これで我々のアプリケーションプログラムは速くなるか？
—マルチコアクラスタ性能WG成果報告—

マルチコアクラスタ性能WG まとめ役　高木亮治
宇宙航空研究開発機構

内容

- WGの概要
- アプリケーションチューニングの実践例
  - 会員のいくつかの例より
- 活動の総括
  - かなり個人的
- まとめ

WG概要

WG設置の背景

- WG立ち上げ：2010年
- FX1から次世代スーパーコンピュータ「京」（、FX10）への流れ
  - ノード内マルチコアの大規模クラスタ
    - FX1：4コア、「京」：8コア、FX10：16コア
  - 「京」の本格稼働：2012年
  - 「京」の利用促進に向けたプログラムの性能評価と高速化手法の検討
- 「京」を始めとしたマルチコアクラスタマシンに向けた、
  - 並列プログラミングモデル
  - 性能評価ツールの利用法、分析手法
  - 高速化チューニング

  に関するノウハウの共有を目指した。

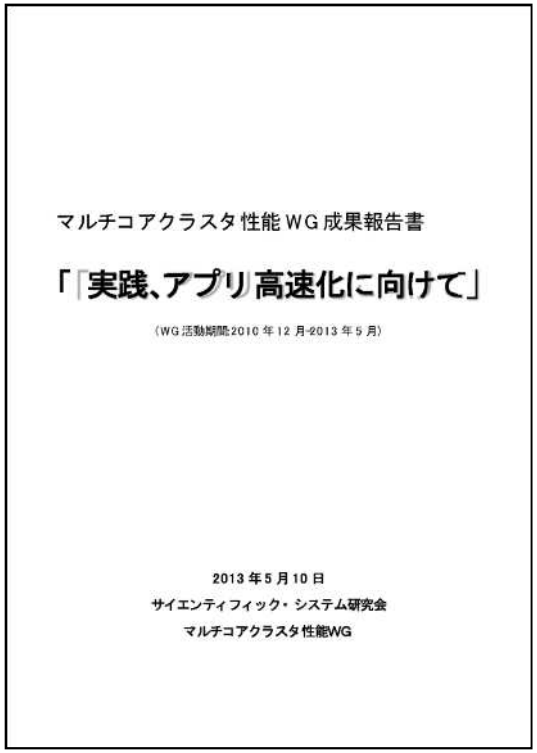
活動期間とメンバー

- **活動期間**：2010年12月～2013年5月 (2.5年)
- **メンバー**：

	氏名	所属		氏名	所属
担当幹事	石井 克哉	名古屋大学	推進委員	福島 正雄	富士通 (株)
推進委員	高木 亮治	宇宙航空研究開発機構		青木 正樹	富士通 (株)
	井戸村 泰宏	日本原子力研究開発機構		山中 栄次	富士通 (株)
	梅田 隆行	名古屋大学		三吉 郁夫	富士通 (株)
	荻野 正雄	名古屋大学		三輪 英樹	富士通 (株)
	坂下 雅秀	宇宙航空研究開発機構		内藤 俊也	富士通 (株)
	佐藤 幸紀	北陸先端科学技術大学院大学		錦 龍生	富士通 (株)
	柴村 英智	九州先端科学技術研究所		瀧 康太郎	富士通 (株)
	野田 茂穂	理化学研究所		千葉 修一	富士通 (株)
	姫野 龍太郎	理化学研究所	オブザーバー	森重 博司	富士通 (株)
	堀之内 成明	(株) 豊田中央研究所	オブザーバー	市川 真一	富士通 (株)
	南 一生	理化学研究所			

活動内容の概略

- 全10回の会合 (2.5年間)
- 活動内容
  - 情報提供
    - 次世代スーパーコンピュータ「京」
    - 性能解析ツール：PAツール、会員ツール
    - 各種チュートリアル
  - 会員アプリの測定報告
    - 性能測定、チューニング
- 成果
  - 成果報告書 (約180ページ)
  - PRIMEHPC FX10チューニングチュートリアル (約300ページ)


マルチコアクラスタ性能WG成果報告書

「「実践、アプリ」高速化に向けて」

(WG活動期間:2010年12月-2013年5月)

2013年5月10日

サイエンティフィック・システム研究会

マルチコアクラスタ性能WG

成果報告書 (1/2)

- **アプリケーションの測定評価 (会員から)**
  - **3次元**FEM**構造解析コード**：ADVENTURE
  - **3次元非圧縮性流体計算プログラム**：COSMOS
  - **3次元電磁界コード**：FDTD3
  - **核融合プラズマ5次元格子コード**：GT5D
  - **圧縮性流体解析プログラム**：UPACS
  - **宇宙プラズマ5次元ブラソフコード**：Vlasov5
  - **流体構造連成解析アプリ**：ZZ-EFSI
  - **超音波集束シミュレータ**：ZZ-HIFU
  - **非構造格子**CFD**ソルバ**：JTAS

成果報告書 (2/2)

- 性能評価ツール (会員から)
  - インターコネクトシミュレータ：NSIM
  - 実行駆動型アプリ解析ツール：Exana
- 共通事項 (富士通から)
  - H/WとS/Wプリフェッチの仕様
  - キャッシュミス数/ミス率
  - FMA命令化

別冊 PRIMEHPC FX10 チューニングチュートリアル

- 第1章 プログラミング言語処理系概略
- 第2章 PAイベント偏
- 第3章 Fortran偏
- 第4章 C/C++偏
- 第5章 チューニングツール偏
- 第6章 ノード内チューニング偏
- 第7章 MPIおよびノード間チューニング偏

