東大FSアプリWG
進捗報告

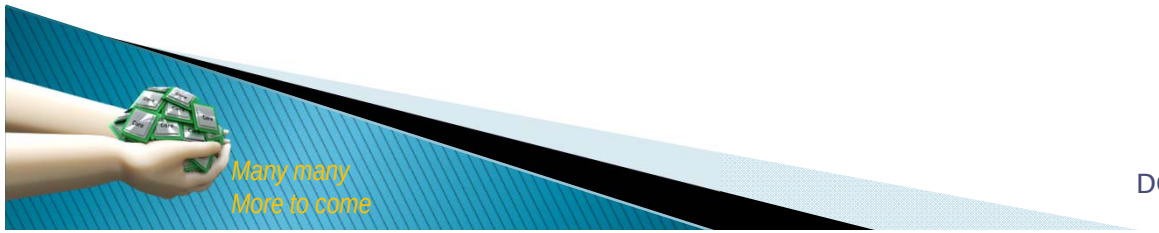
片桐 孝洋，大島 聡史，中島 研吾（東大）
米村 崇，熊洞 宏樹，樋口 清隆，橋本 昌人，高山 恒一（日立情報・通信システム社），
藤堂 眞治，岩田 潤一，内田 和之，佐藤正樹，羽角博康（東大），
黒木聖夫（海洋研究開発機構）

将来HPCIあり方調査研究「アプリ分野」第6回全体ミーティング・アジェンダ
日時：2013年5月31日（金）、場所：TKP東京駅ビジネスセンター1号館（注1） 3階 ホール3A
10:10−11:00　東北・筑波・東大FSチームとの連携状況 (50分)

Feasibility Study on
Advanced and Efficient Latency Core-based Architecture for Future HPCI R&D

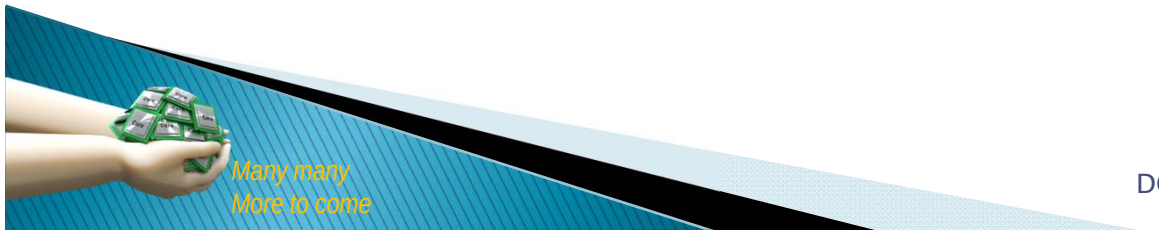
Agenda

- プロジェクト概要
- FX10での性能チューニング実例
- 異機種環境での評価
- 今後の予定



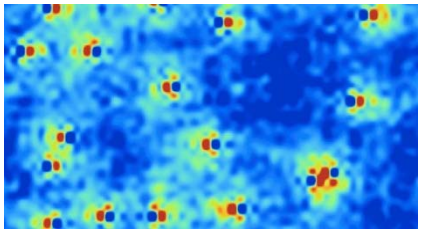
Agenda

- プロジェクト概要
- FX10での性能チューニング実例
- 異機種環境での評価
- 今後の予定

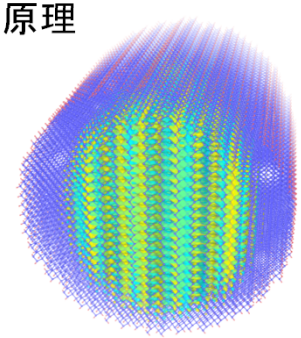


ターゲットアプリケーション群

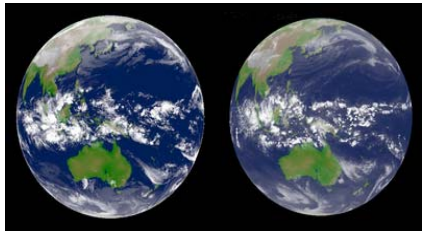
- ALPS/looper
  - 新機能を持った強相関・磁性材料の物性予測・解明。虚時間経路積分にもとづく量子モンテカルロ法と厳密対角化
  - 総メモリ：10～100PB
  - 整数演算、低レイテンシ、高次元のネットワーク
  - 利用シナリオ：1ジョブ当たり24時間、生成ファイル：10GB. 同時実行1000ジョブ、 合計生成ファイル：10TB.



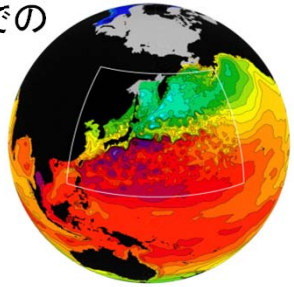
- RSDFT
  - Siナノワイヤ等、次世代デバイスの根幹材料の量子力学的第一原理シミュレーション。実空間差分法
  - 総メモリ：1PB
  - 演算性能：1EFLOPS（B/F =0.1以上）
  - 利用シナリオ：1ジョブ当たり10時間、生成ファイル：500TB. 同時実行10ジョブ、合計生成ファイル5 PB.



- NICAM
  - 長期天気予報の実現、温暖化時の台風・豪雨等の予測
  - 正20面体分割格子非静力学大気モデル。水平格子数kmで全球を覆い、積雲群の挙動までを直接シミュレーション
  - 総メモリ: 1PB、メモリ帯域: 300 PB/sec
  - 演算性能: 100 PFLOPS （B/F = 3）
  - 利用シナリオ：1ジョブ当たり240時間、生成ファイル：8PB. 同時実行10ジョブ、合計生成ファイル：80 PB.



- COCO
  - 海況変動予測、水産環境予測
  - 外洋から沿岸域までの海洋現象を高精度に再現し、気候変動下での海洋変動を詳細にシミュレーション
  - 総メモリ: 320 TB、メモリ帯域: 150 PB/sec
  - 演算性能: 50 PFLOPS（B/F = 3）
  - 利用シナリオ：1ジョブ当たり720時間、生成ファイル：10TB. 同時実行100ジョブ、合計生成ファイル：1 PB.



