ポスト・ムーアの時代の HPC と流体計算

牧野淳一郎
理化学研究所 計算科学研究機構

今日の話は、計算科学研究機構エクサスケールコンピューティング開発プロジェクトの見解を反映したものではありません

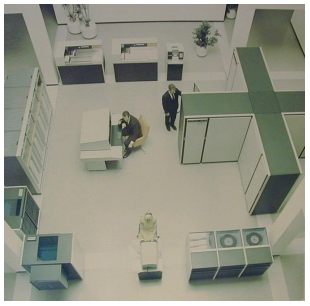
話の構成

- これまでの計算機の進歩と現状の困難
- 何故ポストムーアは問題ではないか？
- 流体計算にとって理想の計算機とは何か
- まとめ

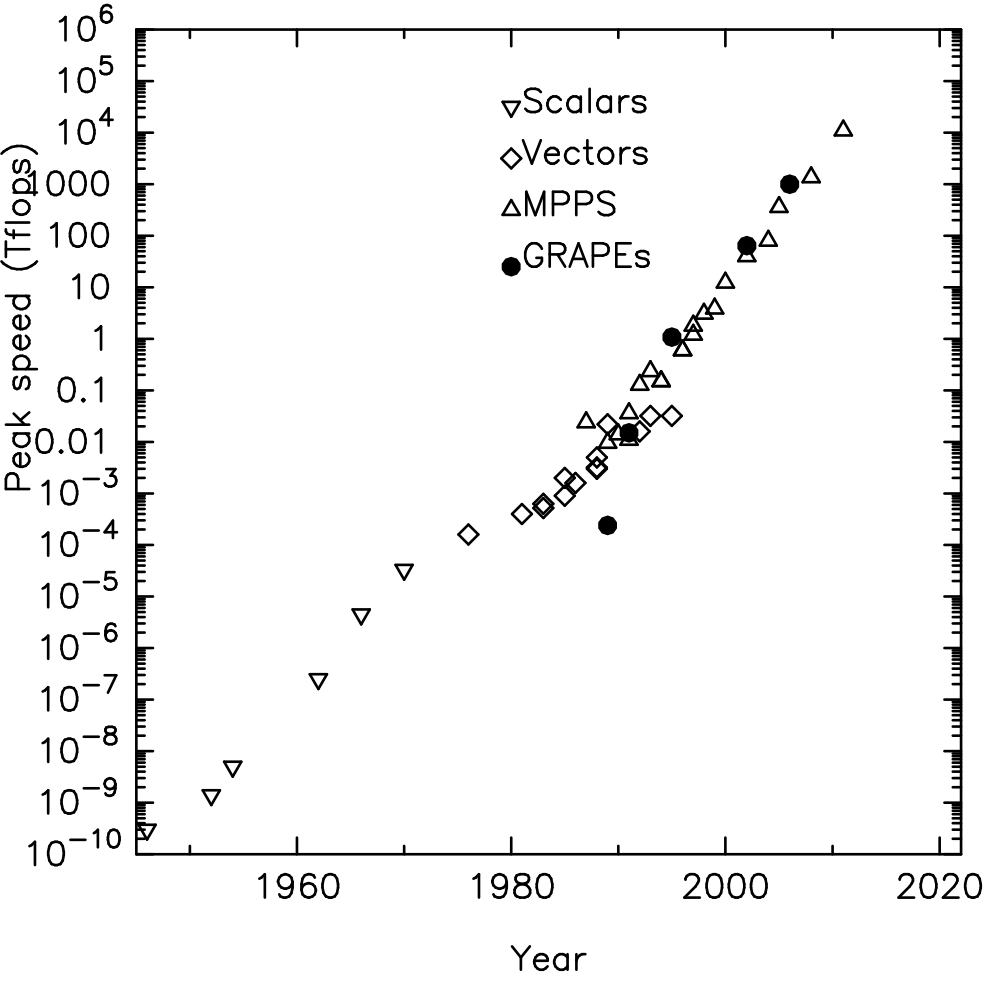
これまでの計算機の進歩

1940年代から現在までの70年間、ほぼ10年で100倍。何故そのような指数関数的進歩を長期に続けたのか?(今後はどうか?)
基本的な理由:

- 使うスイッチ素子が高速になった
- 使うスイッチ素子が小型、低消費電力になって、沢山使えるようになった
- 使うスイッチ素子が安くなって、沢山使えるようになった
(スパコンの物理的大きさは70年代が最小。そこまで段々小さくなって、そこからまた大きくなった)