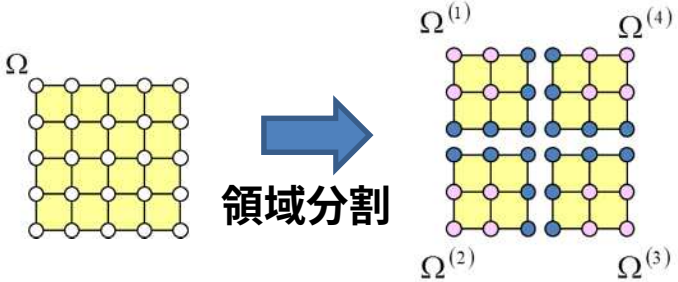
会員のいくつかの例より

アプリケーションチューニングの実践例

ADVENTURE (荻野、名大)

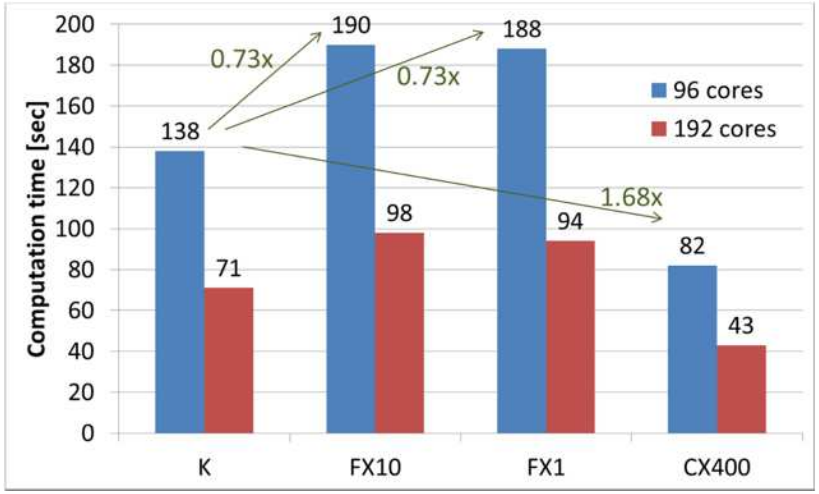
- FEMによる弾塑性解析
  - 非構造格子、疎行列の反転
- FX1, FX10, CX400, 京で評価
  - メモリバンド幅ネック
  - 直接法→反復法でメモリバンド幅ネックを緩和
  - 現時点では直接法が速いが、コア数が増え、コア当たりのメモリバンド幅が低下した場合は反復法が有利

3次元FEM構造解析コードADVENTURE

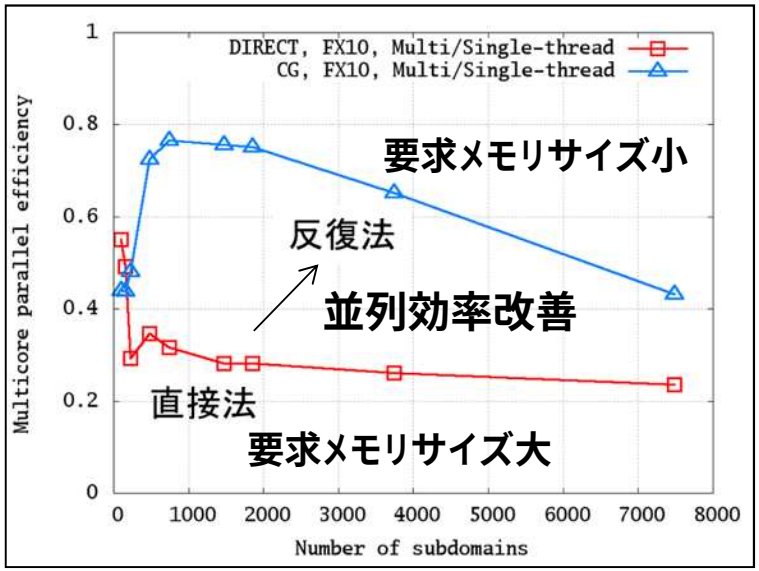



```
#pragma omp parallel for schedule(dynamic,1)
for i=1,...,Nj        領域方向ループ
  s(i)n = -A(i)IB R(i)B pn
  t(i)n = A(i)-1II s(i)n
  q(i)n = A(i)BB R(i)B pn + A(i)TIB t(i)n      領域FEM
#pragma omp critical
  qnj = qnj + R(i)TB q(i)n
endfor
qn = qn + qnj
```

領域分割法における高コスト部分



同一コア数における性能比較



マルチコアCPU内の並列効率

- 領域分割法は，粗粒度の並列性を持つ
- 一方，各スレッドが高いB/Fを要求する
- 同一コア数で比較するとシステムのB/F値に従った性能差が見られた
- 領域FEMの要求メモリサイズが小さくなる実装を行い，CPU内並列効率の改善が得られた

COSMOS (堀之内、豊田中研)

- **非定常非圧縮性乱流計算プログラム**
  - LES, **構造格子 (物体適合、重合) , 陰解法 (行列反転**, SOR)
- RX600, FX1, FX10, **京で評価**
  - **マルチカラー化 (8色) 、コンパイラオプション**
  - **反復解法レベルでのアルゴリズムの見直し**
  - **マルチコアに特化した配列構造の利用？**

チューニング対象とした計算の概要

- LES(*)による非圧縮性流れの計算
  - 基礎方程式: 連続の式，運動方程式（Navier-Stokes 方程式）
    ↓ SMAC法に準じた陰解法（時間積分: Crank-Nicolson法）
    運動方程式，圧力Poisson方程式に対する連立一次方程式に帰着
    （計算時間全体の8割程度を費やす）
  - 格子体系: 構造格子を組み合わせた重合格子
    ↓ 各部分格子ごとに離散化
    係数行列: 規則的（非対称）スパース行列
- 連立一次方程式 $Ax=b$ の解法: SOR(**)法
  - $A=L+D+U$ とした時，下三角行列 L にかかる要素は常に最新の値を参照；
    $(D+\omega L)\boldsymbol{x}^{(n+1)}=\{(1-\omega)D-\omega U\}\boldsymbol{x}^{(n)}+\omega\boldsymbol{b}$
    → 回帰参照が発生
  - マルチカラーオーダリングによる並列化
    ⇒ これの高速化がターゲット

