ポスト「京」プロジェクト
ハードウェア概要

牧野淳一郎
理研 計算科学研究機構 粒子系シミュレータチーム
東工大 地球生命研究所

理研和光ー AICS 合同シンポジウム「京、ポスト京と基礎物理」 2014/1/7

とタイトルは書いてありますが

ポスト「京」は「汎用部」＋「加速部」ということになっていて

汎用部の話は今日はしません。

ちょっとだけ汎用部の話をしておくと

ポスト「京」汎用部

- プロセッサアーキテクチャ: 多分「京」と同様。SIMD の幅、1 ノード (1 チップ) のコア数が増える
- ピーク性能: 1 エクサよりかなり低い (半分いくことは加速部がキャンセルにならない限りない)
- メモリバンド幅、ネットワークバンド幅: 「京」に比べて相対的に悪くなる

「京」からの予測の範囲の機械。使いかたも予測の範囲で。

以下、加速部の話

話の構成

- スパコンの進化:: 1950-2010
- 現状の問題
  - 消費電力
  - 並列処理オーバーヘッド
  - ソフトウェアの開発/メンテ
- 解決？
- 日本の「ポスト「京」」

スパコンの進化: 1940-2000

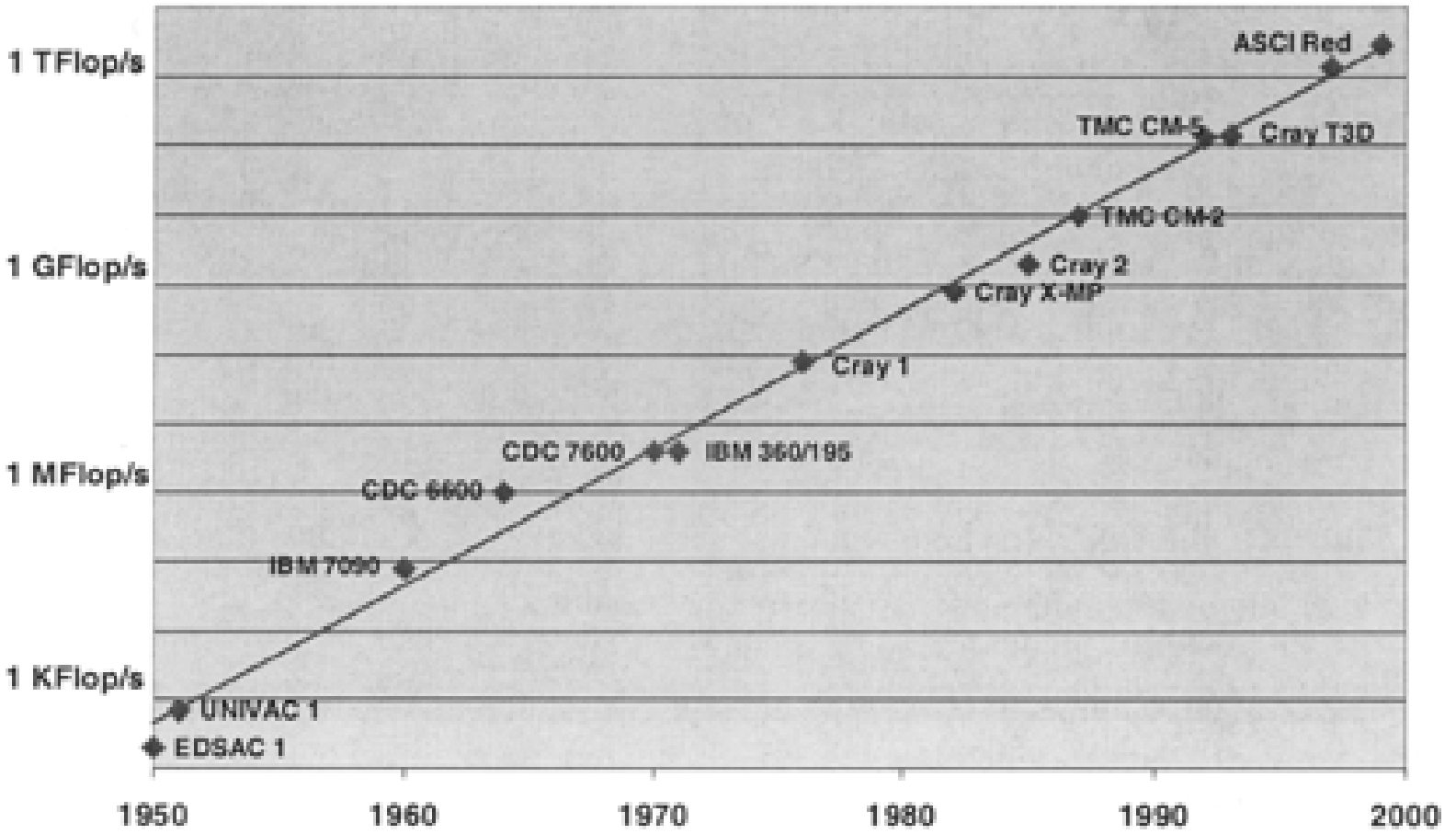
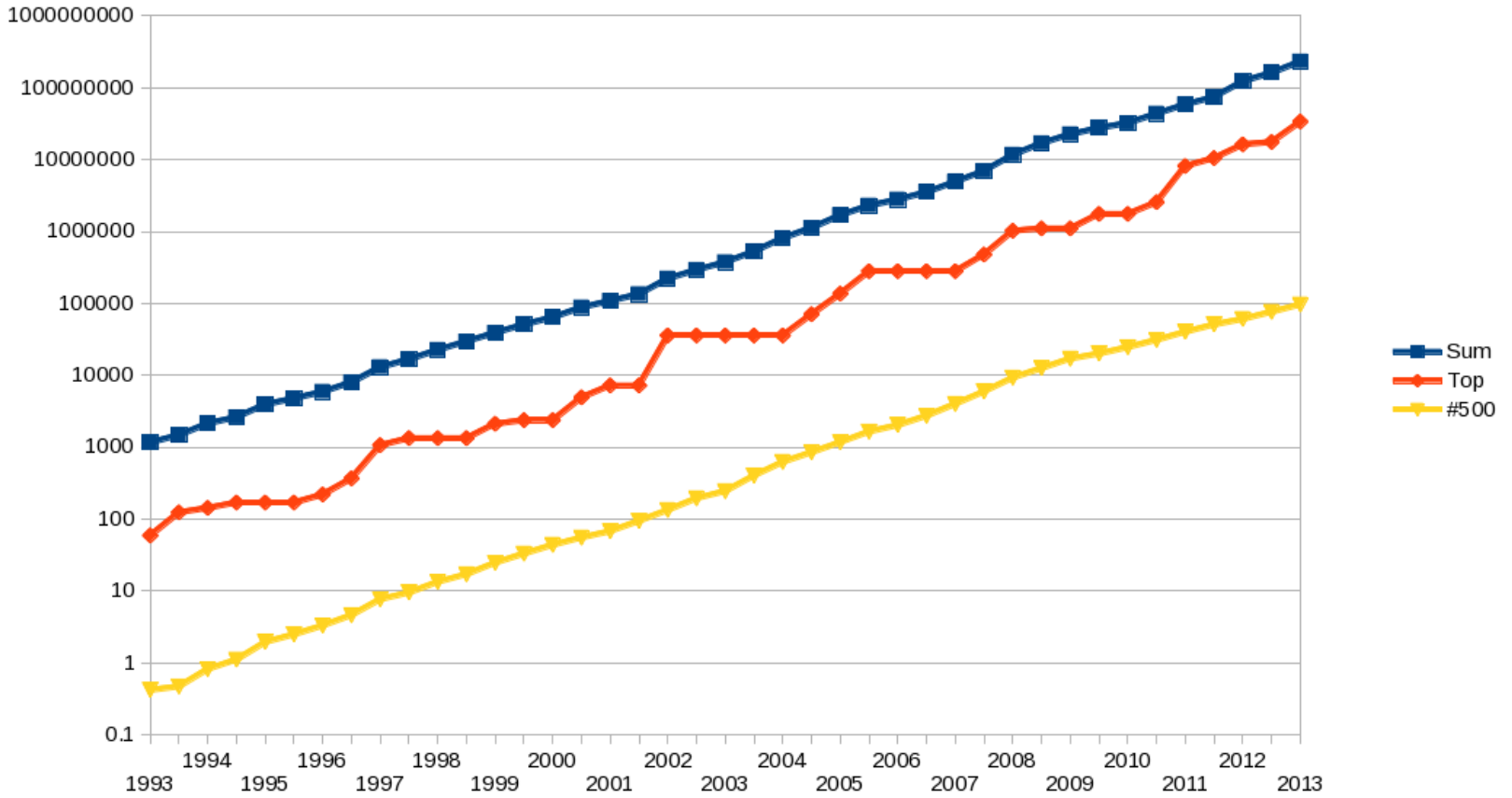


Figure 1. *Moore's law and peak performance of various "supercomputers" over time.*

