Vol.2013-HPC-139 No.4
2013/5/29

レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究のためのアプリケーション最適化と異機種計算機環境での性能評価

片桐孝洋†　大島聡史†　中島研吾†　米村崇††　熊洞宏樹††
樋口清隆†††　橋本昌人†††　高山 恒一†††　藤堂眞治†3，岩田 潤一†4
内田和之†4，佐藤正樹†5，羽角博康†5，黒木聖夫†6

本報告では，レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究におけるコデザインで用いるアプリケーションについて，コード最適化による性能チューニングを行った結果を報告する．Sparc64，Intel，IBM の CPU を構成要素とする異機種計算機環境での性能評価も報告する．コード最適化の結果，実測での byte per flops (B/F)値が上がっても実時間は高速化される例があり，B/F 値のみの評価は不十分なことを示す．

1. はじめに

本報告では，数値計算レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究（以降，単に調査研究とよぶ）におけるターゲットアプリケーションにおいて，概念設計マシンでの性能予測を妥当なものにするための性能最適化について述べる．

本報告の構成は以下のとおりである．2 章で調査研究の目的とターゲットアプリケーションを説明する．3 章では，富士通 PRIMEHPC FX10（以降，FX10）を用いた性能チューニングの内容と効果を紹介する．4 章では，FX10 と日立 SR16000（以降，SR16K）との性能比較を中心に，異機種環境で行った性能評価の結果を紹介する．このとき，チューニング前後において，実測での Byte per Flops (B/F)値の変化について検証する．最後に，本原稿で得られた知見を述べる．

2. 調査研究の目的とターゲットアプリケーション

2.1 将来の HPCI システムのあり方の調査研究

本調査研究は，第 4 期科学術基本計画（平成 23 年 8 月 19 日閣議決定）で掲げられた国家存立の基盤としての世界最高水準のハイパフォーマンス・コンピューティング技術の強化，及び科学術基盤の充実強化に向けた重要な取り組みの一つとして，HPC 技術等の HPCI システムの高度化に必要な技術的知見を獲得することを目的とし，平成 24 年度，平成 25 年度に調査研究を実施するものである[1].

† 東京大学 情報基盤センター スーパーコンピューティング研究部門
†† 日立製作所 情報・通信システム社
†3 東京大学 物性研究所
†4 東京大学 大学院工学系研究科
†5 東京大学 大気海洋研究所
†6 海洋研究開発機構

2.2 レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究

本調査研究は，東京大学を中心とし，九州大学，富士通，日立製作所，日本電気による調査研究である[2]．2018 年頃設置可能な並列システムを，汎用型プロセッサからのアプローチでフィージビリティ・スタディ（FS）を行う．アプリケーション，システムソフトウェア，アーキテクチャの co-design を行う．システムソフトウェアスタック共通化 (From PC cluster to high-end machines)を行う．

本報告はこのうち，アプリケーション性能予測に関する検討事項に相当する．

2.3 本調査研究における進め方

「今後の HPCI 技術開発に関する報告書」[3][4]を尊重し，京および FX10 におけるアプリケーション並列性能および I/O 性能，耐故障性および運用・保守の観点で課題を精査し，概念設計に反映する．進め方の概要を図 1 に示す．

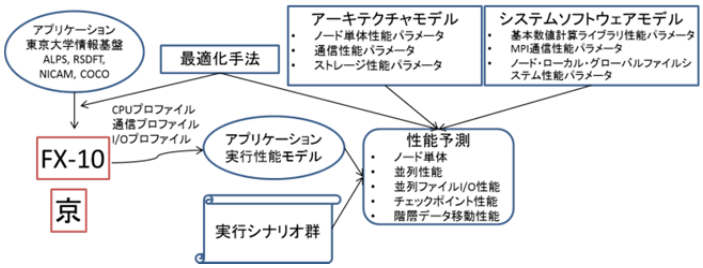

図 1　本調査研究における調査研究の進め方

本報告は，図 1 における性能予測のための手法と最適化方式の調査研究に関連する．

2.4 利用シナリオ

利用シナリオ（図 1 における「実行シナリオ群」）とは，各アプリケーションの実行において，特徴的なジョブの実行形態のことである．主に以下のアンサンブル型を想定し，

入出力ファイルなどの I/O 性能を含め，システム全体の設計に反映させる．

- **アンサンブル型：**
  全系の 1/10～1/100 の資源を利用する 1 ジョブに対し，複数ジョブを同時実行して全資源を使い切る形態．この形態では，複数同時のファイル入力，および複数同時のファイル出力が起こる．