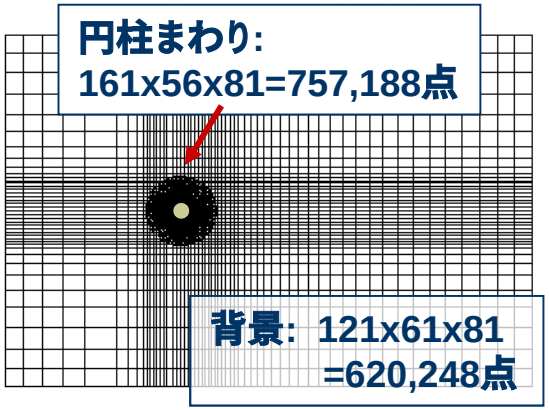


図1 評価例題（円柱周りの流れ）用重合格子

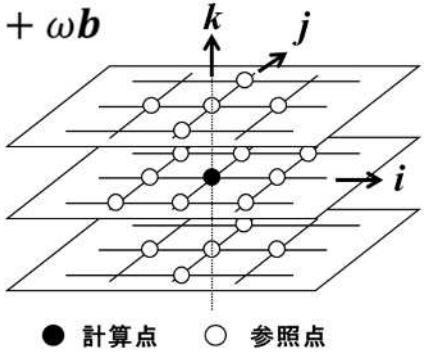


図2 格子点の参照関係

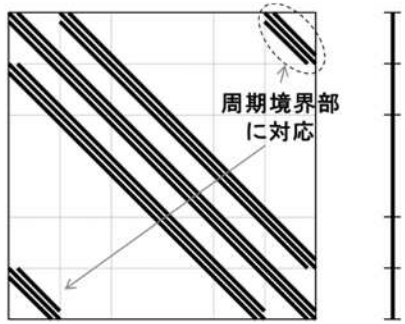


図3 係数行列のイメージ

(*) Large Eddy Simulation
(**) Successive Over-Relaxation

チューニング結果のまとめ

- **オーダリングの修正による高速化** (*on FX1 4core*)
  - オリジナル：3次元の格子点を1次元化した配列に入れて，最小の色数となる7色でオーダリング → 1次元ループのストライドアクセス
  - 改良版: 3次元 (i,j,k) 各方向ごとに2色化した8色でオーダリングし，かつ，各色ごとのループに分ける．(配列のとり方は変えていない)
    ⇨ プログラム全体で8%の実効速度向上 (ｷｬｯｼｭｱｸｾｽ待ち削減)
- **コンパイルオプションによるチューニング**
  (上記改良版に対して，*on FX10 16core*)
  - 圧力Poisson方程式から得られる連立一次方程式の計算:
    ソフトウェアパイプラインによる命令スケジューリング
    ⇨ 該当ルーチンで11%の実効速度向上（浮動小数点演算待ち，整数演算待ち削減 ⇒ ﾒﾓﾘｽﾙｰﾌﾟｯﾄ改善)
  - 運動方程式から得られる連立一次方程式の計算:
    ソフトウェアプリフェッチと，ストライドアクセスオプション指定
    ⇨ 該当ルーチンで11%の実効速度向上
    (浮動小数点ﾛｰﾄﾞﾒﾓﾘｱｸｾｽ待ち，L2ﾐｽﾃﾞﾏﾝﾄﾞ率削減
    ⇒ ﾒﾓﾘｽﾙｰﾌﾟｯﾄ改善)

FDTD3 (梅田、名大)

- 3次元の電磁場解析
  - 構造格子
- FX1, FX10, 京でノード性能の評価
  - メモリバンド幅ネック
    - 配列インデックスの違い (ベクトル型、スカラー型)
    - キャッシュの再利用具合はアルゴリズムに依存
  - ループ分割か融合か？
  - 配列の融合の是非？

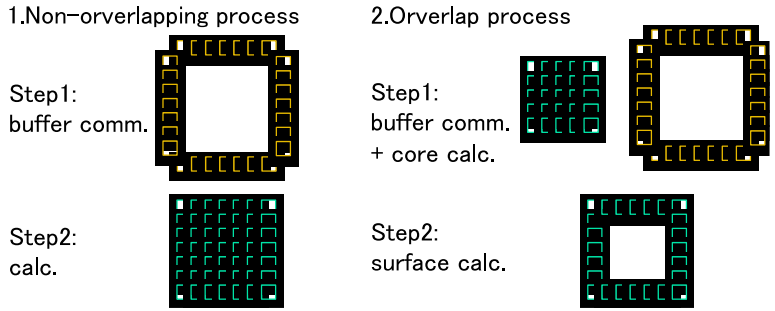
A(i,j,k,n) or
A(n,i,j,k) or
A1(i,j,k), A2(i,j,k),…, An(i,j,k)

GT5D (井戸村、JAEA)

- 第一原理プラズマ乱流コード
  - 5次元位相空間 (3次元空間×2次元速度空間)
    - 3次元流体に比べて100^2倍自由度が大
- BX900, Helios, 京, FX1, FX10で評価
  - バンド幅ネック
  - ルーフラインモデルによる性能予測と実測の比較
  - 通信マスク手法の適用
    - 袖通信：15%削減 (京)、41%削減 (Helios)
    - 60万コアまで99.99989%の並列化効率を達成 (24,576コアから589,824コアまでのストロングスケーリングでの評価)
- 課題：大規模並列I/O、可視化