1940-2000: 10 年に 100 倍

スパコンの進化: 1993-2013

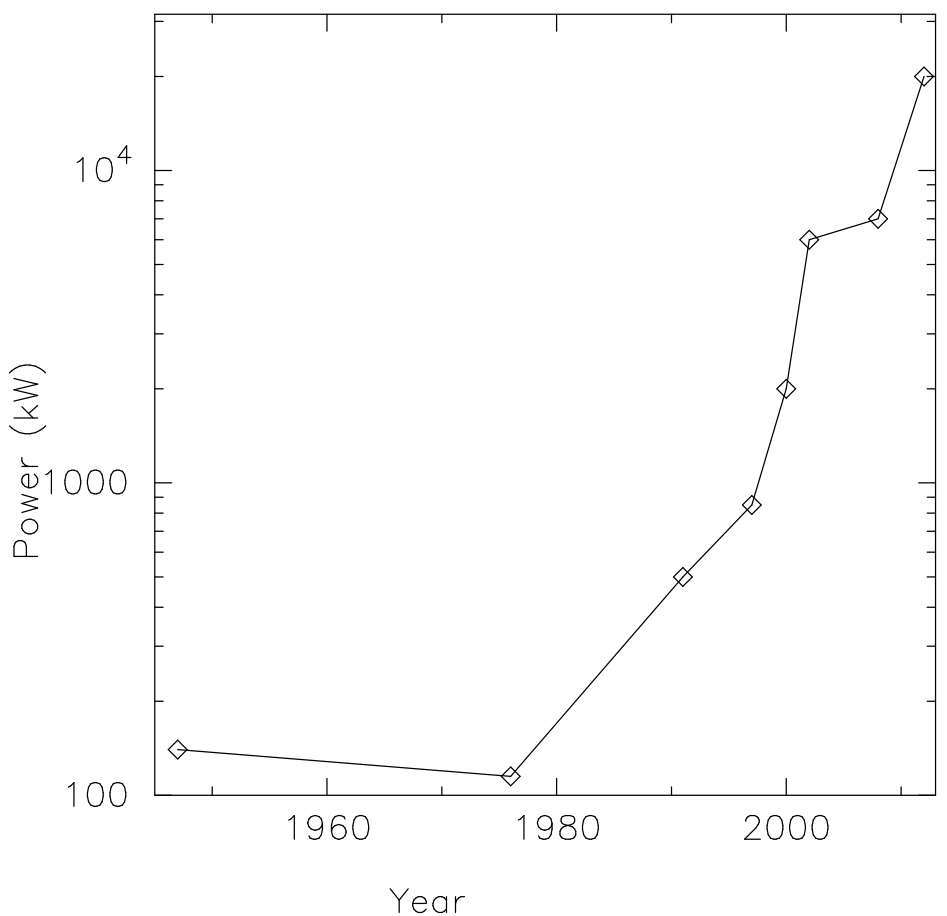


1993-2013: 10年に 500倍!?

問題1: 消費電力

ENIAC	1947	140kW
Cray-1	1976	115kW
Cray C90	1991	500kW
ASCI Red	1997	850kW
ASCI White	2000	2MW
ES	2002	6MW
ORNL XT5	2008	7MW
「京」	2012	20MW