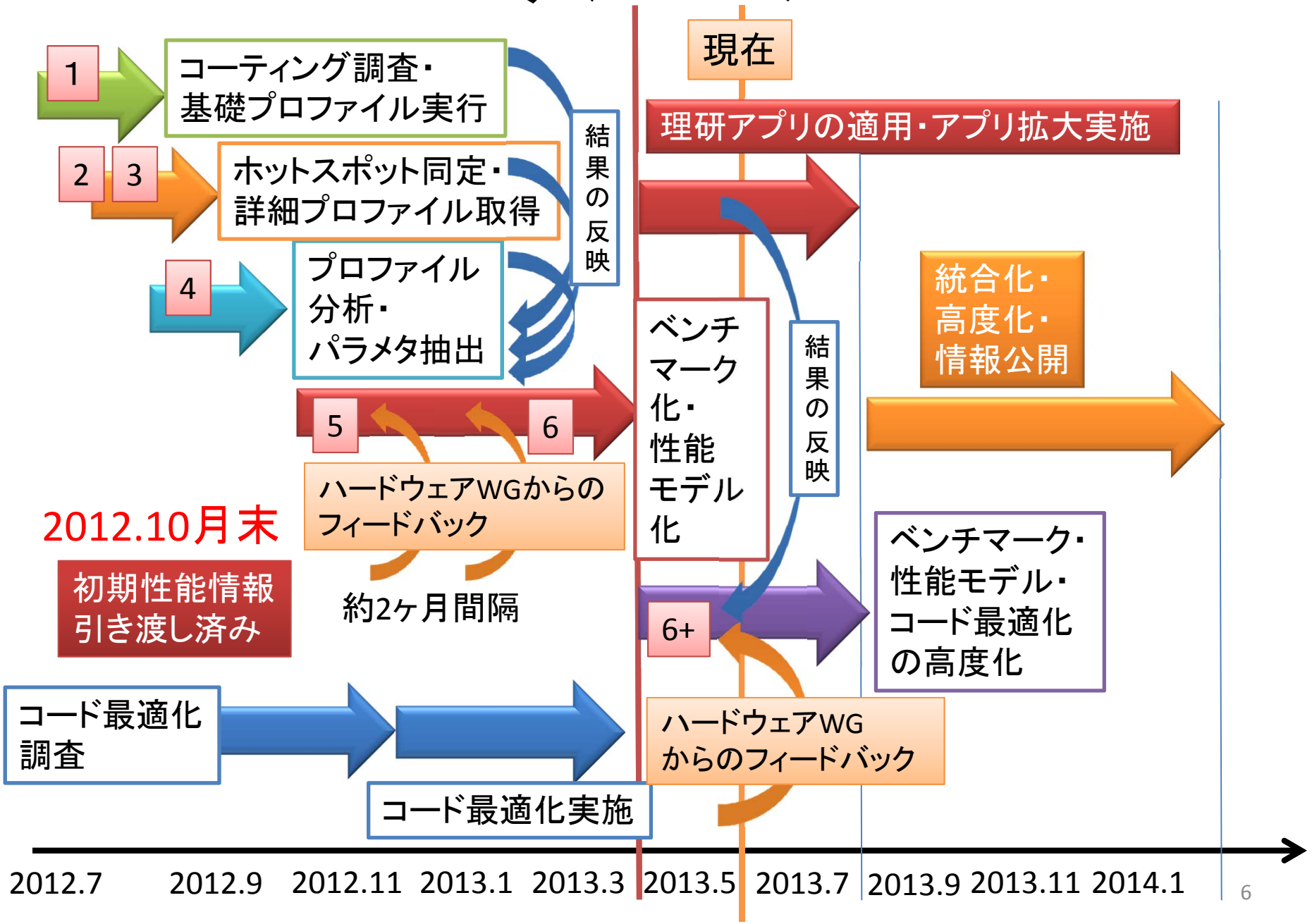
利用シナリオ
アンサンブル型
全系の1/10～1/100資源を用いた1ジョブを、複数同時実行することで、全資源を使い切る形態。

要求性能は「計算科学ロードマップ白書」(2012年3月)の見積値からの抜粋、および、開発者による新見積値である

性能モデル化手法

1. **ホットスポット同定**：**富士通社の**基本プロファイラで複数の**ホットスポット**（ループレベル）を同定、全体性能の予測をホットスポットのみで行う
   - ホットスポットの部品化
   - 数理レベル（支配方程式、離散化方法）の処理ブロックとの対応を検討
2. **カーネル分離**：（目視により）計算部分、通信部分、I/O部分の分離
   - 計算部分：演算カーネル
   - 通信部分：通信カーネル
   - I/O部分：I/Oカーネル
3. **通信パターン確認**
4. **詳細プロファイルと分析**：　**富士通社の**詳細プロファイラを用い、ホットスポットごとにハードウェア性能情報（＝性能パラメタ）を取得し分析
   - 演算カーネルの　演算効率／命令発行量／キャッシュ利用効率　など
   - 通信カーネルの　通信回数／量／通信待ち時間　など
   - I/Oカーネルの　データ読み書き　量／頻度　など
5. **ベンチマーク化**：ホットスポットのみで動作するようにコードを再構成
   - マシン特化の書き方、および、汎用的な書き方、の2種を区別
   - 演算カーネル、通信カーネル、I/Oカーネルの分類
6. **詳細モデル化と予測**：ハードウェア因子を用いた数式による実行時間を近似。
   - 富士通社の性能予測ツールにより、概念設計マシンの実行時間を予測

スケジュール


1
コーティング調査・
基礎プロファイル実行
2
3
ホットスポット同定・
詳細プロファイル取得
4
プロファイル
分析・
パラメタ抽出
結果の反映
現在
理研アプリの適用・アプリ拡大実施
統合化・
高度化・
情報公開
ベンチ
マーク
化・
性能
モデル
化
結果の反映
5
6
ハードウェアWGからの
フィードバック
2012.10月末
初期性能情報
引き渡し済み
約2ヶ月間隔
6+
ベンチマーク・
性能モデル・
コード最適化
の高度化
コード最適化
調査
ハードウェアWG
からのフィードバック
コード最適化実施
2012.7
2012.9
2012.11
2013.1
2013.3
2013.5
2013.7
2013.9
2013.11
2014.1

アプリケーション最適化箇所（演算カーネル）

アプリ名	最適化箇所	補足
ALPS/looper	Timer 6 (OMP_44)	Union、Findの処理
	Timer 11 (OMP_48)	
	Timer 12 (OMP_49)	
	Timer 15 (OMP_51)	
RSDFT	diag_2d	密行列対角化部分
	gram_schmidt_sub_blkcyc	直交化部分
NICAM	mod_oprt2	力学過程に関する処理
	mod_oprt3d_4	
	mod_src5	
	mod_src7	
	mod_oprt_8	
	mod_mp_ns_nsw6_9	物理過程に関する処理
COCO	flxtrc._OMP_2	移流項に関する処理
	flxtrc._OMP_3	
	flxtrc._OMP_5	

概念設計マシンでの性能予測手法

- 4アプリ共に、弱スケーリングを用いて予測
  - 利用可能ノード数まで、問題サイズを大きくできる
- 演算時間予測
  - 富士通社のプロファイルツールで演算時間予測
- 通信時間予測
  - 通信パターン、通信回数、通信量を同定し、
    机上評価で通信時間を予測
- 科学的に意味がある問題規模を設定
  - 計算科学ロードマップ、開発者要求、を考慮して決定済み
- 概念設計マシンでの条件
  - 搭載メモリ量限界まで利用
  - 現実的な時間（約1ヶ月以下）で計算可能か検証