核融合プラズマ5次元格子コードGT5Dの測定評価

通信と演算の同時処理によるオーバーヘッド削減



- **概要**

核融合プラズマ5次元格子コードGT5Dの並列化率向上を目的として、演算と通信を同時処理する通信マスク手法を開発し、10万コア以上のストロングスケーリングを実現

- **通信マスク手法**

①MPIライブラリ（富士通、インテル）における*RendezVouz*プロトコルの問題により演算中に非同期通信が機能しない原因を解明

②この問題を回避する2つの手法を開発

B.MPI_Testによる*RendezVouz*プロトコル促進

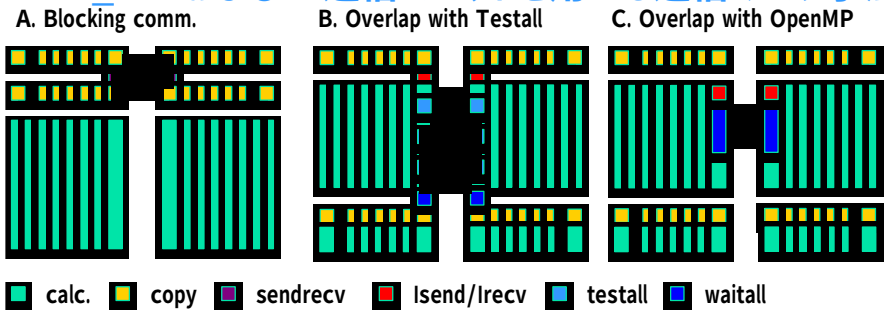
C.OpenMPによる通信スレッドの実装

③手法B、CをGT5Dにおける差分演算の袖領域通信、さらに、手法Cをデータ転置の集団通信に適用し有効性を確認

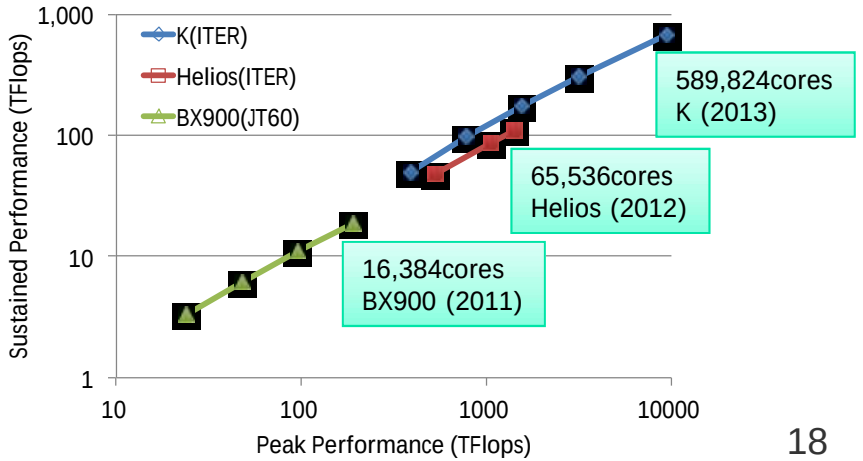
→並列化率～99.9999%を達成し、京60万コアを用いてBX900の約35倍の高速計算を実現

[Idomura et al., Int. J. HPC Appl. 2013]