2.5 性能予測手法

図 1 を進めるに当たり，ターゲットアプリケーションにおける概念設計中の計算機性能の実行時間を予測するため，以下の手法をとる．

1. **ホットスポット同定：**基本プロファイラ（主要な関数やループの実行時間が取得可能な性能プロファイラ）を用いて，複数のホットスポット（ループレベル）を同定する．その後，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を行う．できるだけ，採用されている数理アルゴリズム（支配方程式，離散化方法）とホットスポットの対応がわかるようにする．
2. **ホットスポット分離：**計算部分，通信部分，I/O 部分のホットスポットを基のソースコードから分離する．
   - 計算部分：演算カーネルと呼ぶ．
   - 通信部分：通信カーネルと呼ぶ．
   - I/O 部分： I/O カーネルと呼ぶ．
3. **通信パターン確認：**プロファイラによる可視化ツールや対象コードを解析することで，通信パターンを確認する．
4. **詳細プロファイルと分析：**詳細プロファイラ（対象のループにおけるハードウェア上の性能情報が取得可能な性能プロファイラ）を用い，ホットスポットごとにハードウェア性能情報を取得して分析する．
   - 演算カーネルにおける，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する．
   - マシン特化の書き方，および，汎用的な書き方，の２種を区別する．
   - 演算カーネル，通信カーネル，I/O カーネルの分類をする．
6. **詳細モデル化：**ハードウェア因子による実行時間の予測ができるようにする．

本調査研究では，詳細モデル化を行うに当たり，FX10 で提供される性能プロファイラを用いる．このことで，演算および通信カーネルを抽出できる．また，現存する計算機でのハードウェア因子について，プロファイラを通じて取得ができる．この情報を基に，現在設計中の計算機での性能予測が可能となる．

2.6 ターゲットアプリケーションの特徴

以下にターゲットアプリケーションの計算機システムに対する性能要求をまとめる．なお，メモリ帯域，FLOPS，Byte/Flops (B/F) 値は，「計算科学ロードマップ白書」（2012 年 3 月）[5]における見積値，もしくは，本調査研究でアプリケーション開発者自身が行った見積値である．

(1) ALPS (Algorithms and Libraries for Physics Simulations)

新機能を持った強相関・磁性材料の物性予測・解明のシミュレーションである．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を利用している．

- 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 24 時間，生成ファイル 10GB.
  同時実行 1000 ジョブ，総合生成ファイル：10TB.

(2) RSDFT (Real-Space Density-Functional Theory)

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である．実空間差分法を利用している．

- 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 10 時間，生成ファイル 500TB.
  同時実行 10 ジョブ，総合生成ファイル 5PB.

(3) NICAM (Nonhydrostatic ICosahedral Atmospheric Model)

長期天気予報の実現，温暖化時の台風・豪雨等の予測のシミュレーションである．正 20 面体分割格子非静力学大気モデルを採用し，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する．

- 総メモリ：1PB，メモリ帯域：300PB/sec.
- 演算性能：100PFLOPS（公称値 B/F = 3）.
- 利用シナリオ:１ジョブ 240 時間，生成ファイル:8PB.
  同時実行 10 ジョブ，総合生成ファイル 80PB.

(4) COCO (CCSR Ocean COmponent Model)

海況変動予測，水産環境予測のシミュレーションである．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する．

- 総メモリ：320TB，メモリ帯域：150PB/sec.

- 演算性能：50PFLOPS（B/F = 3）.
- 利用シナリオ: 1 ジョブ 720 時間, 生成ファイル 10TB. 同時実行 100 ジョブ，各生成ファイル 1PB.

2.7 各アプリケーションでの B/F 値の分布

前回の報告[6]において，FX10 のプロファイラを用いて実測された，各アプリケーションの演算カーネルの Byte/Flops 値（B/F 値）を報告した．この B/F 値は，対象の実行に対してハードウェア情報を取得することで，メモリからキャッシュに移動した実際のデータ量（Byte）と，実際に観測された浮動小数点演算数（Flops）から算出したものである．つまり B/F 値が高い演算カーネルは，ノイマン型の計算機においては，一般的に好ましくない特性の演算カーネルといえる．

この報告で扱う演算カーネルは，報告[6]での演算カーネルに対し以下の違いがある：(1)NICAM の演算カーネルの対象が変わっている；(2)コンパイラオプションのチューニングを行っている；今回の演算カーネルの実測 B/F 値の分布を図 2 にのせる．なお図 2 の B/F 値は，性能最適化を行うと変化するので，最終的なベンチマーク性能ではない．