グラフにすると……



ENIAC から Cray-1 まではたいして変わらない

1975-95 の 20 年間に 10 倍

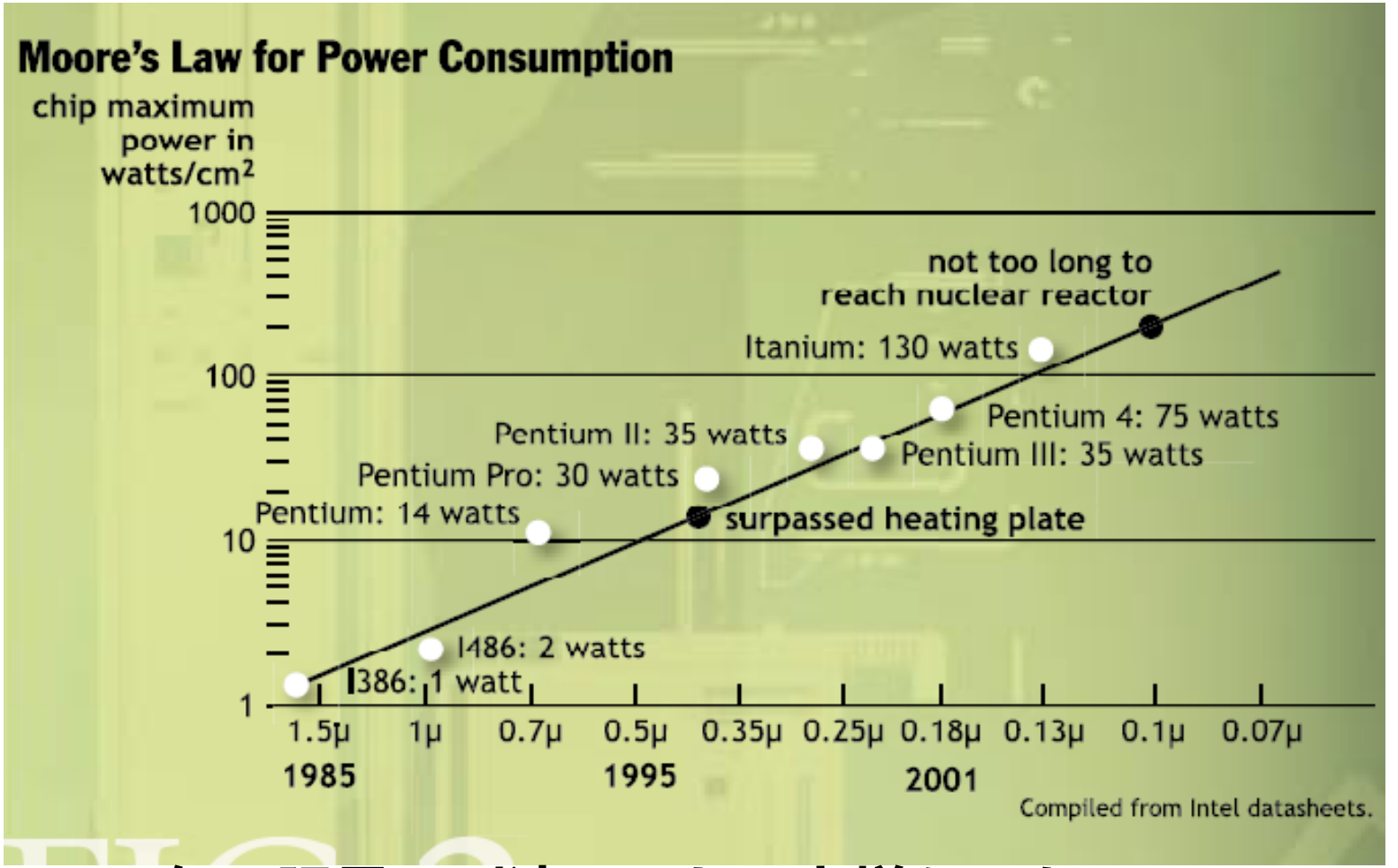
1995-2012 の 17 年間に 30 倍

指数関数より速い増加 (有限時間でクラッシュ)

何故?

- 沢山お金掛けるようになった:: ASCI Red: 50億, 「京」×百億
- チップあたり (ないし面積あたり) の消費電力増加
- チップ面積あたりの価格低下

シリコン面積あたりの消費電力



2003 年に限界に到達、それ以上増えてない (BTX の失敗)

問題 2: 並列化オーバーヘッド

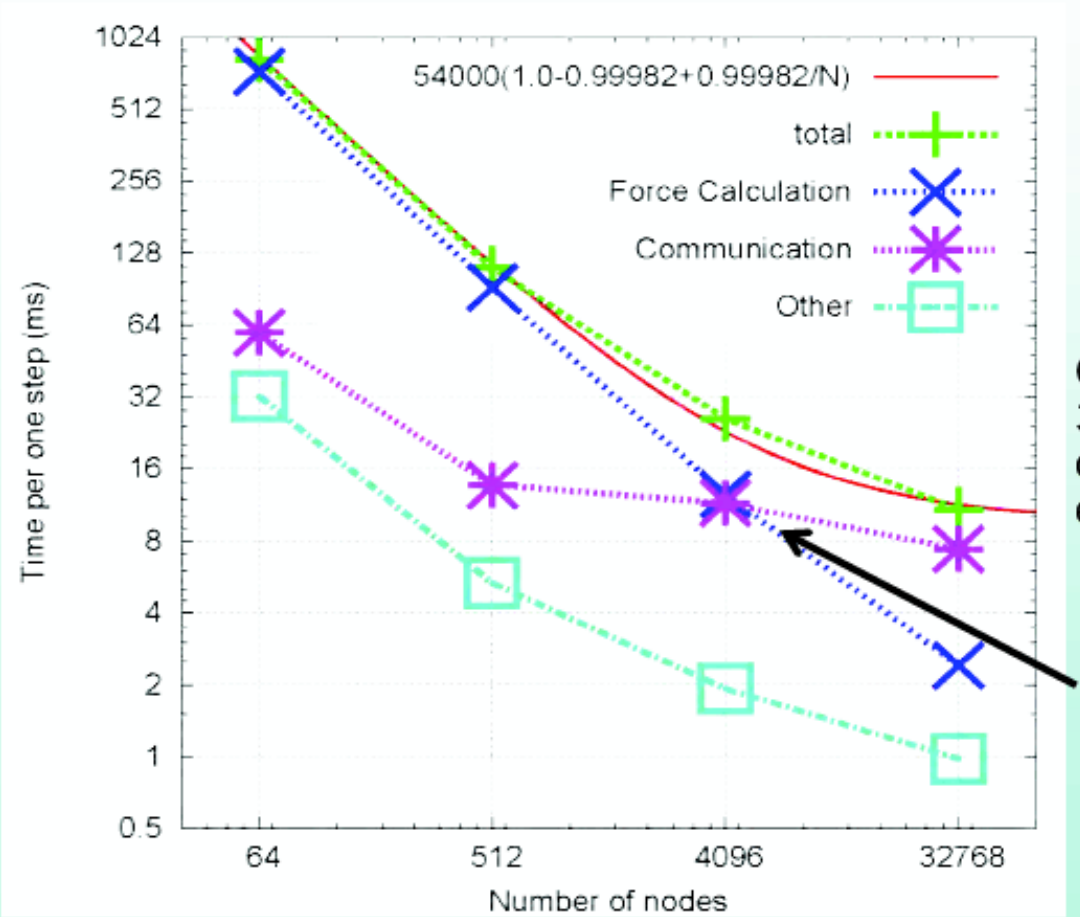
浮動小数点演算器の数 (乗算+加算で1つ)