B.MPI_TestおよびC.通信スレッドを用いる通信マスク手法



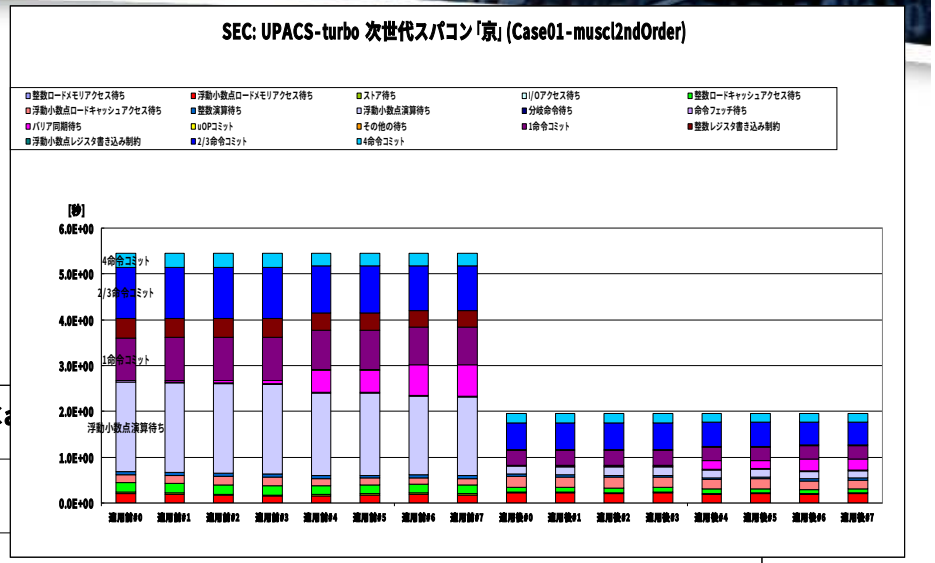
京およびHeliosにおけるGT5Dのストロングスケーリング

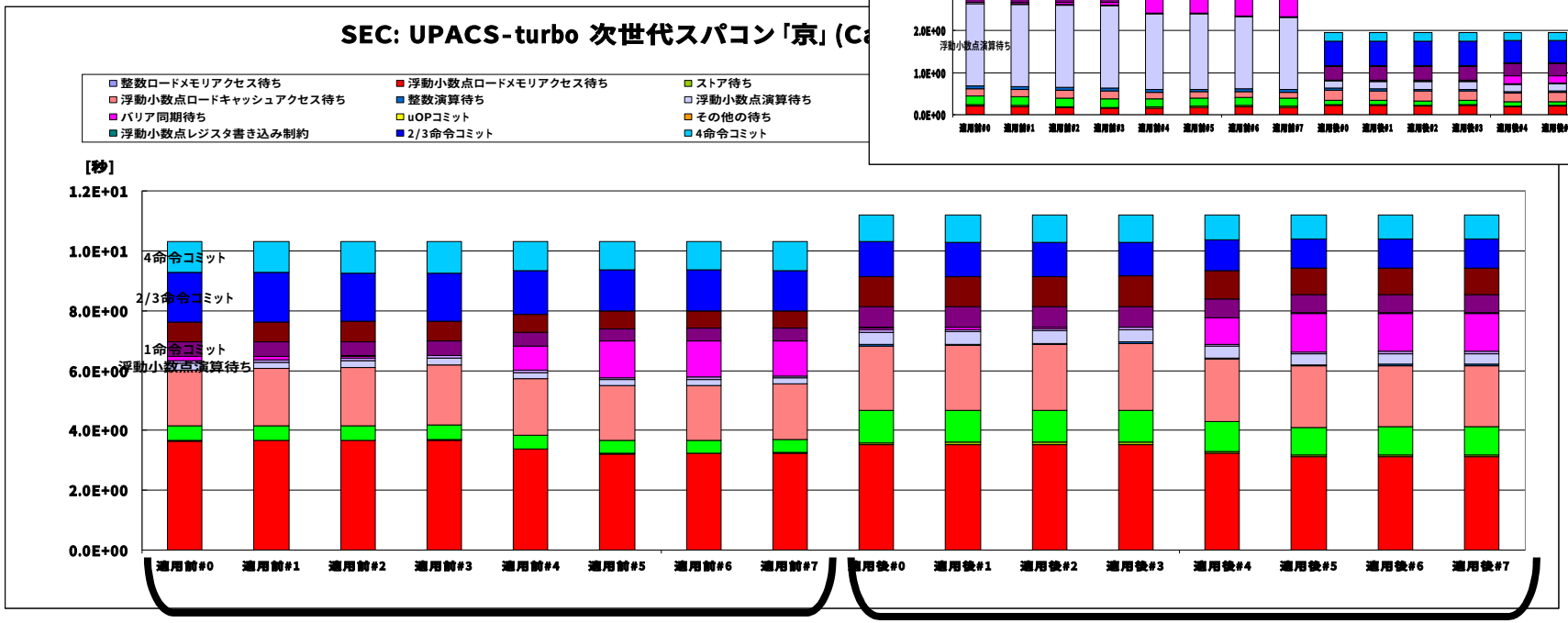


UPACS (高木、JAXA)

- 3次元圧縮性流体解析プログラム
  - 構造格子 (マルチブロック, 重合)
- FX1, 京で評価
  - スレッド並列の促進、キャッシュチューニング、SIMD化
    - Allocate/deallocate：アリーナ開放の抑止
  - SIMD化を促進してもメモリバンド幅ネックの場合は速度向上なし
    - チューニングの指針として何を見るべきか？

- SIMD化率とFLOPSの関係
  - SIMD化を促進しても性能が向上せず。





SIMD化促進前

SIMD化促進後

PAデータ 測定区間	実行時間 (sec)	浮動小数点 演算ピーク比	MFLOPS	MIPS	浮動小数点 演算数	SIMD演算命令率 (/対象演算命令数)
適用前#0	10.31	9.45%	1512	1929	1.56E+10	0.00%
適用後#0	11.21	8.69%	1391	1707	1.56E+10	98.85%

UPACSカーネル (高木、JAXA)

- UPACSのカーネル部分 (対流項、時間積分: 陽解法)
  - 従来のベクトル型ループ
    - 空間スイープが多い
  - 局所性を意識したループ
- JSS, Intel CPUで評価
  - キャッシュミス率の低減
  - 低B/Fでの性能向上が期待

データ&ループ構造

データ：Q(i,j,k,n), I,j,k：空間、n：物理量

ループA：

空間ループ

```
do dir=1,3

  do k=1,kmax, do j=1,jmax, do i=1,imax
       MUSCLの計算
  enddo, enddo, enddo

  do k=1,kmax, do j=1,jmax, do i=1,imax
       FLUXの計算
  enddo, enddo, enddo

  do k=1,kmax, do j=1,jmax, do i=1,imax
       RHS(ΔQ) の計算
  enddo, enddo, enddo

enddo

do k=1,kmax, do j=1,jmax, do i=1,imax
     時間積分
enddo, enddo, enddo
```