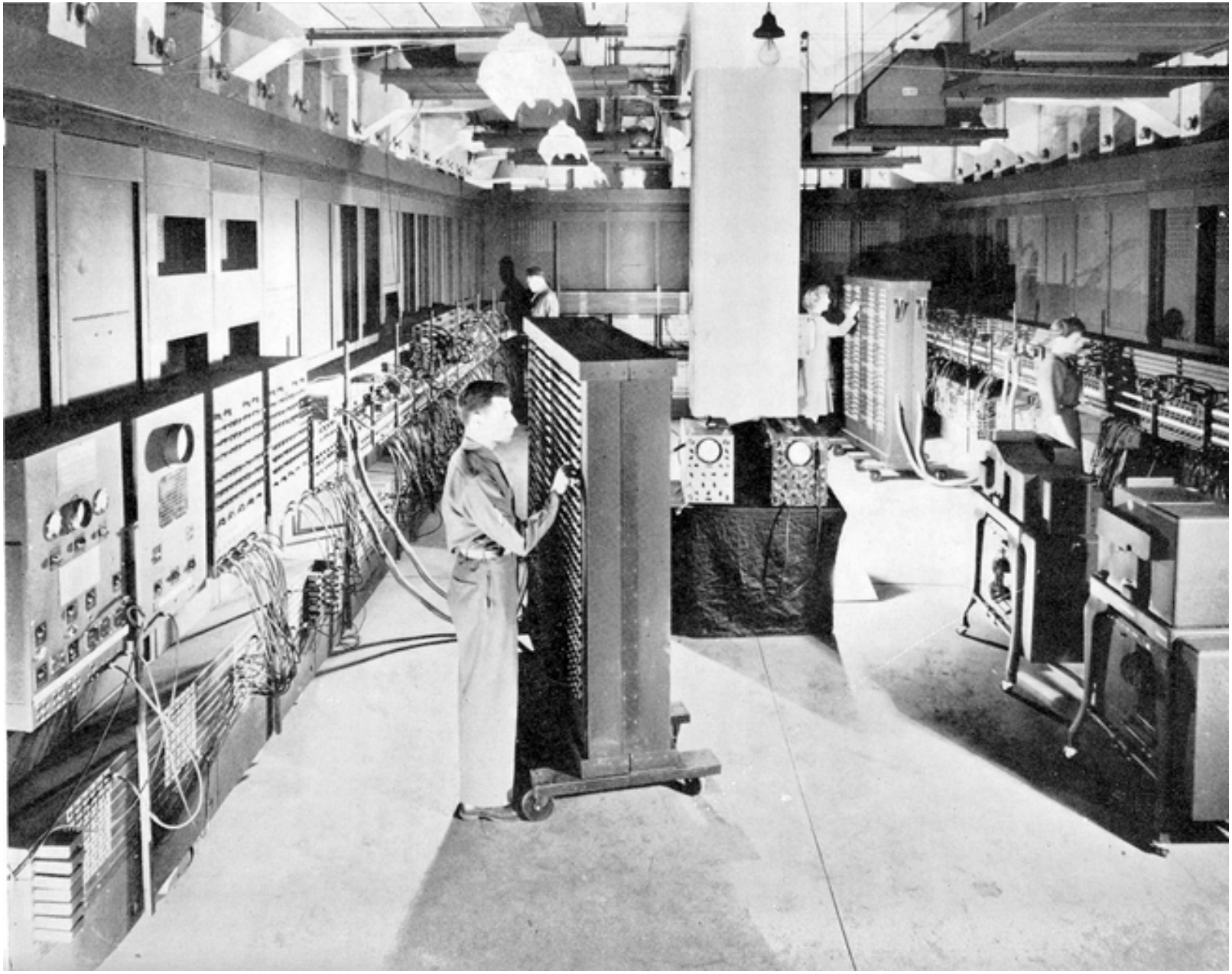




スパコンとは何か



- 単純には、「その時代で最高速クラスの計算機」
- 70 年間で 14 桁速くなった＝ほぼ 10 年で 100 倍
- 何故これほど進歩したか？は「スパコンとは何か」そのもの