実行時間予測方法

- FX10の実行時間

$$T_{FX\ 10} = \sum_i K_i + \sum_i C_i + R$$

K_i : 演算カーネルiのプロファイラによる実測時間

C_i : 通信カーネルiのプロファイラによる実測時間

R : 演算カーネル、通信カーネル、I/Oを除く処理のプロファイラによる実測時間

- 概念設計マシンType(j)の実行時間予測

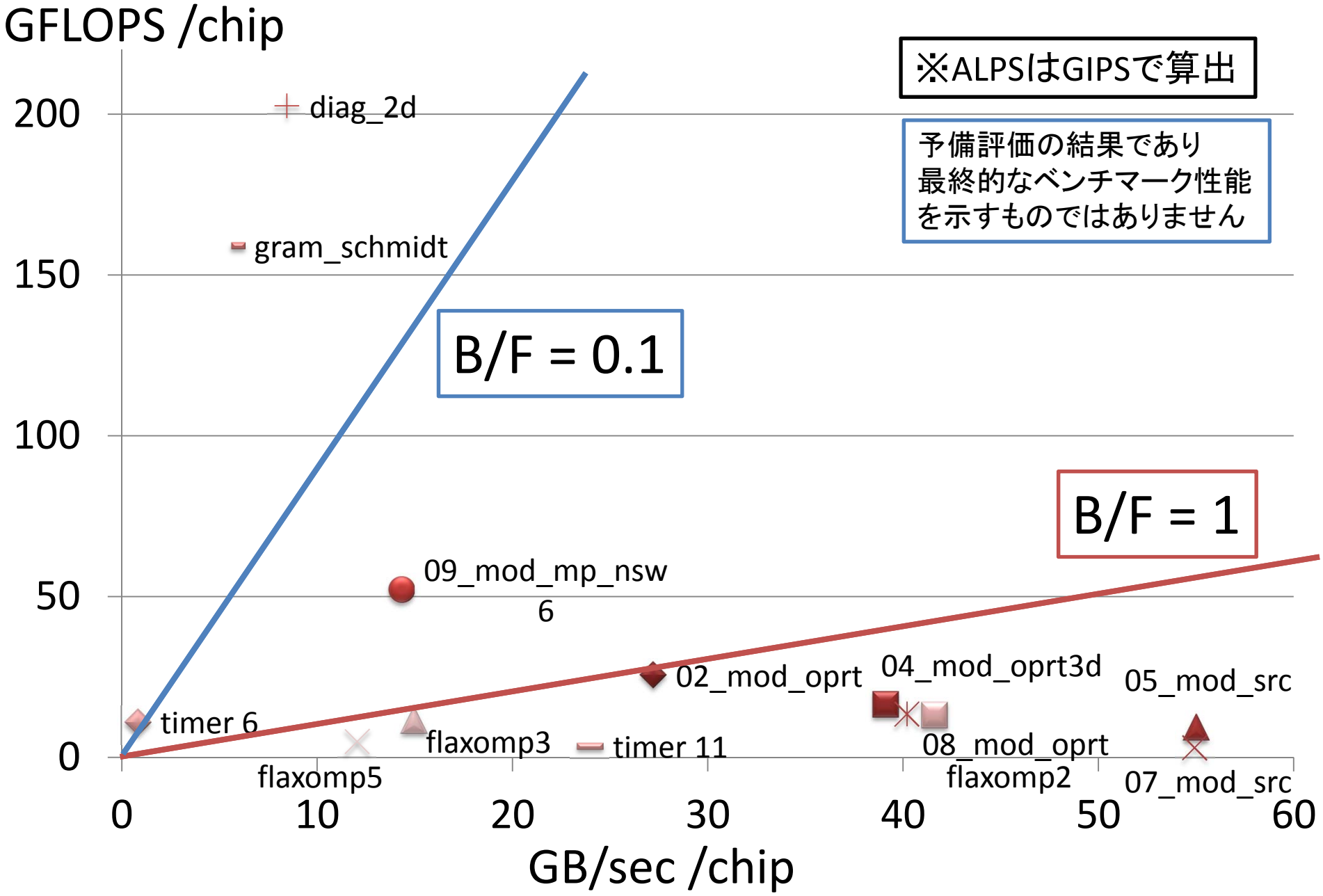
$$T_{Type(j)} = \sum_i \hat{K}_i + \sum_i \hat{C}_i + \hat{R}$$