ループB：

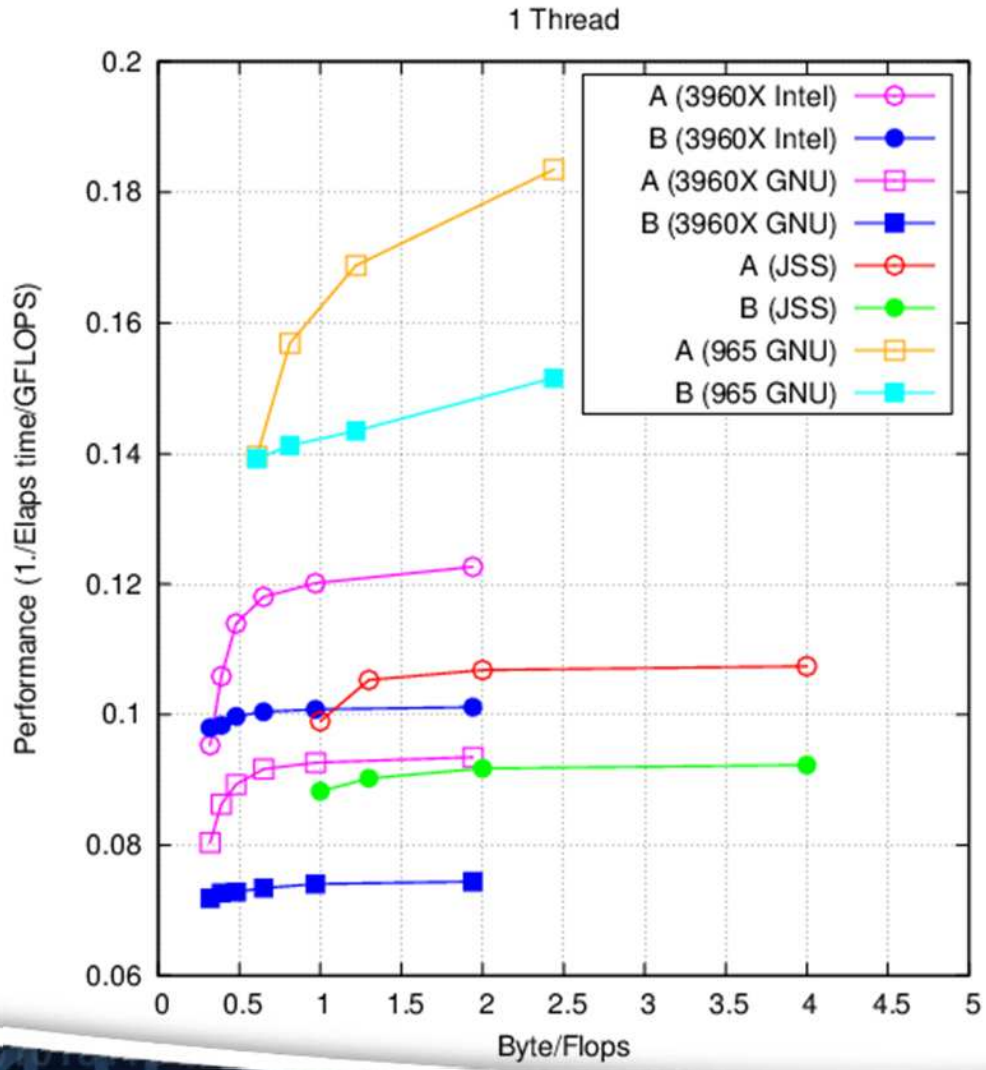
```
do k=1,kmax, do j=1,jmax, i=1,imax

     do ndir=1,3
        MUSCLの計算
        FLUXの計算
        RHS (ΔQ) の計算
        境界での処理 (MUSCL, FLUX, ΔQ)
     enddo

enddo, enddo, enddo

do k=1,kmax, do j=1,jmax, i=1,imax
     時間積分
enddo, enddo, enddo
```

ループAとBの比較 (1スレッド)



- 仮想的にメモリバンド幅を変化させた。
  - スレッド数は1で固定
  - ブロックサイズは80
- 縦軸は理論ピーク性能あたりの性能 (経過時間の逆数)
- Byte/FLOPは理論性能
- B/Fが悪化すると、ループAは急激に性能が悪化する。

ZZ-EFSI (野田、理研)

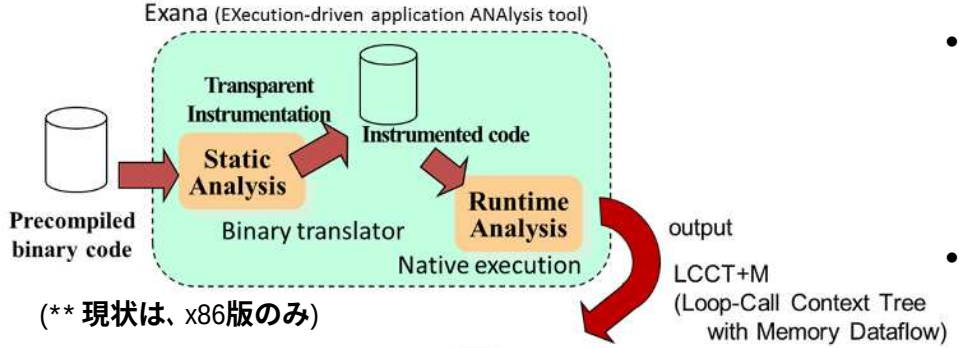
- **流体構造連成解析**
  - **ボクセル格子**, WENO
  - **既存のチューニングではなく新たに設計 (京の性能を出す)**
    - **計算アルゴリズムの選択**
- RICC@**理研**, **京で評価**
  - **高い実行性能**
    - ノード：46.4%、12,288ノード：43.2%
  - **優秀なスタッフの理詰め、でも最後は試行錯誤**

性能評価ツール

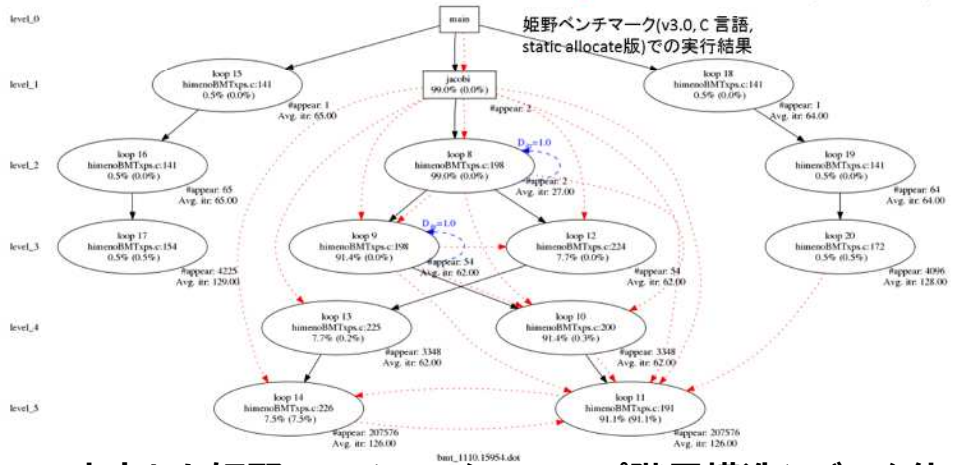
- NSIM (柴村、九州先端)
  - インターコネクトシミュレータ
  - 残念ながらユーザーの利用はなかった。
- Exana (佐藤、北陸先端)
  - プロファイラー
    - ホットスポット、ループ階層構造とそれらの間の並列性の検出
  - 残念ながらユーザーの利用はなかった。
    - チューニングのノウハウや事例に基づき機能要件を検討した。
    - 並列性はループだけでない。
    - キャッシュの挙動の考慮、デバッガとの連携、部分解析、オーバーヘッド