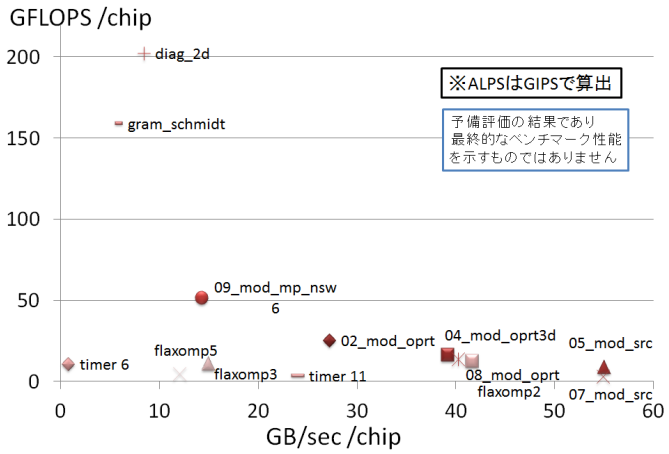

図 2　各アプリケーションの B/F 値の分布（最適化前）．ALPS は GFLOPS の代わりに Giga Instruction Per Second (GIPS)で算出している．

図 2 から，一部の演算カーネル（主に RSDFT の diag_2d と gram_schmidt）は演算束縛であるが，多くはメモリバンド幅束縛の演算カーネルであることがわかる．

3. FX10 におけるチューニング内容と効果

3.1 概要

ここでは, 東京大学情報基盤センターに設置された FX10（SPARC64 IXfx（1.848 GHz，コア数 16））を用いて，各アプリケーションの演算カーネルのチューニングを行った結果（2013 年 4 月 10 日現在）について事例を紹介する．

各アプリケーションの問題サイズは，エクサスケールでの実行を考慮して決められている．プロセス当たりの問題サイズを一定にして評価する「弱スケーリング」を採用している．

3.2 カーネルベンチ

NICAM と COCO については，本プロジェクトで開発した「カーネルベンチ」を利用してチューニングを行っている．このカーネルベンチとは，演算カーネルのみで動作するようにしたベンチマークソフトウェアである．フルアプリケーションにおける演算カーネルで必要なデータを保存しカーネルベンチの動作前にそのデータを読み込むことで，フルアプリケーションの動作を模倣する．

3.3 FX10 における主なチューニングの内容

この節では，FX10 において実施された性能最適化のうち，特に効果のあった最適化手法を説明する．

3.3.1 ALPS

ALPS でチューニングの効果が最もあったのは，以下の Timer12 の演算カーネルに関するコード最適化である．

- Timer12 の演算カーネル（最適化前）

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

以上の Timer12 の演算カーネルは，estimates_g[][]のアクセスが連続アクセスになっておらず，実行環境に依存してキャッシュミスヒットにより性能が劣化する．そこで，以下の修正を行った．

- Timer12 の演算カーネル（最適化後）

```
for (int rs = 1; rs < num_threads; rs += looper::accu_step) {
   int re = rs + looper:accu_step > num_threads ? num_threads : rs + looper:accu_step;
   #pragma omp for schedule(static) Nowait
   for (int c=0; c < nc; ++c) {
     for (int r = rs; r < re; ++r)
       estimates[c] += estimates_g[r][c];
     }
   }
```

以上のコードでは，estimates_g[][]の非連続アクセス長を限定することで，キャッシュヒット率を向上させる．ここで，looper:accu_step はチューニングパラメタであり，FX10 では 5 が設定されている．

以上のコード最適化を行うことにより，1 ノード当たり 16 スレッド実行を行うハイブリッド MPI 実行において，

約 1.9 倍の速度向上を達成した．また 4 節で紹介するように，ループ交換をさらに実施することで，約 10 倍高速化される．

3.3.2 RSDFT

RSDFT では，プログラムで指定が必要な以下の性能パラメタのパラメタ調査を行うことで，性能チューニングを行った．

- NBLK
  - 大きめの値が良い．性能パラメタ MB の約数を指定しなくてはならない．0 を指定すると NBLK=MB が仮定される．
- NBLK1
  - DGEMM を使用せずに DGEMV に切り替える大きさの指定．0 を指定すると NBLK1=4 が仮定される．
- MB_d
  - 固有値計算をまとめて実行する単位の指定．デフォルト値は 4 である．