1940年代の「スパコン」



ENIAC

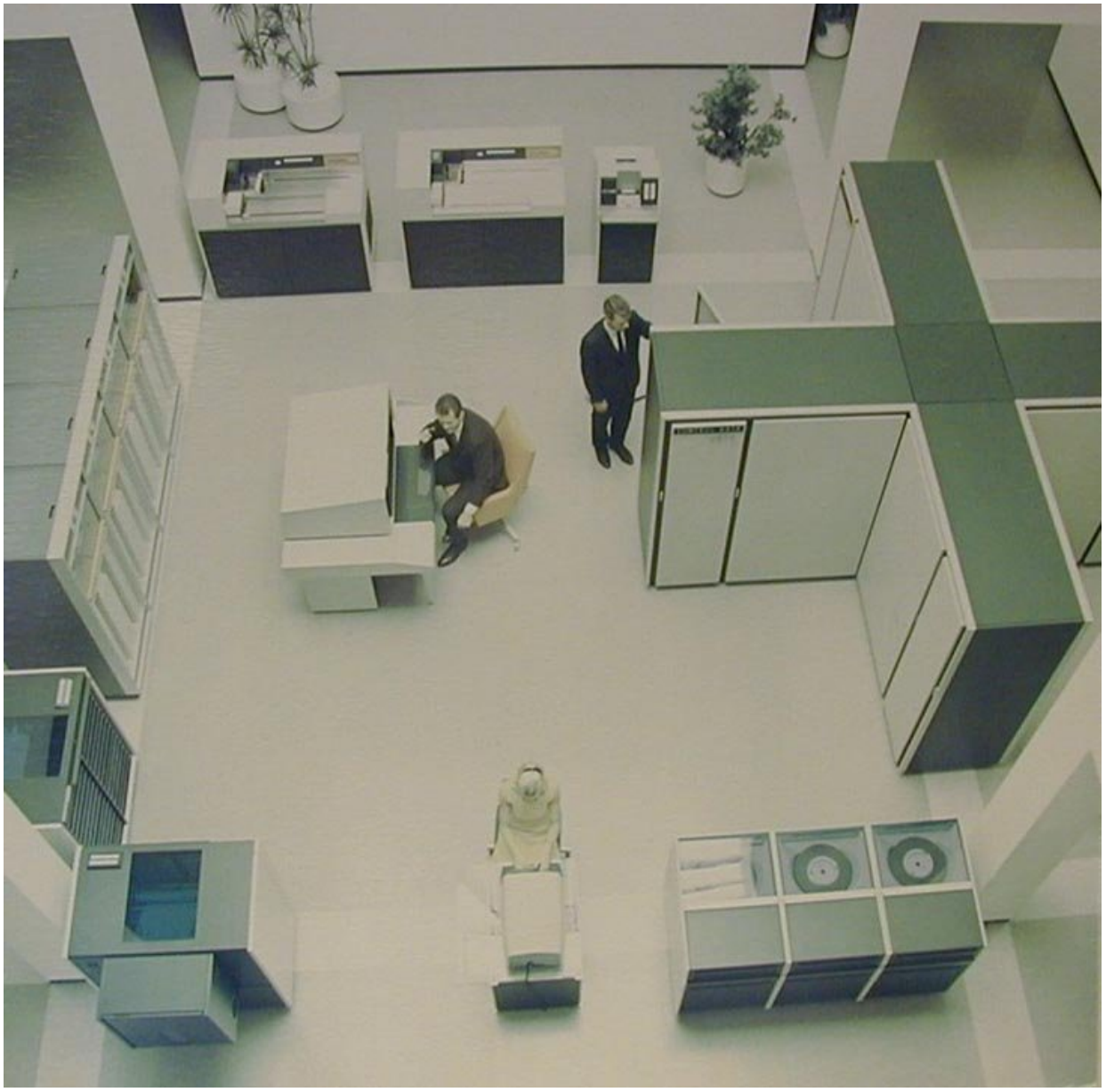
ENIAC

- 第二時世界大戦中に、アメリカ陸軍が弾道計算のために開発。完成は 1946 年
- 18,000 本の真空管、消費電力 150KW、10 進法 10 桁の数が基本
- 「プログラム」はスイッチ、ケーブルの変更で行う
- 加減算が 1 秒 5000 回、乗算が 400 回くらい (らしい)

1950年代の「スパコン」

IBM 709 — IBM 最後の真空管計算機

1960年代の「スパコン」



CDC 6600 1Mflops

CDC6600

- 1964 年から出荷。シリコントランジスタを利用。
- 10MHz クロック、1Mflops 程度の性能
- 近代的計算機アーキテクチャの原型。
  - ロード・ストアアーキテクチャ
  - 複数の演算器の「アウト・オブ・オーダー」実行
- シーモア・クレイが設計

1970年代の「スパコン」

CRI Cray-1 160Mflops

Cray-1

- 1976 年から出荷。IC (ECL 素子による小規模 IC) を使用
- 80MHz クロック、160Mflops
- 最初に成功したベクトル計算機
  - ベクトルレジスタ
  - 半導体メモリ
- シーモア・クレイが設計

ベクトル計算機って？

- 「スカラー計算機」と「ベクトル計算機」と分ける
- 「ベクトル処理」をするかどうか
- スカラー計算機 (あるいは普通の計算機の基本命令): 命令1つで演算1つ。例えば掛け算1つとか。
- ベクトル計算機: 1命令で複数の演算を処理。ベクトルの、要素同士の四則演算が基本。それだけでは足りないので色々追加機能がある。
- 必ずしも並列処理するわけではない。Cray-1 では「パイプライン処理」。長さ n のベクトル命令の実行に 定数$+n$ クロックかかる。

ベクトル計算機の利点

- 演算器の有効利用ができる。クロックサイクル毎に結果を得ることが可能。
- 普通のスカラー計算機だと、演算以外の色々な処理が必要なので、複雑な「スーパースカラー」実行をする仕掛けが必要。ベクトル計算機は単純なハードウェアで演算器を有効利用できる。

欠点:

- メモリとベクトルレジスタの間のデータ移動が高速であることが前提: 高速なメモリがないと性能がでない。

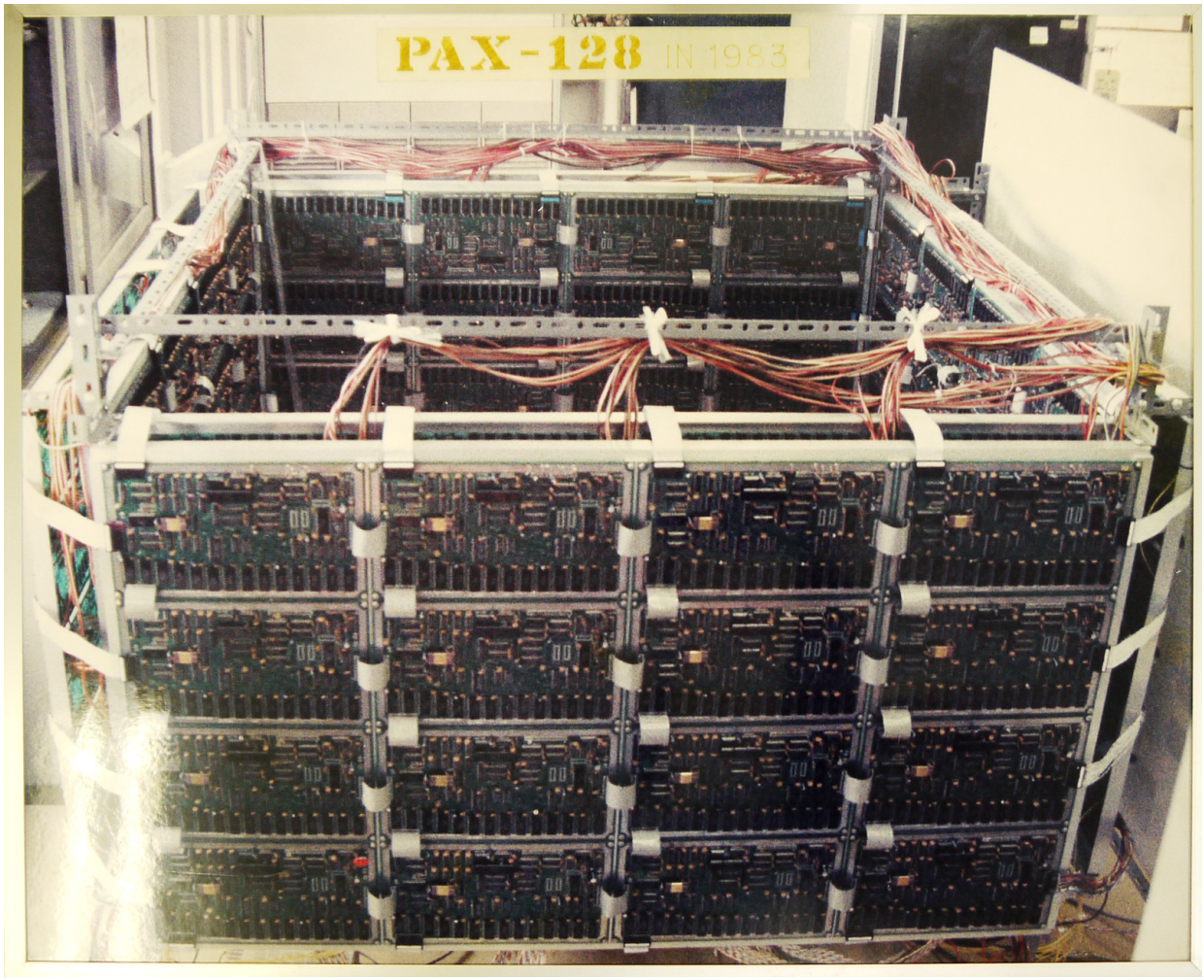
1980年代の「スパコン」

NEC SX-2 1.3Gflops

NEC SX-2

- 1985 年から出荷。CML 論理素子による LSI を使用。1000 ゲート程度 (Cray-1 の IC の 100 倍程度)
- 6ns クロック、掛け算、加減算パイプラインをそれぞれ 4 本
- 富士通、日立が同様なマシンを出荷、NEC は最後
- Cray は共有メモリマルチプロセッサに (Cray XMP, YMP)

1980年代にはこんなのも



PAX-128 4Mflops。筑波大学、星野ら

1990年代の「スパコン」

ASCI-Red, Intel Pentium Pro, 1.8TF

ASCI-Red

- 1996年に完成。200MHz Pentiun Pro プロセッサ 9216台。
- 3次元メッシュネットワーク (ほぼ2次元、、、)
- 核爆弾のシミュレーション用

2000年代の「スパコン」

IBM BG/L 360TF

IBM BG/L

- 2004年完成、カスタムプロセッサを最大 131072個使用。3次元トーラスネットワーク
- ピーク性能360TF (もっと大きい構成もあった模様)
- 後継の BG/P, アーキテクチャを一新した BG/Q の開発後、プロジェクト解散