$\hat{K}_i$: 演算カーネルiの性能予測ツールによる予測時間

$\hat{C}_i$: 通信カーネルiの予測時間

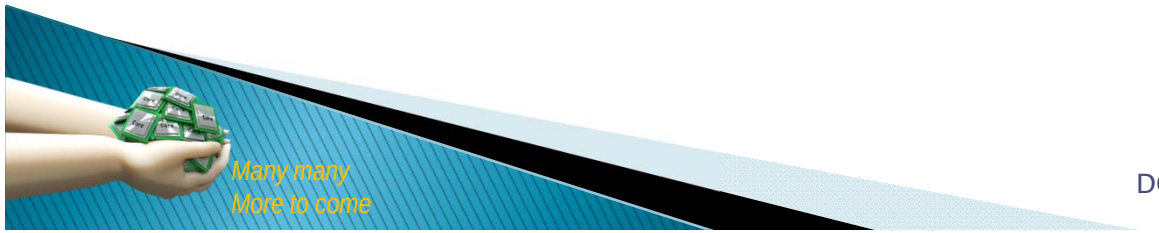
$\hat{R}$: 演算カーネル、通信カーネル、I/Oを除く処理の予測時間

演算カーネルの分類(実測B/F値):最適化前



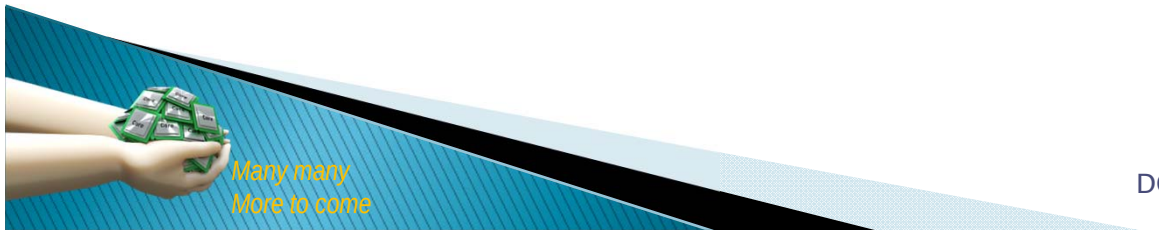
Agenda

- プロジェクト概要
- FX10での性能チューニング実例
- 異機種環境での評価
- 今後の予定

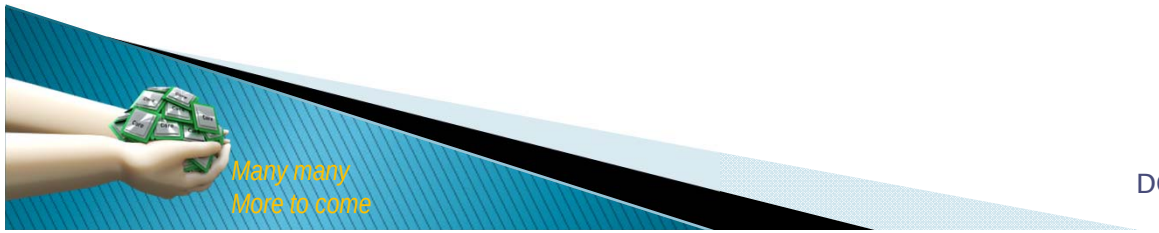


コードチューニング事例紹介

単に高速化が目的ではなく、
性能モデル化のための基礎データ
として妥当な性能にする
ための作業



ALPSの事例

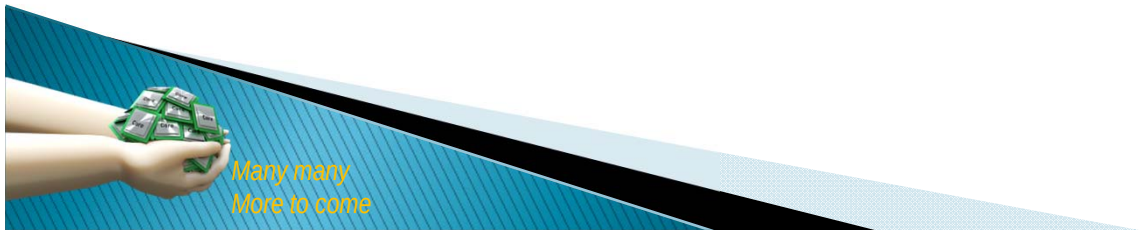


ALPS: Timer12のチューニング（1／3）

オリジナルコード

```
for (int c = 0; c < noc; ++c) {
        for (int r = 1; r < num_threads; ++r)
            estimates[c] += estimates_g[r][c];
}
```

- 各スレッドが演算したestimates_g[スレッド番号][:]の配列を集計してestimates[:]に格納する処理
- estimates[:]の実体はestimates_g[0][:]の為，1番スレッド以降の演算結果を足しこむ処理
  - 16SMPの場合、16箇所のメモリ領域にアクセスする必要がある
  - メモリアクセスが連続でない事から実行効率が出にくい



ALPS: Timer12のチューニング（2／3）

メモリアクセスを効率化するために，ループ交換を行い，
メモリアクセスを連続化

```
for (int r = 1; r < num_threads; ++r) {
      #pragma omp for schedule(static)
      for (int c = 0; c < noc; ++c) {
       estimates[c] += estimates_g[r][c];
      }
}
```

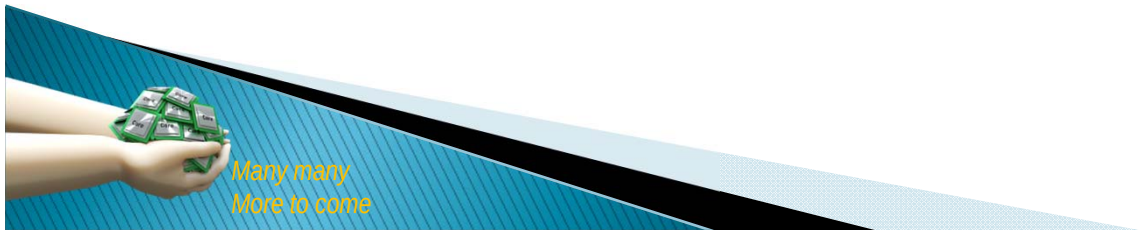
- 1スレッド毎に順次足しこむ形としてメモリアクセスを連続化
- メモリアクセスが連続の為、プリフェッチ等が効果的に働く
- メモリへのアクセス効率が上がり、実行時のメモリスループットがオリジナルの2.7[GB/s] から 56[GB/s] に向上
- ただし、estimates[c]をスレッド数分だけ繰り返しメインメモリから読込み・書込みをする必要がある。メモリへの負担が大きい。

ALPS: Timer12のチューニング（3／3）

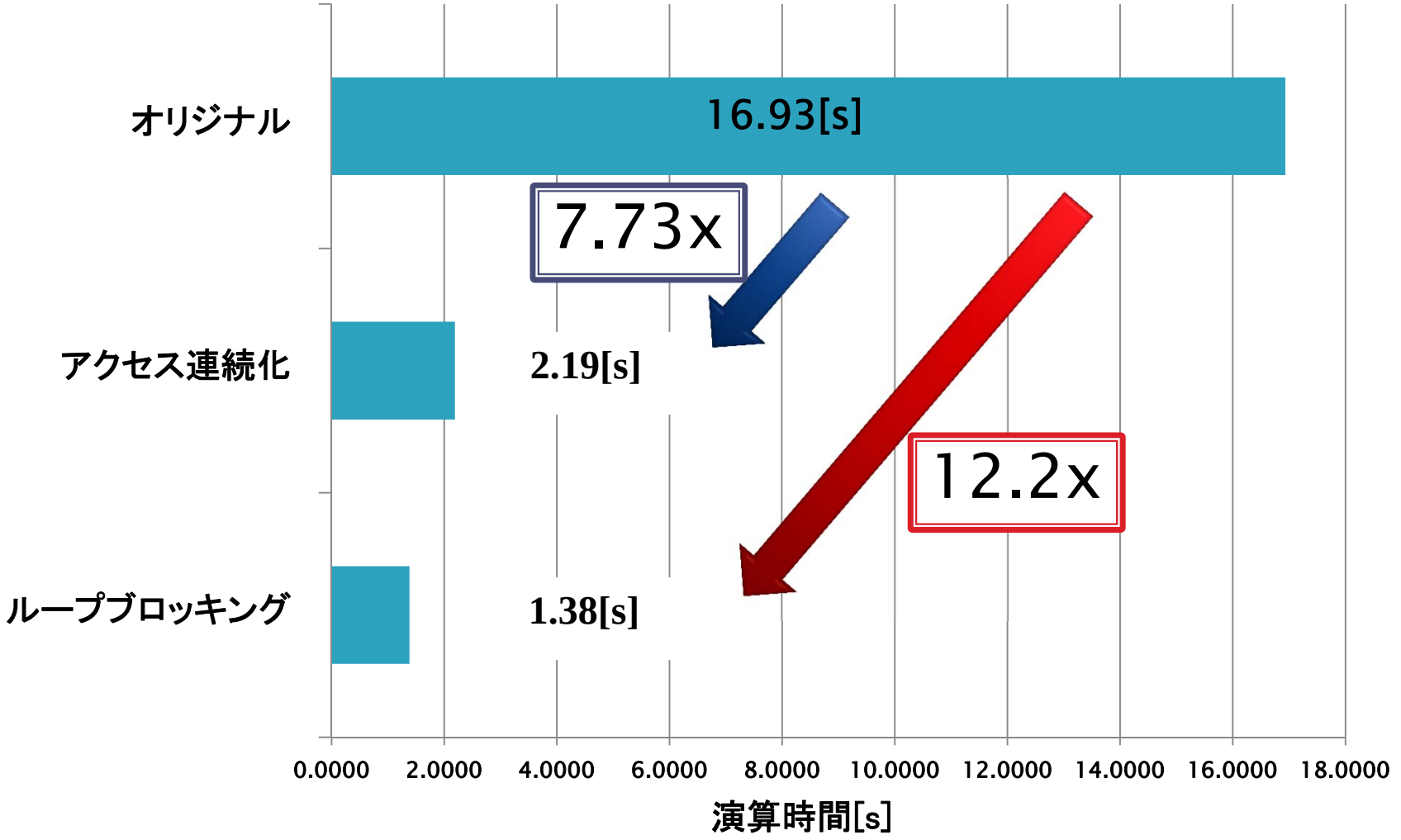
estimates[c]は繰り返しアクセスされる為、ループブロッキングを行い、
データがL2に残るようにする事でメインメモリへの負担を低減

```
looper::accu_step=4096;
for (int c = 0; c < noc; c+=looper::accu_step ) {
    for (int r = 1; r < num_threads; ++r) {
        for (int c2 = 0; c+c2 < noc && c2 < looper::accu_step;  ++c2){
         estimates[c+c2] += estimates_g[r][c+c2];
        }
    }
}
```