実行駆動型アプリケーション解析ツール　Exana

本ツールは動的バイナリ変換によりコード実行時にループおよびデータフロー情報を抽出



(** 現状は、x86版のみ)

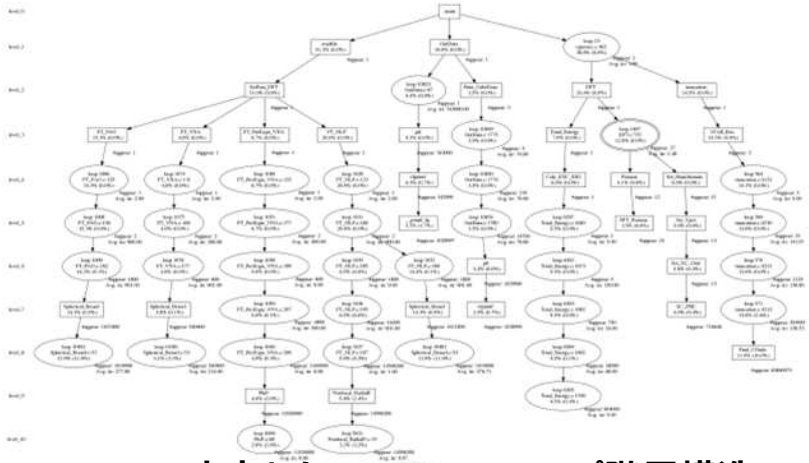


出力した姫野ベンチマークのループ階層構造とデータ依存

丸いノードがループ
四角のノードは関数
実線はコントロールフロー
　親子関係はネストで子ノードは内部ループ
点線はデータ依存

本WGでの議論による知見

- 流体と構造の連成解析のようなマルチフィジクスでは並列性はループだけとは限らないため本手法により関数とループのコンテクストによりコードを俯瞰することは有益
- チューニングへの応用のためには、ループ階層構造をキャッシュの挙動を如何に結びつけるかが鍵



出力したOpenMXのループ階層構造

- データ依存プロファイルをなしとすると解析オーバーヘッドは大幅に小
  - データ依存解析ありで50倍、なしで3倍程度のオーバーヘッド
  - データ依存なしでもプログラムのコードを俯瞰する手段としてはOK
  - ループ階層構造をキャッシュの挙動と結び付ける解析が望まれる
  - 本ツールでキャッシュ性能を推測できるかということは検討が必要

かなり個人的

活動の総括

議論を通じて得られた知見

- メモリバンド幅ネックのアプリケーションが多く、そこのチューニングが主。
- メモリのスループットを上げるために今回はプリフェッチ (PF) に注目
  - 通常はHWPFを使うが、場合によってはSWPFを使った方が良い場合がある。
    - 何時SWPFを使うか？
    - 「今でしょ！！」という簡単明瞭な基準がなくて「ケースバイケース」(←個人的には悪夢の言葉！！)

議論を通じて得られた知見

- HWPFとSWPFの仕組み
- どういう場合はどちらを使うか？
  - ケースバイケースだが、いくつかの事例はまとめた。
  - SWPFを使うとき：
    - 連続アクセスだが、途中でアクセスが飛ぶ
      - 無駄なアクセスをしない
    - 翻訳時オプションと最適化指示子の利用法
      - 最適化指示子が確実
- ベンダーはコンパイラにお任せあれと言うが…。

議論を通じて得られた成果

- チューニング支援機能としてコンパイラへの改善要求
  - FMA命令化のメッセージ等を出力する。
  - 現状の問題点
    - FMA命令化はコーディングスタイルに依存しない → ユーザーは操作できない。
    - FMA命令化したかどうかはアセンブラを見るしかない。

チューニングの現状(かなり個人的)

- やっていること:
  - ある程度見通しをつけたら、まずは試してみる！
  - どんどん試す、ひたすら試す！！！
- 「試行錯誤の世界」
  - 微かにある理詰めも最後は「ケースバイケース」で粉砕される。
- やっぱり一般ユーザーの手に負えないレベル！
  - 専門家にまかせるしかない？
  - 専門家でも試行錯誤！！
    - WGでも何故何故攻撃の繰り返し

チューニングの現状(かなり個人的)

- アーキテクチャが複雑化してチューニングが大変
  - ベクトル型CPUは単純で使いやすかった。
- 何が問題か？
  - B/Fを要求するアプリケーションに対して計算機のB/Fが低下 → アプリケーション側で工夫が必要。
  - 性能評価データの分析も困難。
  - チューニングも含めて、プログラムをどう書けば良いかわからない。
    - 私だけかもしれないが、どうもそうでもない…。
- 何故そうなったか？
  - アーキテクチャ(H/W, コンパイラ)と実際の動きを正確に理解できていない。