2010年代の「スパコン」

神戸の「京」コンピュータ、10PF

「京」

- 2012 年完成、カスタムプロセッサを 8 万個使用。6 次元トーラスネットワーク
- ピーク性能 11PF
- 商用版富士通 FX10, 「ポスト FX10」が開発された。その後継が次の国家プロジェクトで開発される予定

現在 (2000 年以降) の普通のスパコン

- Intel のプロセッサを使用
- アプリケーションにあってれば NVIDIA の GPU を使用
- 沢山並べる
- 高速ネットワークを使う、あるいは Cray 社のシステムを買う
- 独自プロセッサとかはあまり使われない。
- あんまり芸がないので写真は省略

過去のスパコンの進化

何の話をしたかったかというと：何故計算機はどんどん速くなったのか？
基本的な理由:

- 使うスイッチ素子が高速になった
- 使うスイッチ素子が小型、低消費電力になって、沢山使えるようになった
- 使うスイッチ素子が安くなって、沢山使えるようになった (スパコンの物理的大きさは70年代が最小。そこまで段々小さくなって、そこからまた大きくなった)

素子の高速化

- といっても、真空管でもそれなりに速かった。
- スイッチング速度が重要でないわけではないが、配線を信号が伝搬する速度のほうが昔から重要。
- 昔は信号はほぼ光の速さでつたわった。
- 最近の LSI 上の配線は非常に細く (抵抗が大きく)、キャパシタンスを充電しないといけないことによる RC 遅延のため、信号が伝わる速度は光速度よりはるかに低い。
- 太い配線に大電流を流せば速いが、大量の電力消費になる

素子の小型化

- 真空管 → トランジスタ →IC という進化は 1970 年代までは重要
- サイズだけでなく、消費電力が下がることが重要
- 80 年代から重要になったのは CMOS LSI の微細化。10 年でサイズが 1/10 になる
- CMOS 素子では (2000 年くらいまでは) 微細化すると電圧を下げることができ、消費電力が下がり、速度は向上した。いわゆる CMOS スケーリング。
- 2000 年頃からは電圧が下がらないので、電力はちょっと下がるが速度は上がらなくなった。いわゆる CMOS スケーリングの終焉。
- そろそろ微細化も困難になってきた。また、トランジスタの構造・製造工程が複雑になり、微細化するとかえって価格上昇するようになった。いわゆるムーアの法則の終焉。

素子の性能向上、サイズ低下と スパコンの性能向上

「スパコンの進歩」は4つの時期に分けられる

I スパコンが完全パイプライン化した乗算器をもたない時期 (1969年まで)

II スパコンは1つ以上の演算器をもつが、1チップにはまだ演算器1つが入らない時代 (CMOS では1989年まで)

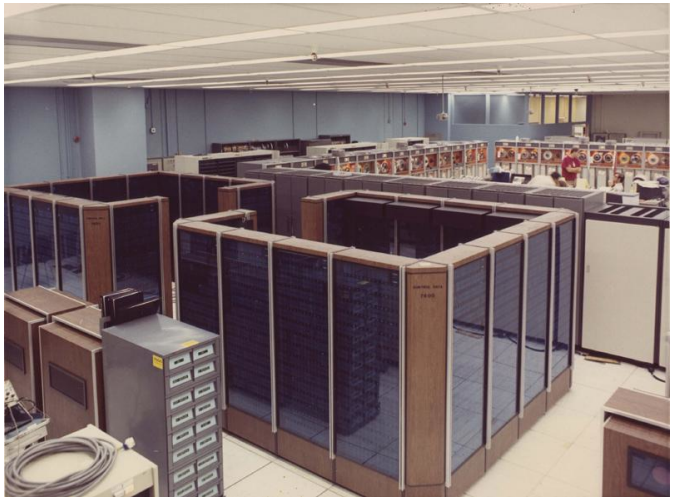
III1チップに複数の演算器がはいるが、微細化すると動作クロックが上がり、電力が下がった時代 (2001年頃まで)

IV チップに多数の演算器がはいるが、微細化してもクロックが上がらず、電力がへらない時代 (2001-)

このそれぞれで、素子の性能向上がどう使われたかは違う。

I期: 1演算器未満の時代 ~ 1969

- この時期には、メモリアクセスのほうが演算より速い
- 演算器の性能向上が最重要課題
- CDC 6600 くらいまで。
- 計算機の構成方法もまだ手探り。
- CDC 7600 で1演算器はいるようになる