- メモリアクセスを連続化した上で、estimates[c]に関するアクセスでのメインメモリへの負担を軽減するループブロッキングを行い、estimates[c]がキャッシュに乗りやすいように修正
- これにより　L2へのミスヒット率が、3.3%から1.6%に軽減

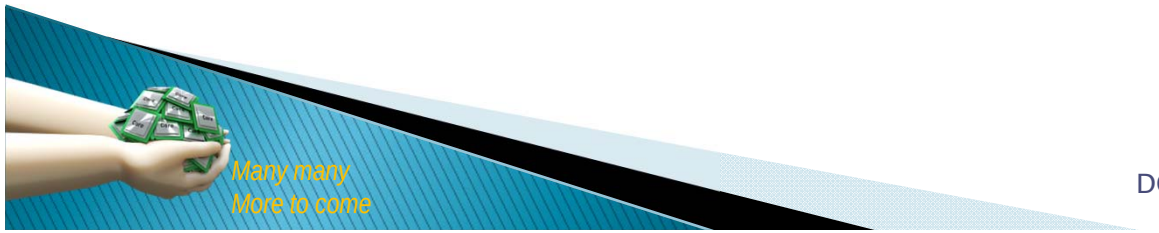


最適化結果：ALPS （Timer12）



ページサイズ256MB時の詳細プロファイルによる比較

NICAMの事例

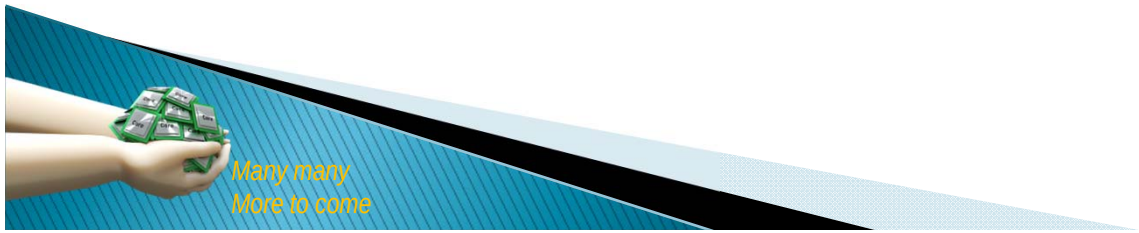


NICAMの最適化(1/6) –最適化結果

●検証内容：ループ切り出しによるカーネルループの最適化

NICAMのループ切り出しによる最適化結果

カーネル	最適化前	最適化後	向上倍率
mod_oprt2	1.06(s)	0.59(s)	1.80
mod_oprt3d_4	2.66(s)	2.21(s)	1.20
mod_src5	0.59(s)	0.51(s)	1.16
mod_src7	0.35(s)	0.35(s)	1.00
mod_oprt8	1.43(s)	1.05(s)	1.36



NICAMの最適化(2/6) -mod_oprt2

最適化前

```
do n=nstart,nend
インデックス計算
   scl(n,k,l)=(                                &
        +cdiv(0,ij,l,1)*vx(ij    ,k,l) &
        +cdiv(1,ij,l,1)*vx(ip1j  ,k,l) &
        +cdiv(2,ij,l,1)*vx(ip1jp1,k,l) &
        +cdiv(3,ij,l,1)*vx(ijp1  ,k,l) &
        +cdiv(4,ij,l,1)*vx(im1j  ,k,l) &
        +cdiv(5,ij,l,1)*vx(im1jm1,k,l) &
        +cdiv(6,ij,l,1)*vx(ijm1  ,k,l) &
        +cdiv(0,ij,l,2)*vy(ij    ,k,l) &
        ・ 途中省略
       +cdiv(6,ij,l,2)*vy(ijm1  ,k,l) &
       +cdiv(0,ij,l,3)*vz(ij    ,k,l) &
        ・ 途中省略
       +cdiv(6,ij,l,3)*vz(ijm1  ,k,l) &
        )*fact
 enddo
```