以上のパラメタ調査のうち，MB_d のチューニングが最も効果があった．その結果を図 3 に載せる．

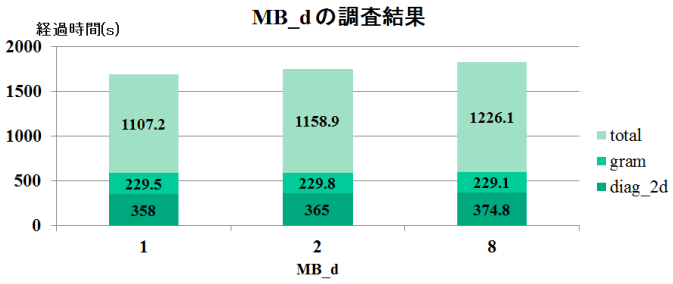

図 3　MB_d のパラメタ調査結果

図 3 により，MB_d=1 の時が最適である．デフォルト実行の MB_d=4 による実行結果は 1226.1 [sec.]であったが，MB_d=1 にすることで 1107.2 [sec.]に高速化される．これは，約 10%の高速化である．

3.3.3 NICAM

NICAM においては，演算カーネルに対し，ループ分割，ループ融合，およびブロック化の手法を，それぞれの演算カーネルに適用した．以下に主な例を載せる．

- **mod_oprt2 の場合（ループ分割＋アンローリング）**

以下のオリジナルループの構成に対し：

```
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) &
                                  : 以上を処理＃１
      +cdiv(0,ij,l,2)*vy(ij      ,k,l) &
        途中省略
      +cdiv(6,ij,l,2)*vy(ijm1    ,k,l) &
                                  : 以上を処理＃２
      +cdiv(0,ij,l,3)*vz(ij      ,k,l) &
        途中省略
      +cdiv(6,ij,l,3)*vz(ijm1    ,k,l) &
                                  : 以上を処理＃３
      )*fact
enddo
```

以下のようにループ分割を適用し，かつコンパイラディレクティブによるループアンローリングを 8 段適用した．

```
!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)
                                  : 処理＃１の部分
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)
                                  : 処理＃２の部分
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)
                                  : 処理＃３の部分
  scl(n,k,l)=scl(n,k,l)*fact
```

```
enddo
```

以上の修正を行い，vx, vy, vz のループ分割を行うことで，オリジナルコードに対して約 1.8 倍の速度向上を達成した．

- mod_oprt3d_4 の場合（アンローリング＋ループ融合）

オリジナルコードに対し，コンパイラディレクティブによる以下の 4 段アンローリングを指定した．

```
!OCL SERIAL,UNROLL(4)
do k = ADM_kmin, ADM_kmax+1
  do ij = 1, ADM_gall
    flx_vz(ij,k) =
( (GRD_afac(k)*VMTR_RGSGAM2(ij,k,l)*rhovx_in(ij,k,l) &
+GRD_bfac(k)*VMTR_RGSGAM2(ij,k-1,l)*  rhovx_in(ij,k-1,l)
&
  途中省略
) * 0.5D0 * VMTR_GSGAMH(ij,k,l) * VMTR_GZZH(ij,k,l)&
    ) + rhow(ij,k,l) * VMTR_RGSH(ij,k,l)
  enddo
enddo
```

以上に加え，以下の箇所のループ融合を行った．

```
do n = nstart, nend
  ij      = n
  ip1j    = ij+1
  ip1jp1 = ij+1 + ADM_gall_1d
  ux1 = -( rhovx_k(ij)      + rhovx_k(ip1j)    )
  uy1 = -( rhovy_k(ij)      + rhovy_k(ip1j)    )
  uz1 = -( rhovz_k(ij)      + rhovz_k(ip1j)    )
    途中省略
  sclt(ij,k,ADM_TI) =
enddo
do n = nstart, nend
```

←この 2 ループを融合

```
  ij      = n
  ijp1    = ij    + ADM_gall_1d
  ip1jp1 = ij+1 + ADM_gall_1d
  ux1 = -( rhovx_k(ij)      + rhovx_k(ip1jp1) )
  uy1 = -( rhovy_k(ij)      + rhovy_k(ip1jp1) )
  uz1 = -( rhovz_k(ij)      + rhovz_k(ip1jp1) )
    途中省略
  sclt(ij,k,ADM_TJ)=
enddo
```