Cray-1	1976	1
Cray C90	1991	16
ASCI White	2000	16,384
地球シミュレータ	2002	40,960
「京」	2012	2,820,096

「京」は、大自由系の短時間計算はできても、小 (といっても100万とか、、、) 自由度系の長時間計算にはむかない

性能スケーラビリティの例

Strong Scaling (内訳)



Cutoff 28Å,
3,349,656 atom ,
calculate energy
every 4 step

Cross over at
817 atom/node

「京」での分子動力学計算

- 1 タイムステップが 5ms を切らない
- 通信オーバーヘッドが問題

5ms で十分速いか？

- タンパクとかだとマイクロ秒くらいしか計算できない
- 専用機 ANTON は 100 倍以上速い

現在のアーキテクチャの延長では大きな改善は難しい。

並列化オーバーヘッドの起源

演算器の数自体が問題なわけではない

- 「通信時間」のほとんどは、メモリ読み書きのオーバーヘッド
- CPU→キャッシュ→メモリ→NIC→NIC→メモリ→キャッシュ→CPU
- 同期や総和になるともっと大変なことに

問題 3: どうやってプログラム書くの？

- MPI
- OpenMP
- SIMD 拡張
- Cache の有効利用
- アクセラレータ
- ...
- ...

解決の方向は?

- 消費電力と通信オーバーヘッドの両方を減らしたい
- メモリはそんなにいらない (という問題も多い)。100 万原子:100MB。1 兆になっても 100TB。タイムステップが少し多いと 1 兆粒子はエクサでも終わらない。

一つの方向:

- 「小さな」オンチップメモリしかもたないプロセッサチップ (「小さなといっても 256MB とか、、、))
- 単純なコアを多数 SIMD で動かすことで、同期、通信のオーバーヘッドを減らす。