処理#1　処理#2　処理#3

最適化後

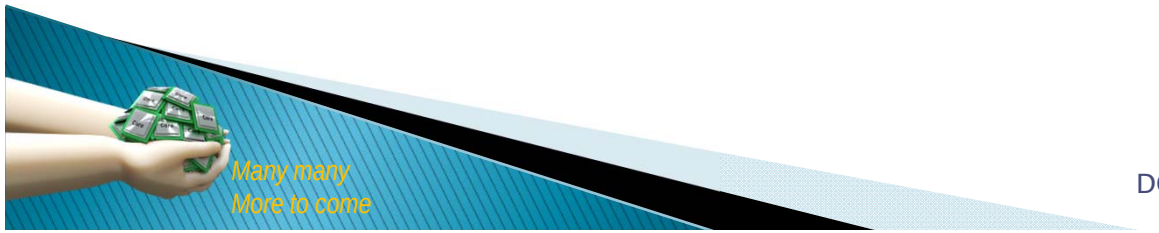
```
!OCL PARALLEL,UNROLL(8)
   do n=nstart,nend
     インデックス計算
      scl(n,k,l)=(                                &
           +cdiv(0,ij,l,1)*vx(ij    ,k,l) &
           +cdiv(1,ij,l,1)*vx(ip1j  ,k,l) &
           +cdiv(2,ij,l,1)*vx(ip1jp1,k,l) &
           +cdiv(3,ij,l,1)*vx(ijp1  ,k,l) &
           +cdiv(4,ij,l,1)*vx(im1j  ,k,l) &
           +cdiv(5,ij,l,1)*vx(im1jm1,k,l) &
           +cdiv(6,ij,l,1)*vx(ijm1  ,k,l) &
   enddo
!OCL PARALLEL,UNROLL(8)
   do n=nstart,nend
     インデックス計算
      scl(n,k,l)=scl(n,k,l)                       &
           +cdiv(0,ij,l,2)*vy(ij    ,k,l) &
           ・ 途中省略
           +cdiv(6,ij,l,2)*vy(ijm1  ,k,l) &
   enddo
!OCL PARALLEL,UNROLL(8)
   do n=nstart,nend
     インデックス計算
      scl(n,k,l)=scl(n,k,l)                       &
           +cdiv(0,ij,l,3)*vz(ij    ,k,l) &
           ・ 途中省略
           +cdiv(6,ij,l,3)*vz(ijm1  ,k,l) &
      scl(n,k,l)=scl(n,k,l)*fact
   enddo
```

処理#1　処理#2　処理#3

●最適化検討内容

- 7点格子のステンシル演算
- ロード・ストア43、演算数42でByte/Flop=8.19
- vx,vy,vzはij±1を参照。N展開でByte/Flop減少
  N展開でロードが6*N減少
- コンパイラはNで2展開
- Nで8展開しvx,vy,vzのループ分割で1.80倍向上

COCOの事例

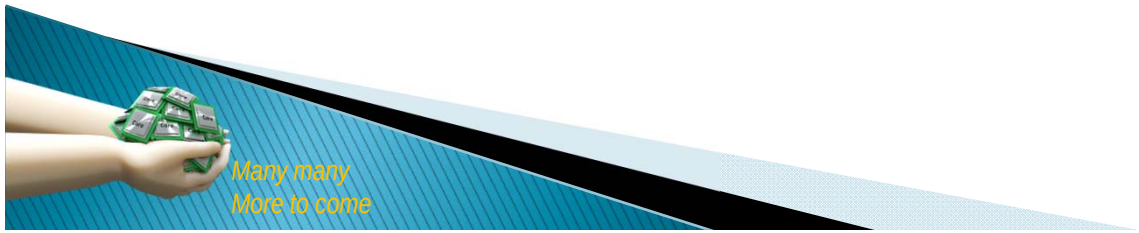


COCOの最適化(1/3) –最適化結果

●検証内容:ミニアプリによるカーネルループの最適化

COCOのミニアプリによる最適化結果

カーネル	最適化前	最適化後	向上倍率
flxtrc._OMP_2	1.433(s)	1.313(s)	1.09
flxtrc._OMP_3	0.926(s)	0.926(s)	1.00
flxtrc._OMP_5	1.084(s)	0.967(s)	1.12



COCOの最適化(2/3) – flxtrc._OMP_2

●最適化：if文の変更

最適化前

```
      DO IJ = IJTSTR-NXDIM-1, IJTEND+NXDIM+1
        IJLW  = IJ + LW
        FTX_REG = FTX (IJ, K, N)
        ALF_REG = ALF(IJ)
        IF ( UV(IJ, K) .GT. 0.D0 ) THEN
          ALFQ  = ALF_REG * ALF_REG
          ALF1  = 1.D0 – ALF_REG
          ALF1Q = ALF1 * ALF1
          F0(IJLW) =  ALF_REG * ( S0(IJLW, K, N)
     &         + ALF1 * (SX(IJLW, K, N)
     &         + ( ALF1 – ALF_REG ) * SXX(IJLW, K, N) ) )
                .
                途中略
        ELSE
          ALFQ  = ALF_REG * ALF_REG
          ALF1  = 1.D0 – ALF_REG
          ALF1Q = ALF1 * ALF1
          F0(IJLW) = ALF_REG * ( S0(IJ, K, N)
     &         – ALF1 * (  SX(IJ, K, N)
     &         – ( ALF1 – ALF_REG ) * SXX(IJ, K, N) ) )
                以降略
```

最適化後

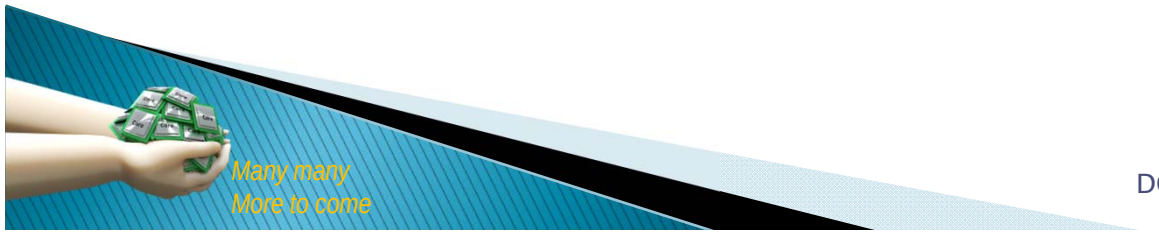
```
      DO IJ = IJTSTR-NXDIM-1, IJTEND+NXDIM+1
        IJLW  = IJ + LW
        FTX_REG = FTX (IJ, K, N)
        ALF_REG = ALF(IJ)
        IF ( UV(IJ, K) .GT. 0.D0 ) THEN
          IJP=IJLW
          SS=1.0
        ELSE
          IJP=IJ
          SS=-1.0
        ENDIF
        ALFQ  = ALF_REG * ALF_REG
        ALF1  = 1.D0 – ALF_REG
        ALF1Q = ALF1 * ALF1
        F0(IJLW) =  ALF_REG * ( S0(IJP ,K, N)
     &       + SS*ALF1 * (SX(IJP ,K, N)
     &       + SS*( ALF1 – ALF_REG ) * SXX(IJP ,K, N) ) )
                .
        以降略
```