CDC7600 36.4MHz clock

1969

IC はまだ使ってない

II期: 1演算器以上の時代 ~ 1990

- 1つ以上の演算器を有効に使うことが課題
- パイプライン化 (ベクトル命令)、複数の演算ユニットの SIMD and/or MIMD 並列実行が必要になる
- 演算器の数が増えると、メモリとどうつなぐかが課題になる。
- 演算器 16-32 個で破綻する (1992 年頃): メモリと演算器の間のデータ移動回路が大規模・複雑になり過ぎるため
- 末期の計算機: Cray T-90, 日立 S-3800
- 富士通 VPP500 では、分散メモリにすることで当座しのごうとした:ある程度の成功
- 他の方向: 1チップマイクロプロセッサでの分散メモリ。プロセッサ間の通信は細い線でいいことにする。これが 90 年代以降の主流になった

III期: 1チップ高速化の時代 ~ 2005

- 1チップマイクロプロセッサを分散メモリで多数つなぐのは II期末期から
- 1チップに1演算器以上がはいるが、ありあまるトランジスタを演算器を増やすのにはつかわないで動作クロックの向上、キャッシュメモリの大型化に使った時代
- 1990年代。 牧野の意見としては「失われた10年」
- 計算機全体としては II期に起こった問題がチップレベルでも起こるのを先送りにした
- メモリとの接続は速度が不足になった(memory wall): キャッシュでごまかす方針
- 迷走したプロセッサ開発も多い:各社の複数チップ共有メモリプロセッサ。(失敗プロジェクト: ASCI Red 以外のASCI マシン)
- 末期の計算機: Intel Pentium 4, DEC Alpha 21264

IV期: 1チップ多演算器化の時代 ～ 2020?

- 引き続き、1チップマイクロプロセッサ、分散メモリ。
- デザインルール 130 nm あたりから、動作電圧低下、速度向上に限界
- マルチコア化、コア内 SIMD 化を同時に進めている
- 80年代のベクトルスパコンの辿った道を追いかけている
- ということは、16-32コアで破綻がくるはず
- 破綻がくることの予見: Intel Xeon Phi (60コア)
- GPGPU はちょっと別。

まとめると

- 半導体技術は 2000 年頃に CMOS スケーリングが終焉、さらに微細化自体のスローダウンが進行中であり、もはや微細化が技術進歩を牽引していない。
- 昔の Cray-1 みたいな「スパコン」の進歩は、プロセッサコア 32 個くらいの並列度で限界、破綻した
- 現在のマイクロプロセッサは、それくらいのコア数に達してきており、破綻が近い (あるいは既に破綻している)
- この 2 重の困難をどう解決するか、という展望が必要。
- 「ポストムーア」は本当の問題ではない。

何故ポストムーアは問題ではないか？

現代の「最先端のマイクロプロセッサ」の設計は驚くほど非効率的

- チップ上のトランジスタの $\mathbf{97\%}$以上
- 消費電力の (多分)$\mathbf{90\%}$ 以上

は、「演算論理以外」に使われている (**Xeon Phi** の場合。**Xeon** だともっと非効率)。

何故それほど非効率なのか?

根本的な理由: 「計算機の設計」はまだ科学にも工学にもなってない。

- 熱機関の設計: 熱力学第一、第二法則が理論限界＝理想の熱機関を与える
- 航空機の設計: 誘導抗力・摩擦抗力・それ以外 (「圧力抗力」) への分離が「理想の航空機」を与える

つまり: 科学的設計＝理論限界に近づけること。

ところが:(トランジスタではなくて) 計算機の理論限界、という考え方自体がまだ存在していない。

というわけで

計算機設計者が本来していないといけないこと:

1. 理想の計算機＝理論的限界を明らかにする
2. 現実の設計をそれに近づける

現状やっていること

1. 現在ある設計を、現在ある色々な制約の範囲で色々いじってみる。思い付きのアイディアもいれたりする。
2. よさげなものを作ってみる
3. 会社がつぶれていなければ上を繰り返す

現状批判はいいとしてどうしたものか？

明らかにするべきこと:

理想の計算機を明らかにし、それに近づいていくべく努力する

では、理想の計算機とは？

文部科学省による「指標」

(2/10 日経新聞)
以下の 4 指標

- 省エネ性能
- 計算速度
- 使いやすさ
- 幅広い分野に活用して画期的な成果を出せるか

この「指標」の問題

- 定量的でないものがまざっている。
- 定量的なものも科学的でない。限界に比べた効率という概念がない。

「省エネ性能」には、いくつかの仮定の下で限界を定義できる。

「省エネ性能」の科学的定義

- 「論理設計」の効率を問題にする＝半導体技術は与える、ないし規格化する
- (細かいことをいいだすと物理設計も問題だがこれはちょっとおく)
- トランジスタは無限に沢山使ってもいいことにする (熱力学における準静的変化に対応)

この仮定の下で、ある問題をあるアルゴリズムで解くのに必要な電力消費の最小値は存在するはず。これを理論限界＝理想の性能とすればよい。

理論限界をもうちょっと具体的に

- 問題を解くのに必要な演算の組合せ論理だけの消費電力を考える
- 言い換えると、以下のようなものの電力はすべて考えない
  - メモリ・レジスタ読み書き
  - データ移動
  - 制御回路の消費電力
  - リーク電流
- 演算毎に「必要最小限の精度」で行うとする