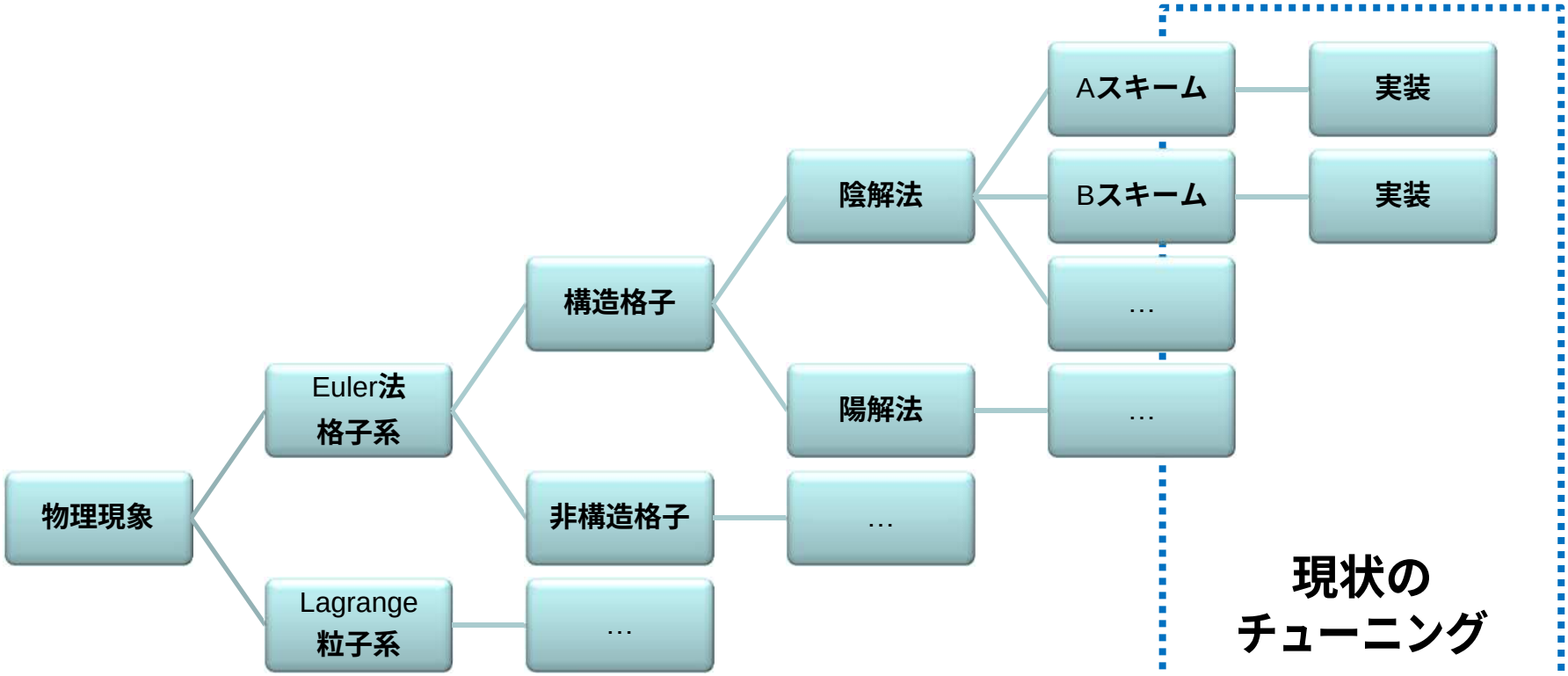
チューニングの現状(かなり個人的)

- ユーザーはH/W、コンパイラについてどんだけ勉強すれば良いのでしょうか？
  - 「試行錯誤」、「ケースバイケース」では先に進めない！！
- そもそもこんなスーパーな人はいない？
  - ユーザーの何故何故攻撃に答えられる。
  - 理詰めで分析･チューニングができる。
  - ケースバイケースの判断基準が明確。
  - 試行錯誤しなくてもどっちが良いか判断できる。

WGの限界

- 個別アプリケーションのチューニングに終始
  - 「試行錯誤」と「ケースバイケース」
- メタな情報としては汎用化、共有化できるが、具体化したとたんに個別ケースになりがち。
- もっとメタと個別をつなぐ部分が必要。
  - ある意味Co-Designだと思うが、どう実現すれば良いか？

Co-Design (ex. **流体解析**)



Co-Designではどこまで上流に行けるか？
上流に行けば行くほど共有化、汎用化が可能では？

まとめ

- マルチコアクラスタ性能WGでは、FX1,「京」,FX10などのマルチコアクラスタを対象としたアプリケーションプログラムの性能評価および高速化チューニングを実施した。
  - 報告書、FX10向けチューニングチュートリアル
- 今後のエクサスケール計算機開発に向けて、H/Wおよびシステム開発者、コンパイラ開発者の支援の下、ユーザーが主体的に性能評価・高速化チューニングを実施する本WG的活動が継続されることを期待する。

まとめ

- これで我々のアプリケーションプログラムは速くなる（なった）か？
- 回答：（少なくともWGに参加すれば）速くなった。
  - WGでは専門家（H/W、コンパイラ）に分析をしてもらえる。
    - 進むべき道も教えてもらえる？
  - 先達ユーザーからも教えてもらえる。
  - 継続することが必要。
- 皆さんWGに参加しましょう！！

今後

- チューニングも必要だがCo-Designを目指した活動が必要！！
  - WGでやれるか？（SS研でできるか？）
  - もう少し柔軟な活動形態が適切かも。
- SS研ならではのWG活動が求められている。
  - HPCI, HPCIコンソーシアム
  - 京、AICS、…

謝辞

- 2.5年間のWG活動に参加していただいた会員の皆様、富士通担当者様、SS研事務局の皆様に感謝の意を表します。
- WGでは以下の環境を利用させていただきました。改めてここに感謝の意を表します。
  - スーパーコンピュータ「京」試験利用枠、一般公募枠
  - 宇宙航空研究開発機構 JSS
  - 日本原子力研究開発機構 BX900, FX1
  - 国際核融合エネルギー研究センター Helios
  - 名古屋大学 FX1
  - 東京大学 FX10
  - 九州大学 CX400, FX10
  - 理化学研究所 RICC
  - 富士通社内機