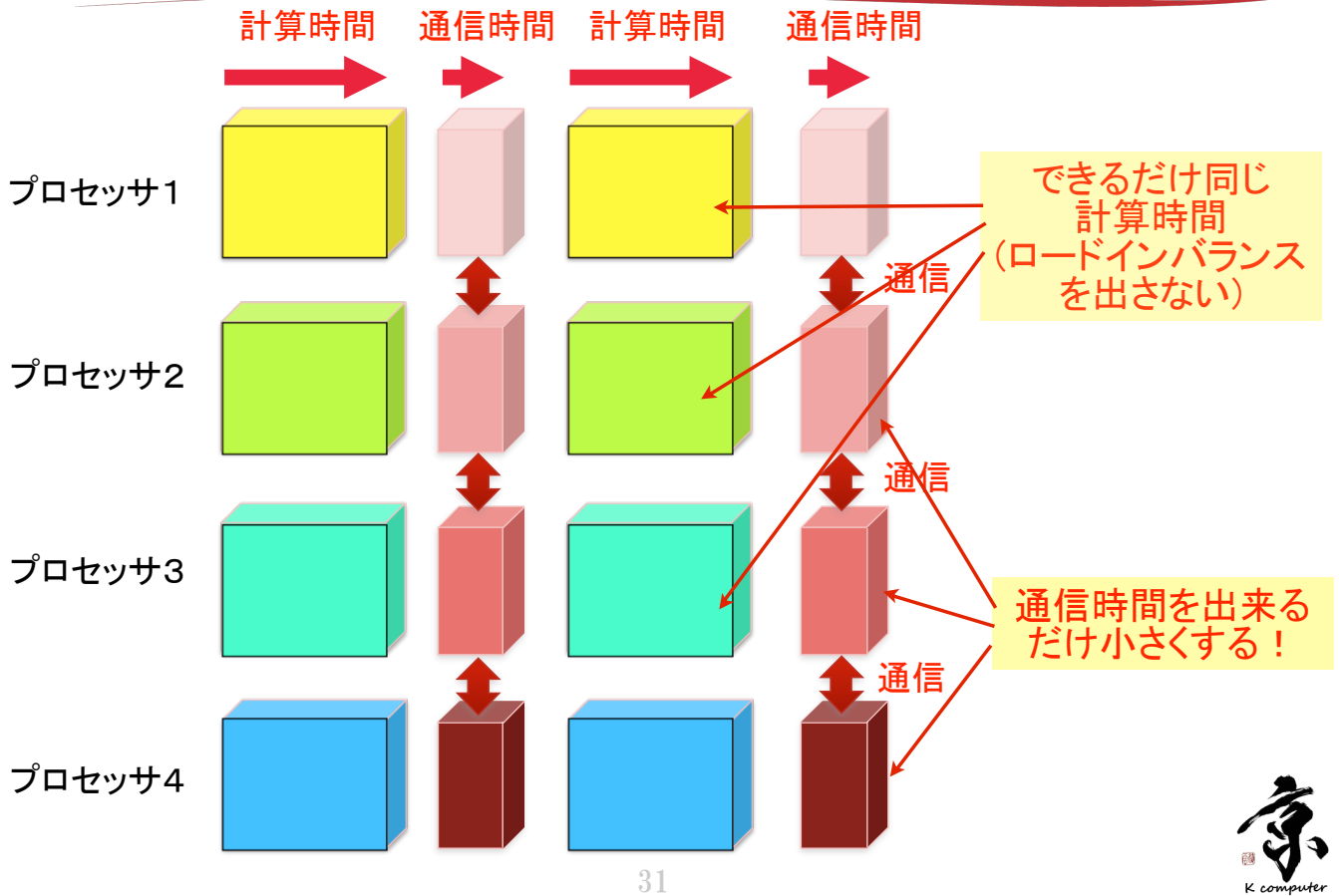
京
K computer

高並列化のための重要点

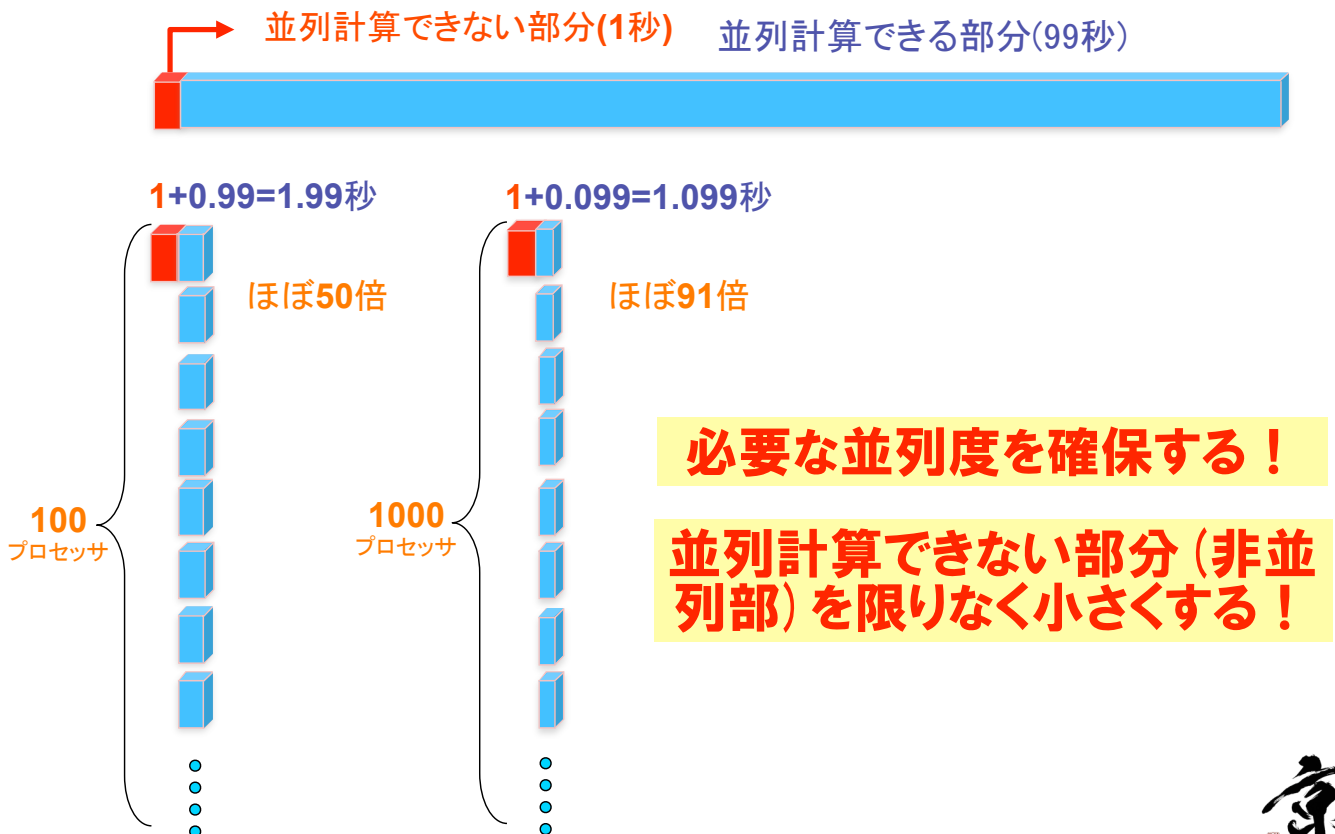
そもそも並列化とは？（1）



そもそも並列化とは？ (2)

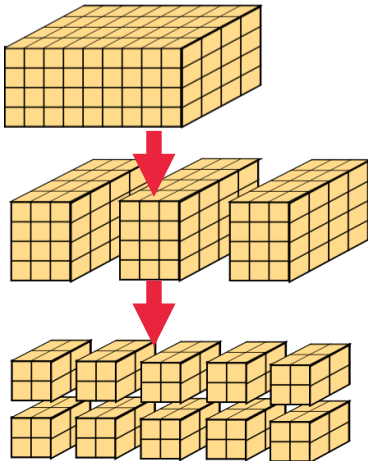


そもそも並列化とは？ (3)



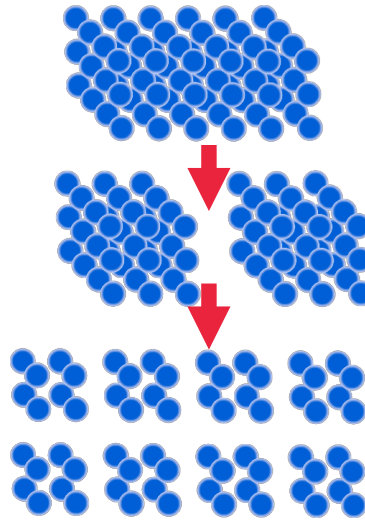
必要な並列度を確保するとは？

領域分割



- 原理的にメッシュ数以上には分割できない
- 実際的にはそんなに分割すると通信ばかりになる
- 以下の手順で分割数を見積もる事が重要
 - (1) 解きたいメッシュ数を設定し (2) 実行時間を見積もる
 - (3) 解きたい時間を設定し (4) 分割数を設定する
 - (5) 分割数が多すぎる場合、並列数の拡大を検討
 - (5) については全てのケースでできる訳ではないが後の講義でテクニックを例示する。