  そうすると、少なくとも問題･アルゴリズムを決めれば「理想」が定義可能

流体計算にとって理想の計算機とは何か

まず、流体計算とは何かから、、、
いくつかの観点

- 規則格子・不規則格子・粒子
- 陽解法・陰解法
- (規則格子では) 適応的・固定

以下では、「規則格子、陽解法、非適応的」を考える。

(牧野の偏見:「規則格子、陽解法、非適応的」と「粒子・陽解法」以外はいずれ滅びる)

規則格子陽解法と理想の計算機

- 差分スキームまで固定すれば、形式的には、「あるタイムステップでの全格子点情報」を入力すれば「次のステップでの全格子情報」がでてくる論理回路は構成可能。
- この論理回路の消費電力の最小値が理想を与える

とはいえ、差分スキームまで計算機にしちゃうのはちょっと無理。差分だけでない計算も色々ある。プログラム可能にする妥協は必要。
妥協の例:

- 演算器毎にレジスタとメモリをつける
- 演算器間は 2D ないし 3D メッシュネットワークでつなぐ
- 複数チップ間はチップ内部のネットワークを延長

(まあそのポスト「京」FS で我々が考えてたものです)
理想との差は評価可能

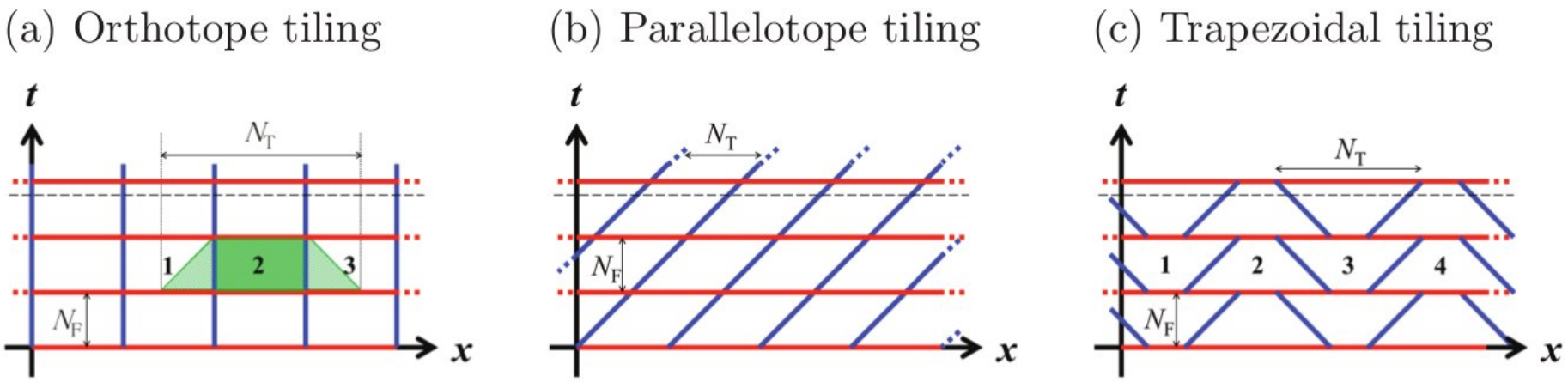
現実的問題点

- チップに載るメモリだけでは(多数チップ使っても)あまり大きな計算はできないのではないか？
- チップ間ネットワークはチップ内に比べてバンド幅が小さくレイテンシが長いので、つないでも並列計算の効率が上がらないのではないか？