以上を併用して行うことで，オリジナルコードに対して約 1.2 倍の速度向上を得た．

- **mod_src5 の場合（ブロック化）**

オリジナルコードに対して，以下のキャッシュブロック化（タイリング）を行った．

```
do l = 1, ADM_lall
do ib = 1, ADM_gall, iblock
  i1 = ib
  i2 = min(ib + iblock -1, ADM_gall)
  do k = ADM_kmin, ADM_kmax+1
  do ij = 1, ADM_gall  -> do ij = i1, i2
      ：ij のアクセス範囲をキャッシュブロック長に変更
    flx_vz(ij,k,l) =
          ( ( AFACovGSGAM2(ij,k,l) * rhogvx(ij,k    ,l) &
          + BFACovGSGAM2(ij,k,l) * rhogvx(ij,k-1,l)   &
  )*0.5D0*VMTR_GSGAMH(ij,k,l)*VMTR_GZXH(ij,k,l) &
            + ( AFACovGSGAM2(ij,k,l) * rhogvy(ij,k,l) &
            + BFACovGSGAM2(ij,k,l) * rhogvy(ij,k-1,l) &
  )*0.5D0*VMTR_GSGAMH(ij,k,l)*VMTR_GZYH(ij,k,l) &
              +( AFACovGSGAM2(ij,k,l)*rhogvz(ij,k,l) &
            + BFACovGSGAM2(ij,k,l) * rhogvz(ij,k-1,l) &
  )*0.5D0*VMTR_GSGAMH(ij,k,l)*VMTR_GZZH(ij,k,l) &
  ) + rhogw(ij,k,l) * VMTR_RGSH(ij,k,l)
  enddo
enddo
```

以上のブロック化により，オリジナルコードに対して約 10%の速度向上を確認した．

- **mod_oprt8 の場合（ループ分割＋アンローリング）**

オリジナルコードに対し，コンパイラディレクティブによる 2 段アンローリングに加え，2 か所のループ分割を行った．チューニング済みコードは以下である．

```
do k = 1, ADM_kall
  nstart = suf(ADM_gmin-1,ADM_gmin-1 )
  nend   = suf(ADM_gmax, ADM_gmax )
!ocl unroll(8)
  do n = nstart, nend
    sa_p = s(n,k,l)   &
     +wrk(n,k,l,dsx)*(cp(n,k,l,ADM_AI,GRD_XDIR)-・・・
     +wrk(n,k,l,dsy)*(cp(n,k,l,ADM_AI,GRD_YDIR)-・・・
    +wrk(n,k,l,dsz)*(cp(n,k,l,ADM_AI,GRD_ZDIR)-・・・
    sa(ADM_AI,n,k,l) =(0.5D0+sign(0.5D0,c(1,n,k,l)))*sa_p+
  以降、 ADM_AIJ,ADM_AJ に関する同様の式が続く
  end do
!ocl unroll(8)
  do n = nstart, nend
     sa_m = s(n+1,k,l) &
     +wrk(n+1,k,l,dsx)*(cp(n,k,l,ADM_AI,GRD_XDIR)-・・・
     +wrk(n+1,k,l,dsy)*(cp(n,k,l,ADM_AI,GRD_YDIR)-・・・
     +wrk(n+1,k,l,dsz)*(cp(n,k,l,ADM_AI,GRD_ZDIR)-・・・
    sa(ADM_AI,n,k,l)= sa(ADM_AI,n,k,l) +
      (0.5D0-sign(0.5D0,c(1,n,k,l)))*sa_m
  以降、 ADM_AIJ,ADM_AJ に関する同様の式が続く
```

```
   end do
end do
```

以上のチューニングにより，オリジナルコードに対して約 1.36 倍の速度向上を得た．

3.3.4 COCO

COCO については， if 文の変更，および，ループ融合の最適化を行った．

- **flxtrc._OMP_2 の場合（if 文の変更）**

以下のオリジナルコードに対し：

```
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
```

以上は、正負が反転する以外は同様の処理をしていることに注目し，以下のようにコードを修正する．

```
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 をかける

以上の変更により，オリジナルコードに対して約 10%の速度向上を得た．

- **flxtrc._OMP_5 の場合（ループ融合）**

以下のオリジナルコード中のループ＃１と＃２を融合した．