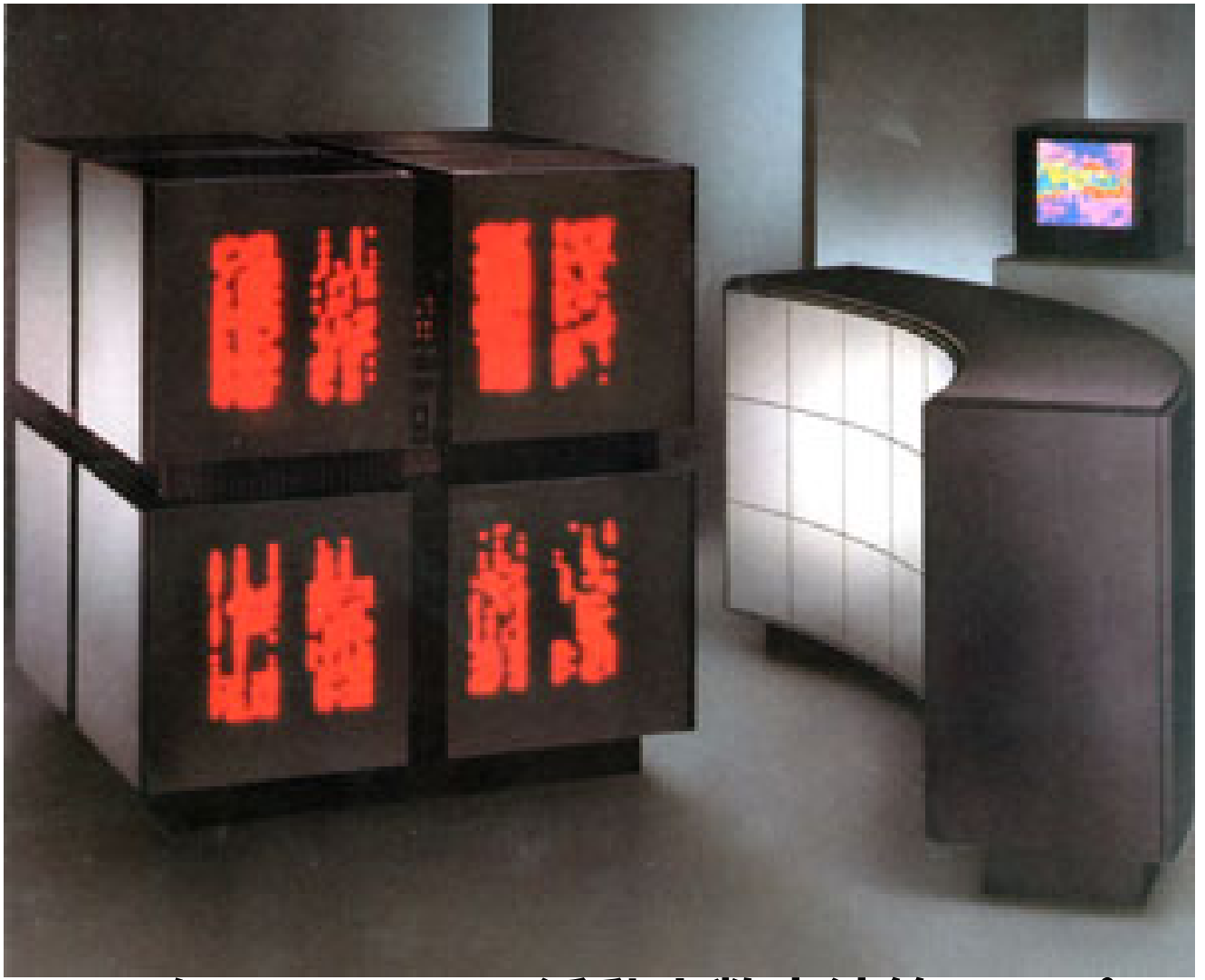
超並列SIMD計算機

— A lost technology —

- Goodyear MPP (1970s)
- ICL DAP (Late 1970s)
- Thinking Machines Connection Machine-1/2 (Late 1980s)
- Maspar MP-1/2 (Early 1990s)

CM-2 はそれなりに成功だった

TMC CM-2



2048 個の Weitek 浮動小数点演算チップセットを SIMD で動かす

TMC CM-2

- 64k 個の 1-bit プロセッサ。メモリ量 64k ビット
- 2048 個の浮動小数点演算器。1 つが 32 プロセッサにシェアされる
- 12 次元ハイパーキューブネットワーク (16 プロセッサが 1 チップ)

現代 (というかちょっと未来) の半導体技術なら、 4-8 台くらいの CM-2 を 1 チップにいれて、100 倍のクロックで動作させ、 10-20 Tflops くらいを消費電力 100W 以下で実現可能

CM-2 ソフトウェア

- *Lisp: データ並列 Lisp
- C*: C++ で実装したデータ並列 C
- CM-Fortran (ほぼ HPF)

もちろん「天才」 Guy Steele がいたからできた話

私のハードディスクから発掘された C* コードの残骸

```
typedef domain node_domain {
    EXTERN int index;              /* index for particle/node */
    EXTERN REAL mass;              /* mass of the node */
    EXTERN REAL position[NDIM]; /* position */
    EXTERN REAL velocity[NDIM]; /* velocity */
    EXTERN REAL acc[NDIM];         /* accelleration */
    EXTERN REAL acc_old[NDIM];  /* accelleration of previous ste
    EXTERN REAL potential;         /* current potential of the part
    EXTERN REAL out_potential;
    EXTERN REAL r_work[NDIM+1];
} NODE_DOM, *NODE_PTR;
```

C* コードの残骸の続き

```
void node_domain::calc_accel()
{
  int myindex;
  mono int i,k;
  REAL dx[NDIM];
  acc[0] = acc[1] = acc[2] = 0.0;
  potential = 0.0;
  myindex = (int)this - (int)&node[0];
  if(this && this < &node[nbody]){
    for(i = 0; i < nbody; i++){
      REAL rsq;
      REAL pot, rsqinv, rinv;
      if(myindex != i){
        rsq = eps2;
        for (k = 0; k < NDIM; k++){
          dx[k]=node[i].position[k]
                -position[k];
          rsq += dx[k]*dx[k];
        }
        rsqinv = 1.0/rsq;
        rinv = sqrt(rsqinv);
        pot=node[i].mass *rinv;
        potential+=pot;
        pot *=rsqinv;
        for (k=0; k<NDIM; k++) {
          acc[k]+=pot*dx[k];
        }
      }
    }
  }
}
```

いいとこ

- 「データ並列」を表現。通信/プロセッサ内データの切り分け、データレイアウトその他はコンパイラ+ランタイムが面倒みてくれる
- 通信は単に配列への間接アクセスで書ける
- 実際に相当複雑な処理が書ける。Barnes-Hut ツリー:構築、粒子毎にバラバラにツリー探索、といったことも。

汎用コアの SIMD との違い: メモリが独立、独立な範囲内ではランダムアクセスできる。ベクタプロセッサ的で、ストライドでもインダイレクトでも性能あまり落ちない。