原子分割



- 原理的に原子数以上には分割できない
- 実際的にはそんなに分割すると通信ばかりになる
- 後は領域分割と同様

33



非並列部が問題になる場合

- 領域分割の場合、完全に領域分割されていれば非並列部が発生する場合は少ない。
- 領域分割されていぬ配列や処理が主要な計算部に残る場合あり。
- また初期処理に分割されていない処理が残っている場合もある。具体的には通信テーブルの作成等。
- 量子計算のアプリ等で領域分割でなくエネルギーバンド並列等を使用している場合は非並列部が残る場合がある。

```

subroutine m_es_vnonlocal_w(ik,iksnl,ispin,switch_of_eko_part)
  +-call tstatc0 begin
  loop_ntyp: do it = 1, ntyp
    loop_natm : do ia = 1, natm -----原子数のループ
    +-call calc_phase
    T-do lmt2 = 1, ilmt(it)
    +-call vnonlocal_w_part_sum_over_lmt1
    +-call add_vnlph_l_without_eko_part
    subroutine add_vnlph_l_without_eko_part()
      T-if(kimg == 1) then
        T-do ib = 1, np_e -----エネルギーバンド並列部
        T-do i = 1, lba(ik)
        V-end do
      V-end do
    +-else
      T-do ib = 1, np_e -----エネルギーバンド並列部
      T-do i = 1, lba(ik)
      V-end do
    V-enddo
    V-end if
  end subroutine add_vnlph_l_without_eko_part
  V-end do
  V-end do loop_natm
  V-end do loop_ntyp
end subroutine m_es_vnonlocal_w
    
```

34



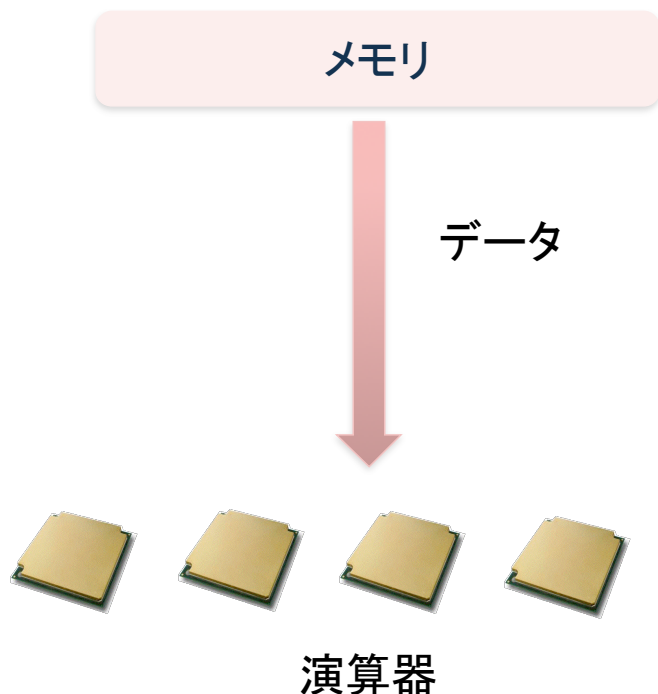
単体性能向上のための重要点

35



プロセッサの単体性能を引き出す(1)