正負でインデックスを変更

thenとelseで正負が反転している所はSSを掛ける。

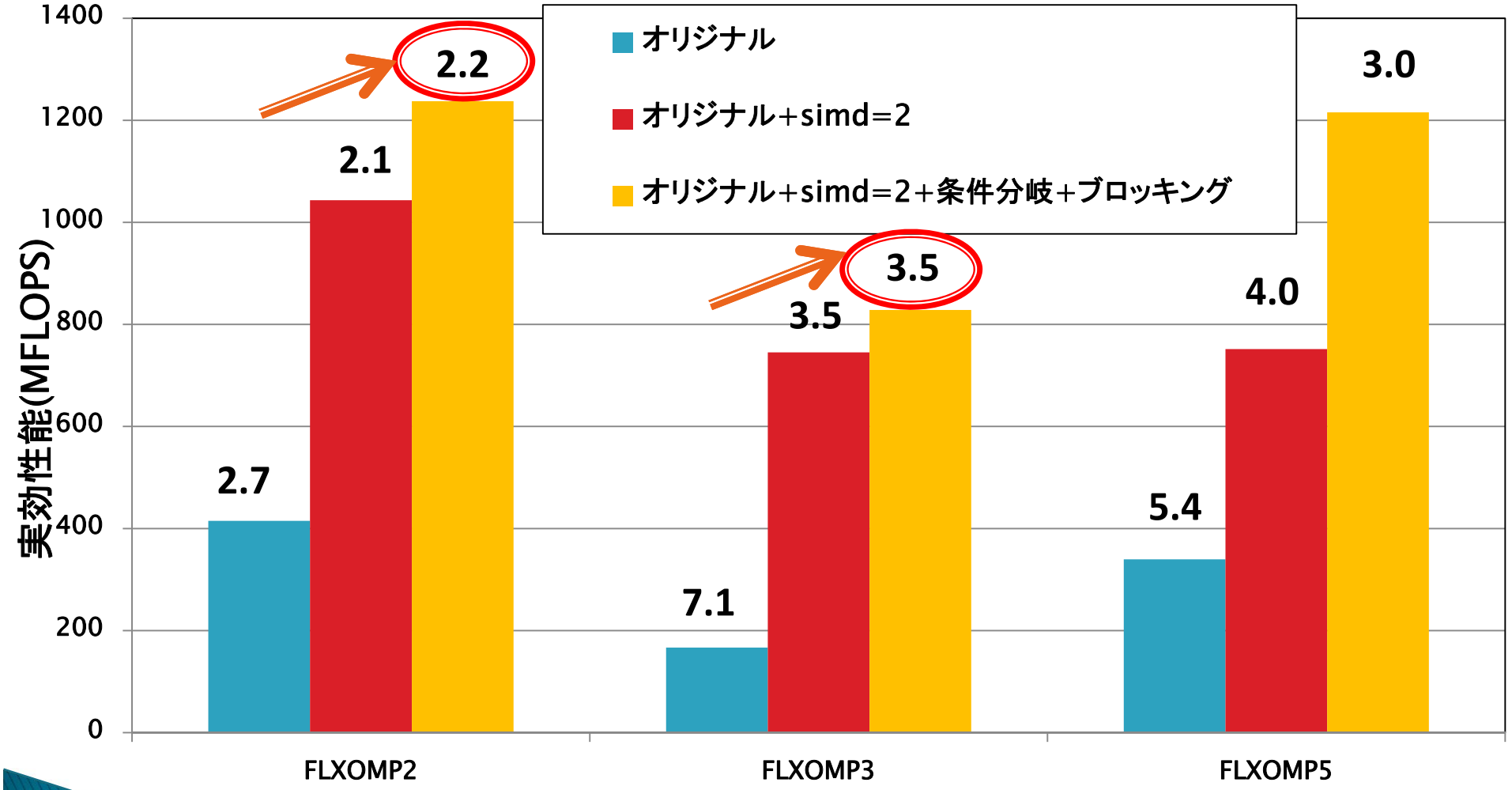
- ifのthenとelseは似た式
  ①配列ロードのインデックスがIJLWとIJで異なる。②正負の一部が異なる。
- ifのthenとelseのインデックスの内容を変更し、以降のifを削除
- if文の変更で1.09倍性能向上