実際には

コンパイラがバグばっかりだったのでこんなの書く羽目に

```
for(k=0; k<4; k++){
    host_work[k] = CM_u_read_from_processor_1L(&node[i],
            &r_work[k], REAL_LEN);
    CM_u_move_constant_1L(&r_work2[k],host_work[k], REAL_LEN);
}
rsq = eps2;
for (k = 0; k < NDIM; k++){
   CM_f_subtract_3_1L(&dx[k],&r_work2[k],&position[k],
                    SIG_LEN,EXP_LEN);
   CM_f_mult_add_1L(&rsq,&dx[k],&dx[k],&rsq,
            SIG_LEN,EXP_LEN);
}
CM_f_divinto_constant_3_1L(&rsqinv,&rsq,1.0,
            SIG_LEN,EXP_LEN);
CM_f_sqrt(&rinv, &rsqinv, SIG_LEN, EXP_LEN);
CM_move(&pot, &r_work2[NDIM], REAL_LEN);
CM_f_multiply(&pot, &rinv, SIG_LEN, EXP_LEN);
```

```
    CM_f_add(&potential, &pot, SIG_LEN, EXP_LEN);
    CM_f_multiply(&pot, &rsqinv, SIG_LEN, EXP_LEN);
    for (k=0; k<NDIM; k++) {
       CM_f_mult_add_1L(&acc[k],&dx[k],&pot,&acc[k],
                SIG_LEN,EXP_LEN);
    }
}
```

非構造格子?

- 非構造疎行列だって書くのは難しくない。全節点でデータ並列動作。ポインタでアクセスすればPE間通信でデータ取ってくる。
- 性能は実は結構でる: 実はほとんどのアクセスは PE メモリ内ですむ。(ように節点番号ふって欲しいな) メモリバンド幅は高い (しアクセス粒度も小さい) から。

汎用スカラー並列に比べてどう改善？

- 消費電力:
  - データ移動が減る。キャッシュ階層がなく、チップ外メモリも (第一義的には) ない
  - 命令フェッチ・デコードのコストも減っている。1 チップに 1 ユニット。
- 通信オーバーヘッド
  - データ移動が減る。外部メモリ、キャッシュ階層がない
  - ハンドシェイク、同期のオーバーヘッドも減る。SIMD で動いている範囲では始めから同期しているので同期操作不要。細粒度通信が可能。
  - チップ間はもうちょっと考える必要がある。データフローマシン的動作をさせたい

SIMD超並列機のネットワーク

ポスト「京」の FS で一応「検討中」以下は検討中の1例

- チップ内: 16-64コア程度をバス結合したものが基本ユニット。ユニット間は隣接 (4D) メッシュ+ファットツリー
- チップ間: 4D トーラス程度のネットワーク。
- このネットワークでつなぐのは 2048-4096 チップ程度。
- チップ間リンクは 10-20GB/s 程度