- かつては研究者やプログラマーは物理モデル式に忠実に素直にプログラミングすることが一般的であった



メモリウォール問題

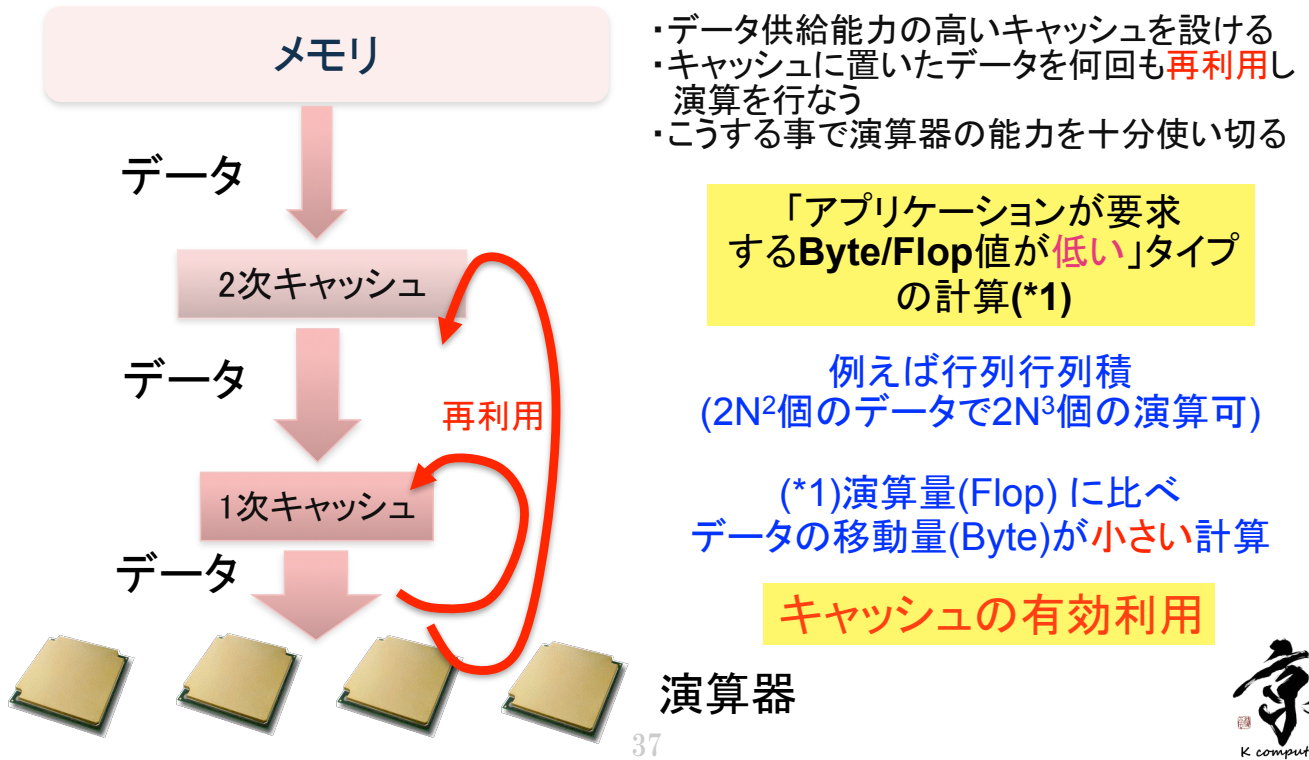
- ✓ 昔の計算機はメモリのデータ供給能力と演算器の能力がバランスしていた
- 現代の計算機は演算器の能力が高くなりメモリのデータ供給能力が相対的に不足している

36



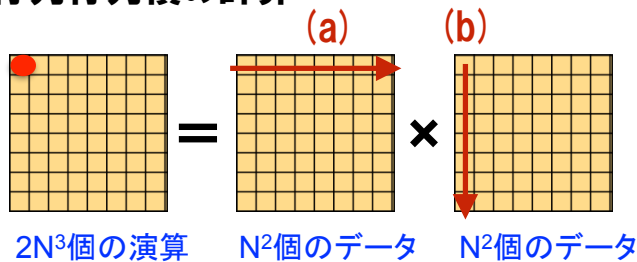
プロセッサの単体性能を引き出す (2)