メモリ量の問題

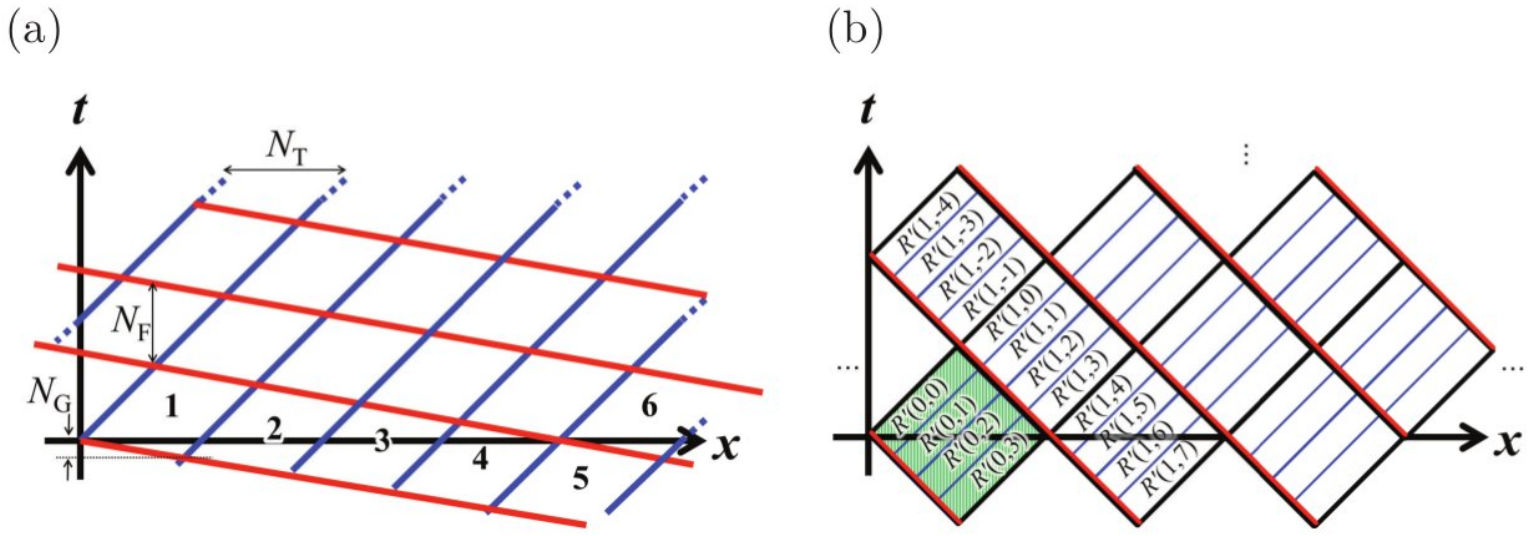
- 現在の 28nm テクノロジーだと、 64-128MB/チップくらい。1万チップで1TB。演算速度が10-100PF くらいいっちゃうのでちょっと小さ過ぎる
- もちろん、外付けメモリをつければいいが、メモリバンド幅で性能が律速、消費電力もこっちで決まって理想から遠ざかる。

Temporal Blocking



- 局所的な領域をチップ内メモリに読み込んで複数時間ステップ進める
- 外付けメモリへのアクセスを減らせる
- 3次元で並列化できる方法や性能向上の理論限界はよくわかってなかった

Optimal Temporal Blocking



Muranushi and Makino 2015

- 3次元で並列化可能な方法を定式化した
- 理論限界も求めた
- 現在のキャッシュあり計算機でも理論的には有効

とはいえ、、、、

こんな計算法のプログラムは人間には書けない。最適化とか並列化とかムリ

人間以外にプログラムを書かせる

(人外が書くという話では一応はない)

- 人間は差分スキームくらいを記述
- あとは計算機が素晴らしいプログラムを勝手に吐く

そんなうまい話があるはずない

- 昔なんかそういうのあったよね?日立の **DEQSOL** とか
- あれ消えたのはそれなりの理由が (とはいえ、共有メモリベクトル機でしか動かなかったので、、、)

車輪の再発明？

Formura `https://github.com/nushio3/formura`

- **AICS** で鋭意開発中 (基本的に村主君がやっている)
- 言語仕様定義した。まだ夢のような **temporal blocking** とかはできてないけど実行可能なコードはでる。
- 目標としては年度内に **temporal blocking** と並列化まで実装。

Formura の例

```
dimension :: 3
axes :: x, y, z

ddx = for(a) (a[i+1/2,j,k] - a[i-1/2,j,k])/2
ddy = for(a) (a[i,j+1/2,k] - a[i,j-1/2,k])/2
ddz = for(a) (a[i,j,k+1/2] - a[i,j,k-1/2])/2

∂ = (ddx,ddy,ddz)

Σ = for (e) e(0) + e(1) + e(2)

begin function init() returns dens_init
      float [] :: dens_init = 0
end function

begin function dens_next = step(dens)
  float :: Dx, Dt
  Dx = 4.2
  Dt = 0.1
  dens_next = dens + Dt / Dx**2 * Σ for(i) (∂ i . ∂ i) dens
end function
```

この例 (村主君謹製) では ddx とか ∂ とか $\sum$ とかはみんなマクロ (ラムダ式、、、) こうも書いても同じはず:

dens_next=dens + Dt/(Dx*Dx) *(dens[i,j+1]+dens[i,j-1] +dens[i-1,j]+dens[i+1,j] -4*dens[i,j])

まとめ

- 「ポストムーア」は今度は本当。シリコン半導体技術の限界は確実にきている。
- しかし、計算機設計という観点から見ると、問題はそこではなくて、非効率的な設計のほう。
- 例えば陽解法流体計算にとって理想的な計算機、というのは構成可能。演算器とメモリをペアにした小規模なプロセッサを多数並べる。チップ内部は3次元ネットワーク。
- 実用的には外付けメモリは必須で、これはバンド幅不足。但し、**temporal blocking** で回避可能
- **temporal blocking** するプログラムを人間が書くのは無理、という問題はドメイン特化言語からのコード生成で回避可能。
- 話としては今年度中に「京」で動く実用レベルのものができるはずなので皆様御期待を。