```
DO K = KSTR, KEND
      DO IJ = IJ1, IJ2
         IF ( UV(IJ, K) .GT. 0.D0 ) THEN
            ZF0(IJ-IJ1+1, KU) =…
            FTZ (IJ, K, N) =…
         ELSE
            ZF0(IJ-IJ1+1, KU) =…
            FTZ (IJ, K, N) =…
         END IF
      END DO
   END DO
   DO K = KSTR, KEND
      DO IJ = IJ1, IJ2
         IF ( UV(IJ, K) .GT. 0.D0 ) THEN
            SM (IJ, KU, N) =…
            SXY(IJ, KU, N) =…
         ELSE
            SM (IJ, KU, N) =…
            SXY(IJ, KU, N) =…
         END IF
      END DO
END DO
```

ループ＃１

ループ＃２

以上の変更により，オリジナルコードに対して約 1.12 倍の速度向上を得た．

4. 総合評価

4.1 計算機環境

本性能評価では，東京大学情報基盤センターに設置された FX10（SPARC64 IXfx(1.848 GHz，コア数 16)）を中心とし，HPCI で提供される以下の計算機資源を利用した．

- スーパーコンピュータ「京」(理化学研究所計算科学研究機構)
  - SPARC64 VIIIfx (2.0GHz，コア数 8)
- スーパーコンピュータ CRAY XE6（京都大学学術情報メディアセンター）
  - AMD Opteron 6200 系 (2.5GHz，コア数 16 x 2 ソ

ケット)

- 高性能演算サーバ Fujitsu PRIMERGY CX400S1（九州大学情報基盤研究開発センター）
  - Intel Xeon E5-2680 (2.7GHz, コア数 8 x 2 ソケット)

また HPCI 提供マシンでないが以下の計算機を利用した。

- HITACHI SR16000/M1（東京大学情報基盤センター）
  - IBM Power7 (3.83 GHz, 8 コア x 4 ソケット)

4.2 FX10 における B/F 値の変化と実行時間との関係

ここでは FX10 を用いて，前節のチューニングを適用する前後で，実測する B/F 値（ALPS においては，実測する B/I 値）がどのように変化するか調査する．

図 4 に，ALPS における実測 B/I 値の変化をのせる．

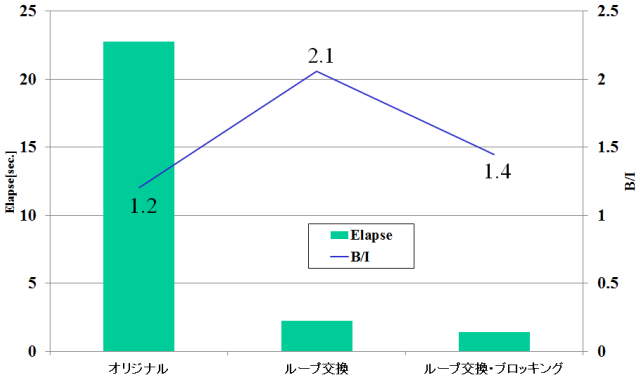

図 4　ALPS におけるチューニング前後での
実測 B/I 値の変化（演算カーネル Timer12）

図 4 から，チューニングを施すことで実行時間は劇的に改善し高速化されるが，B/I 値は必ずしも低下しない（むしろ少し増加する）ことがわかる．

図 5 に，NICAM における実測 B/F 値をのせる．各棒グラフの上の数字はそれぞれの実測 B/F 値を示す．

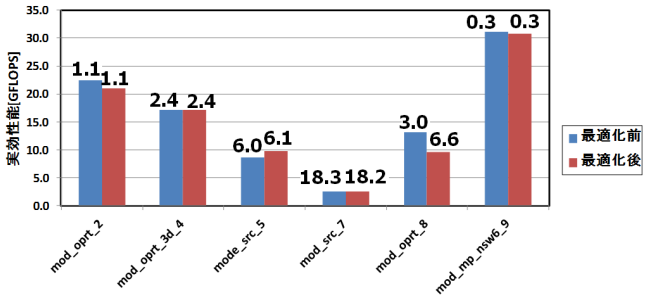

図 5　NICAM における
チューニング前後での実測 B/F 値の変化

図 5 から，チューニングの前後で B/F 値が変化しない．またチューニングを施すと性能が低下するものがあるが，これはカーネルベンチで高速化される実装方式が，必ずしもフルアプリケーションで高速化されないことがあるからである．

図 6 に，COCO における実測 B/F 値をのせる．各棒グラフの上の数字はそれぞれの実測 B/F 値を示す．なお図 6 中の(simd=2)とは，コンパイラオプションによる SIMD 化の促進指示による速度向上の結果を指す．

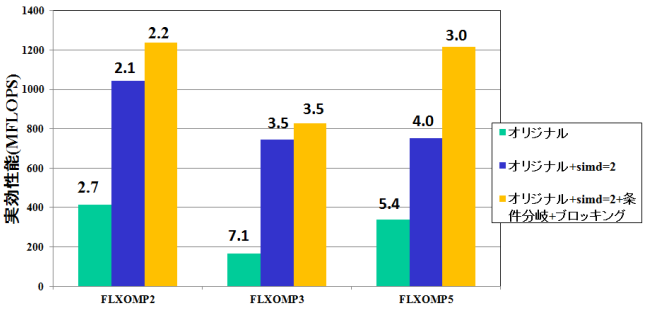

図 6 COCO における
チューニング前後での実測 B/F 値の変化

図 6 では，チューニングすると B/F 値が低下する演算カーネルもあるが（FLXOMP5），チューニングしても B/F 値は変化しないか，若干増加するものが残りすべてである．

以上の図 4～図 6 の結果から，**必ずしも B/F 値が低い演算カーネルが良いとは言えない**．B/F 値が高くても，実行時間が高速であることが重要である．この議論は，ハードウェアにおける B/F 値にもいえると予想される．必ずしも B/F 値が高い計算機が良いわけではなく，対象の計算機の特性を考慮し，実際の実行時間を評価することが必要である．そのため，演算カーネルを対象の計算機向けにきちんとチューニングすることがより重要である．