B/F値の変化



実効性能とB/F（Byte per Flops）値　：COCOの実例

●COCOの主要ループに最適化を適用したときの実効性能とB/F値

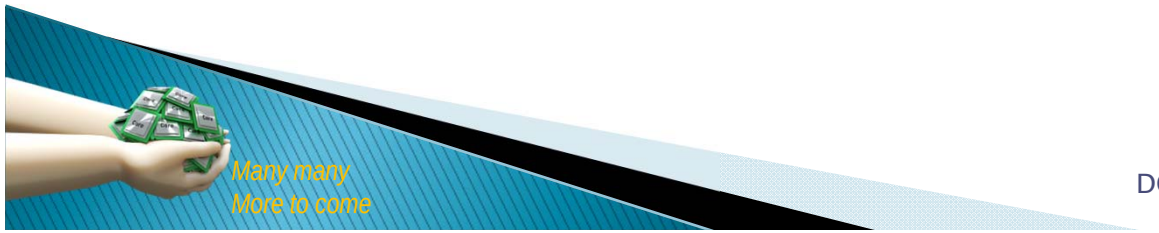


実効性能が高くなっても B/F値が低下していない場合がある

⇒B/F値だけでなく、実時間をみないとダメ！

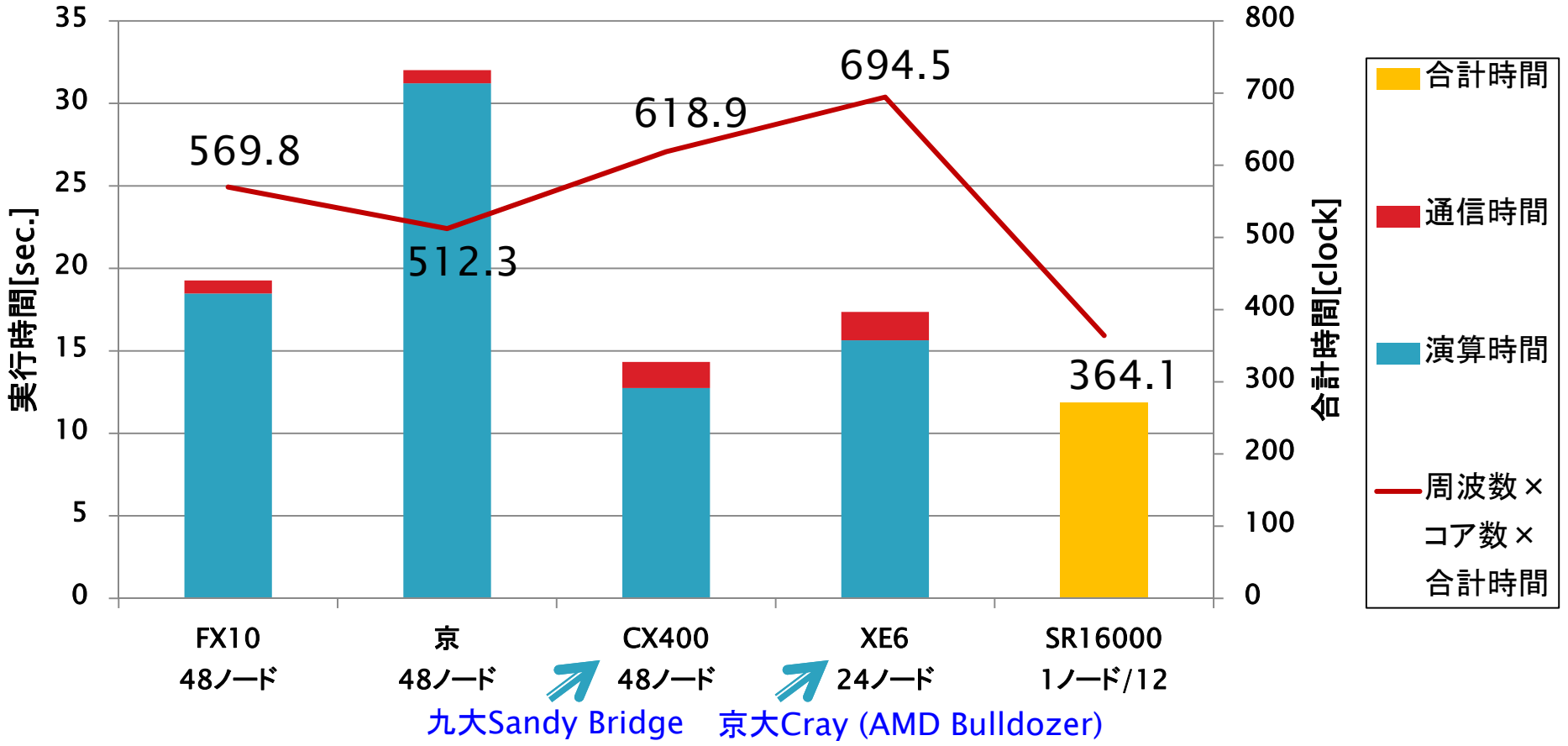
Agenda

- プロジェクト概要
- FX10での性能チューニング実例
- 異機種環境での評価
- 今後の予定



ALPSの実行時間評価
(異機種環境)

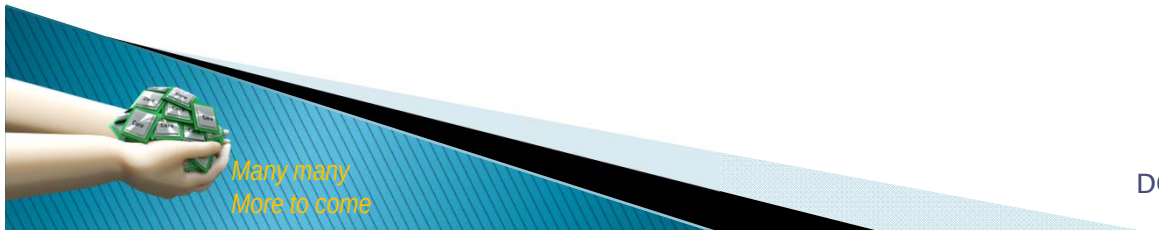
・FX10及びCX400:48ノード 48MPI×16OMP
・京:48ノード 48MPI×8OMP
・XE6: 24ノード 48MPI×16OMP
・SR16000:1ノード 4MPI×8OMPの結果をMPI数の比である1/12で換算



- 演算時間はCX400が最短
- 通信時間はFX10が最短
- 実行時間に周波数及びコア数を掛けて正規化した場合SR16000が最も高効率
- XE6は整数2パイプ低IPC(Instructions Per Cycle)の為、若干効率が低め

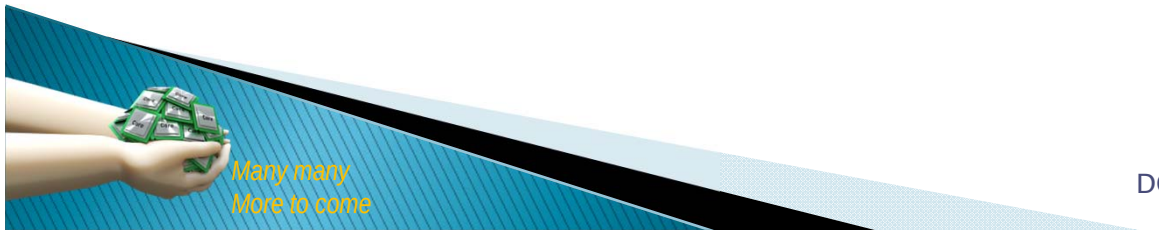
Agenda

- プロジェクト概要
- FX10での性能チューニング実例
- 異機種環境での評価
- 今後の予定



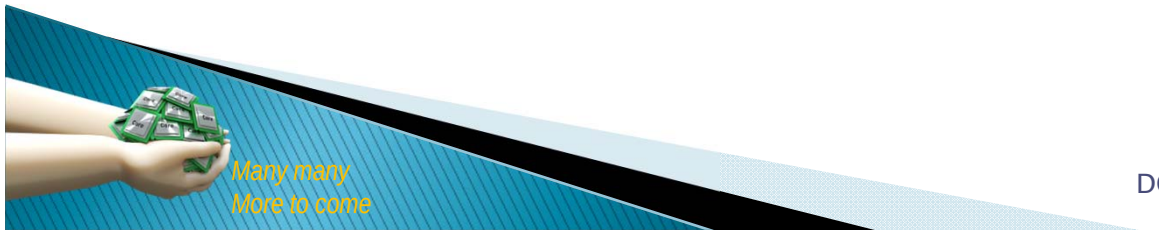
H25年度　理研FS選定ミニアプリ評価

- 4アプリについての、性能プロファイル、性能チューニング情報は、理研FS側に提供済み
- QCD
  - 現時点で、東大FSにコード引き渡し済み
  - FX10で評価を開始済み
- Modylas
  - ライセンス契約中
  - 東大、日立、富士通、九州大、ごとに個別にライセンス契約を結ぶ
  - 6月から評価開始？
- その他、2本程度のアプリを理研側が選定し、東大FSアプリWGで受け入れの予定



H25年度予定(1／2)

- H24年度残務作業
  - RSDFTの通信時間解析と性能予測
  - 性能予測の高度化
    - H24年度の最新のコードチューニング済みコードで再評価
- カーネルベンチの公開
  - COCOは問題なし。
  - NICAMはライセンス確認が必要。
- 従来4種アプリの高度最適化
  - IntelおよびPower7(SR16K)CPUとの比較
  - その他のチューニング(アルゴリズム変更)



H25年度予定(2／2)

- 超並列向きアルゴリズム採用と性能評価（性能予測）
  - RSDFTの直交化処理
    - 通信回数が少ない新アルゴリズムを適用し性能評価
    - CAQR (Communication Avoidance QR)
    - 米澤CREST採択課題の筑波大櫻井グループと連携（開発中のコードを利用させてもらう）
      - CAQRプログラムをRSDFTに組み込み性能評価を予定