メモリウォール問題への対処



もう少し詳しく説明すると

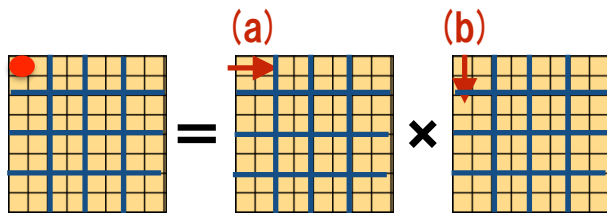
行列行列積の計算



B/F値
 = 移動量(Byte)/演算量(Flop)
 = $2N^2/2N^3$
 = $1/N$

原理的にはNが大きい程小さな値

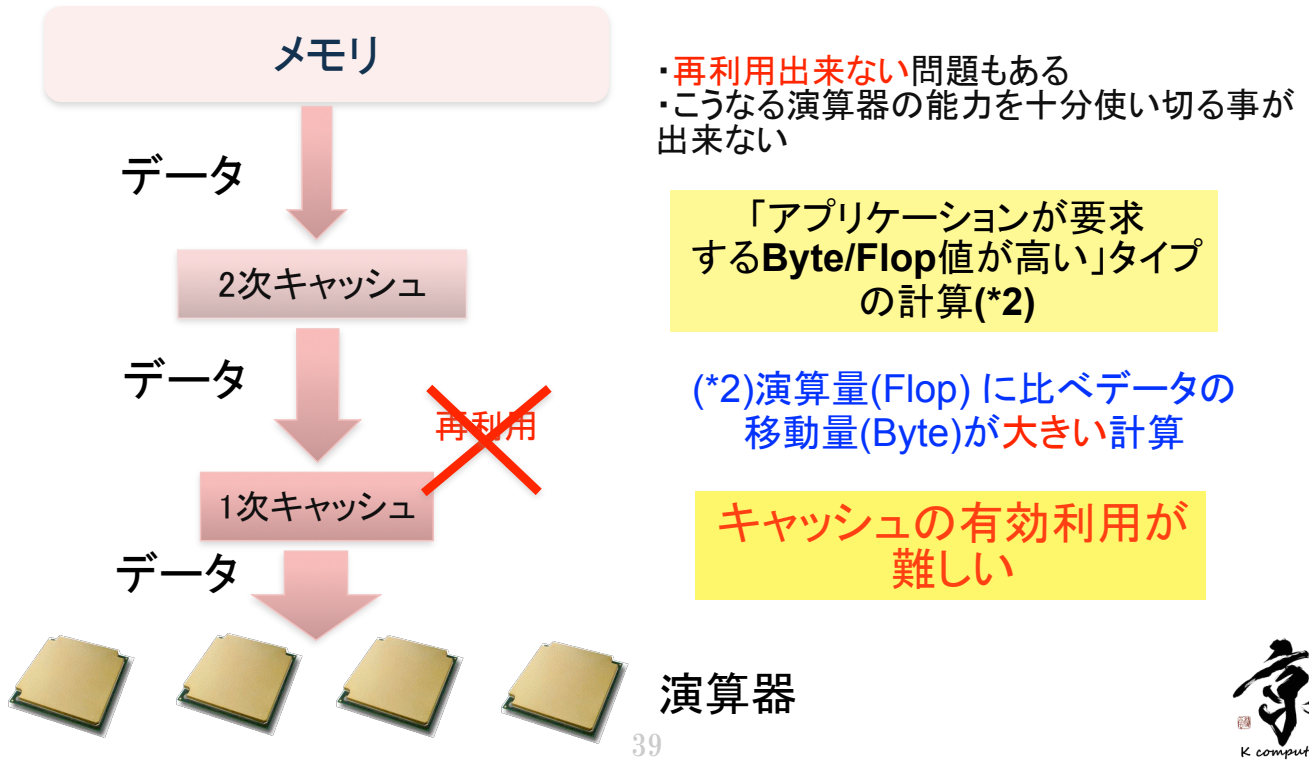
- 現実のアプリケーションではNがある程度の大きさになるとメモリ配置的には (a) はキャッシュに乗っても (b) は乗らない事となる



- そこで行列を小行列にブロック分割し (a) も (b) もキャッシュに乗るようにしてキャッシュ上のデータだけで計算するようにすることで性能向上を実現する。

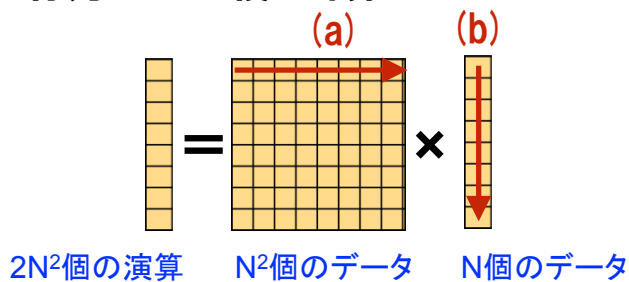
プロセッサの単体性能を引き出す (3)

とは言っても……



例えば

行列ベクトル積の計算



$$\begin{aligned}
 \text{B/F値} &= \text{移動量(Byte)}/\text{演算量(Flop)} \\
 &= (N^2+N)/2N^2 \\
 &\approx 1/2
 \end{aligned}$$

原理的には1/Nより大きな値

- 行列を小行列にブロック分割して (a) も (b) もキャッシュに乗るようにしてもB/F値は大きいので性能向上はできない。

まとめ

- **スーパーコンピュータとは？**
 - シングルプロセッサの時代
 - メモリウォール等の問題
 - 並列プロセッサアーキテクチャへ
- **アプリケーションの性能とは？**
 - アプリケーションの超並列性を引き出す
 - プロセッサの単体性能を引き出す
- **高並列化のための重要点**
 - 並列度の確保
 - 非並列部を最小化する
 - 通信時間を再消化する
 - ロードインバランスを解消する
- **単体性能向上のための重要点**
 - B/F値とは？
 - キャッシュの有効利用ができる例
 - キャッシュの有効利用がしにくい例