4.3 異機種環境での性能

4.3.1 FX10 と SR16K の比較

この節では，FX10 と SR16K の性能を比較することで，FX10 におけるチューニングコードの性能について検証する．なお SR16K の最適化について，FX10 で行ったチューニング手法と同様のチューニングを適用する意味で，SR16K 向けに最適化がされている．なお，FX10 と SR16K では利用するノード数が異なるが，MPI プロセス当たりの問題サイズは同等として，弱スケーリングを用いて評価する．また，理論性能比を換算して実行時間の比を求めることで，両者の実行効率を検証する．

表 1 に ALPS での結果をのせる．

表 1 ALPS での性能比較

計算機	ノード数	MPI × OMP	実行時間[sec.]	実行時間比	理論性能[GFLOPS]
FX10	16	16x16	48.8	1.00	3784.0
SR16K	1	4x16	142.3	1.32	980.48

まず本節で示す「実行時間比」について，算出方法を説明する．表 1 では，FX10 に対する SR16K の理論性能比は，

3784.0[GFLOPS] / 980.48[GFLOPS] = 3.85[倍]である．したがって SR16K において，この理論性能比を考慮した実行時間は，142.3/3.85=36.9[sec.]と考えられる．この理論性能比を考慮した SR16K での実行時間に対する，FX10 での実測の実行時間の比を，実行時間比と定義する．表 1 の例では，実行時間比は 48.8 [sec.] / 36.9 [sec.] = 1.32 となる．

表 1 から，FX10 と SR16K での実行時間比について，SR16K では 30%ほど FX10 に対して性能が良い．この原因は解析中であるが 1 つの要因は，SR16K は SMT による 64 論理コア実行により，32 物理コア実行以上の並列効率が達成できるので，そのためと推察される．また，MPI プロセス実行数も SR16K の方が少ないので，通信時間は FX10 より少ないと予想される．結論として，SR16K で 32 物理コア実行の推定性能から勘案すると，FX10 の性能は適度にチューニングされており，評価に適しているといえる．

表 2 に RSDFT での結果をのせる．

表 2 RSDFT での性能比較

計算機	ノード数	MPI × OMP	実行時間[sec.]	実行時間比	理論性能 [GFLOPS]
FX10	4	32x2	109.2	1.00	946.0
SR16K	1	32x1	69.4	1.52	980.48

表 2 から，SR16K の実行時間比が 50%ほど良い．この理由は解析中であるが，1 つの理由は，SR16K では複数ノードを利用していない実行であることから，通信時間の占める割合が FX10 よりも少なくなるからと予想される．

表 3 に NICAM での結果をのせる．

表 3 NICAM での性能比較

計算機	ノード数	MPI × SMP	実行時間[sec.]	実行時間比	理論性能 [GFLOPS]
FX10	5	5x16	281.9	1.00	1182.5
SR16K	1	5x4	608.3	0.89	612.8

表 3 から，SR16K の実行時間のほうが，FX10 よりも 10%ほど効率が悪い．この理由は解析中であるが，1 つの要因は，FX10 および SR16K でのスレッド並列実行はコンパイラによる自動並列化を利用していることがあげられる．

FX10 では，ソースコード中に富士通コンパイラ用ディレクティブが挿入されており，効率よく自動並列化がされている．一方 SR16K では，コンパイラによる自動並列化がされているものの，日立コンパイラ用ディレクティブは挿入されていない．したがって，自動並列化コードのスレッド並列実行効率の違いにより，この性能差が生じた可能性がある．

結論として FX10 の実行効率は，SR16K での実行効率と比較して良いといえるので，FX10 でのコード性能は適度にチューニングされており評価に適しているといえる．

表 4 に COCO での結果をのせる．

表 4 COCO での性能比較

計算機	ノード数	MPI × OMP	実行時間[sec.]	実行時間比	理論性能 [GFLOPS]
FX10	4	4x16	196.8	1.00	946.5
SR16K	1	4x8	179.4	1.06	980.48

表 4 から，6%ほど SR16K のほうが性能がよい．COCO の演算カーネルはメモリバンド幅拘束のため，ハードウェアのメモリアクセス性能が影響する．FX10 では 1 ソケットあたり 85GB/秒であるが，SR16K は 1 ソケット当たり 128GB/秒であるため，FX10 のほうがハードウェア上のメモリアクセス性能が悪い．したがって，このハードウェア性能差により，実行性能の差が生じたと考えられる．FX10 の方がハードウェア性能が悪いのにもかかわらず，ほぼ同等の性能が出ていると考えられる．ゆえに，チューニング済みの FX10 のコード性能は適度にチューニングされており，評価に適しているといえる．