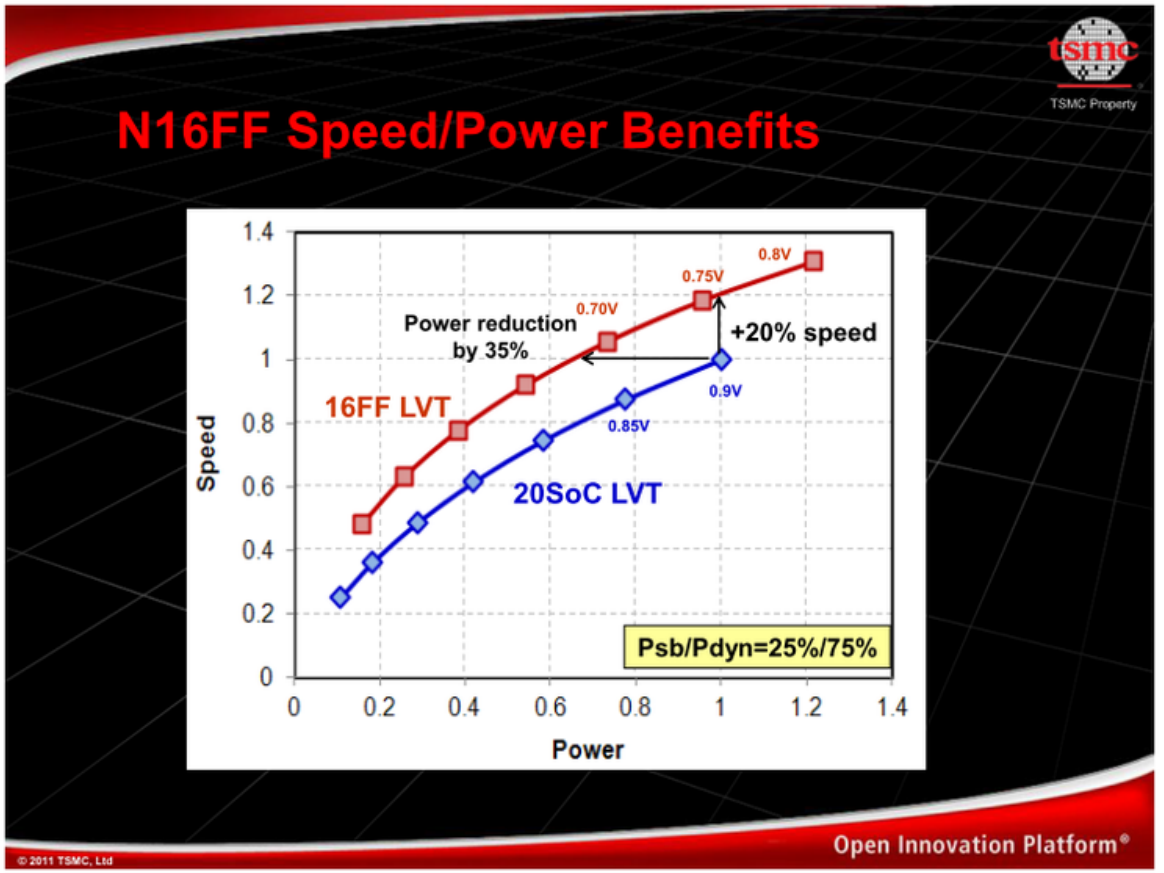
性能皮算用

GRAPE-DR に比べてどこまでいけるか、という観点

	GRAPE-DR	ポスト「京」加速部
テクノロジー	90nm	10nm
DP 乗算器の数	256	16384?
クロック	500MHz	500MHz
演算性能	256GF	16TF
消費電力	60W	120W
動作電圧	1V	0.75V
メモリ量	1MB	512MB にしたいが、、、

- 牧野の計算なのでちょっと楽観的

低電圧動作(TSMC側の主張)



低電圧動作(TSMC側の主張)

- 16nm 0.75V に比べて 0.5V だと電力 1/6、性能 4 割。電力あたり性能は 2 倍以上。
- これ使わない手はない、、、

メモリ量

- 2048 チップシステムでも、オンチップメモリだけでは1TBしかない
- 「アクセラレータ」として、行列乗算や粒子間相互作用計算等の、アプリケーションの一部だけを実行するならこれでもOK
- 流体計算等、アプリケーション全体が加速部で動作する場合には不足
- 不足を補うため、そこそこ高速 (1TB/s 程度) の DRAM をつける。

スケジュール

- ポスト「京」本番機は2019-20 あたり完成。ちょっと将来過ぎる
- 加速部は色々よくわからない部分が (まあ少なくとも外からみて) ある。
- プロトタイプシステムを 2015 年度サンプルチップ、2016 年度システム完成を目標に開発する。これは 10nm ではなく 16nm ベース (性能が全体に半分程度)
- チップ 8TF、1024 チップシステムで 8PF、メモリはオンチップでは、、、128-256GB。
- オフチップメモリは 8TB 程度

この辺でできるサイエンスを一緒に考えていきませんか？

QCD 性能見積もり

- KEK の松古さん他と 2012 年度にやったもの
- 計算機想定: 8192 コア、単精度ピーク 16TF、チップ間 4D トーラス、リンク 10GB/s
- 1PE8 格子点 (16kB メモリ)
- クローバークォーク: チップに $16^3 \times 32$, 7.3TF/チップ
- ドメインウォールクォーク: チップに $8^4 \times 32$, 2.3TF/チップ

どちらも通信リミット。チップ間通信減らすようなアルゴリズムがもうちょっと頑張れるとするとまあまあの性能。

メモリ増えるともうちょっと通信は楽。

他に検討中のアプリケーション

- 古典分子動力学
- 規則格子での流体等 (宇宙 MHD、地震、……)
- 宇宙 N 体

公式に検討していることになっているもの

理研内の検討

- **古典** MD、O(N) **量子化学** (CONQUEDT)、 QCD(**筑波**)、**地震**、**気象**、

つくば FS **での検討**

- QCD, MD, **宇宙** MHD, **地震**, **量子物性** (RSDFT)

まとめ

- 現代の大規模スカラー並列機は「小さい」問題にむかない (現在の「小さい」＝自由度 10^7、2020 年だと 10^9)
- この問題の解決の一つの方向: オンチップ SIMD 超並列
  - オンチップメモリには高いバンド幅
  - 超低レイテンシ通信
  - 低消費電力
- SIMD 超並列による加速部が今のところポスト「京」プロジェクトにはいっている。
- 加速部は、QCD はできるように作りたい、あと、古典 MD 等粒子系、、差分法流体の、比較的小規模な問題を高速に解くことを目標にする。
- 使うと面白いサイエンスができると思うので、一緒に考えていきませんか？