4.3.2 異機種環境での性能比較

ここでは，異なる複数のハードウェア環境（異機種環境）での性能を評価する．本プロジェクトでのベンチマークは京もしくは FX10 で動作検証がされているものを使っているため，異機種環境では正常動作しないことがある．そこで，異機種環境で正常動作した ALPS での結果を紹介する．

この時，以下の実行条件で性能を測定した．なお，問題サイズについては今までと同じように，1 プロセス当たりの問題サイズをほぼ同じ大きさにしてある．

- FX10 及び CX400：48 ノード 48MPI×16OMP.
- 京：48 ノード 48MPI×8OMP.
- XE6：24 ノード 48MPI×16OMP.
- SR16K：1 ノード 4MPI×8OMP の結果を MPI 数の比である 1/12 で換算.

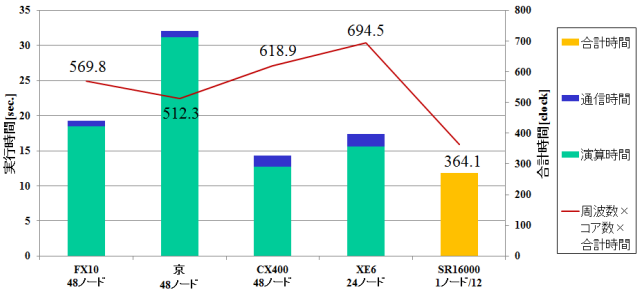

図 7　異機種環境での ALPS の性能

図 7 より，演算時間は CX400 が最短である．一方，通信時間は FX10 が最短である．

実行時間に周波数及びソケット当たりのコア数を掛けて正規化した数値（正規化値）と比べる場合，SR16K が最も高効率となった．ただし SR16K は，多数のノードで通信時間を含んだ結果ではない（1 ノードのみ）．XE6 がこの正規化値の意味で効率が悪い理由の一つとして，ALPS は整数演算が多いアプリケーションであるが，XE6 は整数 2 パイプで低 IPC のため，やや効率が低めになったことがあげられる．

図 7 より異機種環境での性能に対しても，FX10 のコード性能は適度にチューニングされており，評価に適しているといえる．

5. おわりに

本報告では，数値計算レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究での計算機設計に用いるアプリケーションにおいて，性能予測のための基礎データ取得時に必要な最適化について説明した．

本報告を通じて，以下のことが言える．性能プロファイラから得られる実測 B/F 値について，チューニング前後で検証したところ，チューニング後のコードの B/F 値は，必ずしもチューニング前の B/F 値に対して低くならなかった．場合によっては，10 倍高速化されてもチューニング後の B/F 値のほうが高い値を示すことさえある．

以上を鑑みると，アプリケーションの B/F 値のみで性能を議論することは危険である．B/F 値のみで判断せず，実行時間の観点で最適化を行い，計算機システム全体の性能を評価することが重要である．

謝辞 本研究を行うに当たり，富士通 PRIMEHPC FX10 の性能プロファイラ情報など多数のご支援をいただいた富士通社の諸氏に感謝いたします．

本研究は，文部科学省「将来の HPCI システムのあり方の調査研究」（平成 24 年度～平成 25 年度）の支援による．また本論文の結果の一部は，理化学研究所のスーパーコンピュータ「京」を利用するとともに，「京」以外の HPCI システム利用研究課題を遂行して得られたものです（課題番号:hp120128）．

参考文献

1) http://www.mext.go.jp/b_menu/houdou/24/06/1322138.htm
2) http://www.mext.go.jp/b_menu/shingi/chousa/shinkou/028/shiryo/__icsFiles/afieldfile/2012/08/14/1324574_2_1.pdf
3) http://www.mext.go.jp/b_menu/shingi/chousa/shinkou/020/shiryo/__icsFiles/afieldfile/2012/04/12/1319671_03.pdf
4) HPCI 技術ロードマップ白書，2012 年 3 月．http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf
5) 計算科学ロードマップ白書，2012 年 3 月．http://open-supercomputer.org/wp-content/uploads/2012/03/science-roadmap.pdf
6) 片桐ほか：レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究のためのアプリケーションと性能評価，情報処理学会研究報告 2012-HPC-137 (2012)
7) https://www.hpci-office.jp/summary/institution.html