将来のレイテンシコアの高度化・高効率化によるシステムについて

東京大学情報基盤センター
片桐 孝洋

日時：2013年8月27日(火) 分科会 10:30-17:20
13:55-14:25、 講演 30分 [講演3-1]
場所：汐留シティセンター 24階 富士通株式会社 大会議室

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

話の流れ

- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- 異機種環境での評価
- おわりに

話の流れ

- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- 異機種環境での評価
- おわりに

Towards Next-generation General Purpose Supercomputer

Feature of Target System:
- ✓Deployment by 2020
- ✓Power consumption 30MW
- ✓2000 m^2 constraints

Approach:
- ✓Material and Climate Sciences are the first target applications
- ✓Approach from evolution of the K architecture
- ✓System Software Stack is designed for both the proposed machine and commodity-based machines

PI: Yutaka Ishikawa, U. of Tokyo
- ➢ Organization
- ➢ System Software Stack
- ➢ Performance Prediction and Tuning

Co-PI: Yuichi Nakamura, NEC
- ➢ System Software Stack

Applications

System Software Stack
(MPI, parallel file I/O, PGAS, Batch Job Scheduler, Debugging and Tuning Tools)

Commodity-based Supercomputer

Next-Gen General Purpose Supercomputer

Co-PI: Tsuneo Iida, Hitachi
- ➢ Storage Architecture and System Software Stack

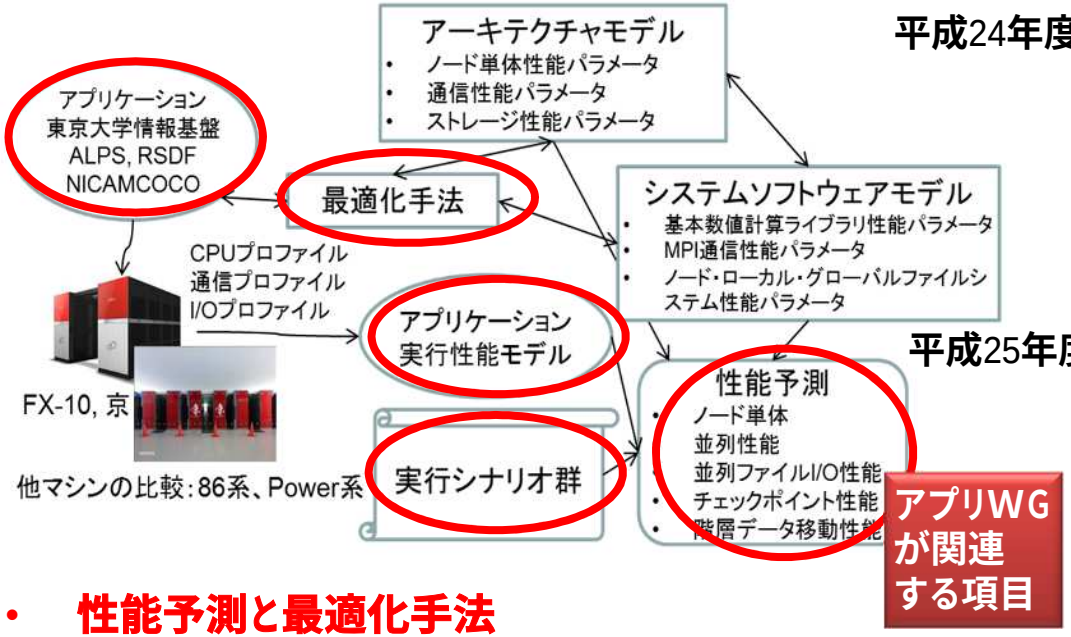
Co-PI: Kei Hiraki, U. of Tokyo
- ➢ Architecture Evaluation, Compiler, and Low power technologies

Co-PI: Mutsu Aoyagi, Kyushu U.
- ➢ Network Evaluation Environment

Co-PI: Naoki Shinjo, Fujitsu
- ➢ Processor, Node, Interconnect Architecture and System Software Stack

プロジェクト概要　と　東大アプリWG　の目的

- 2018年ごろ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ＦＳ
  - 京コンピュータ、ＦＸ１０ＣＰＵを基に、メニィコア化、SIMD化、通信機構の高度化を検討



平成24年度

	7〜9	10〜12	1〜3
システムソフトウェア	Proof of Concept 実装＆評価項目精査	Proof of Concept実装	
性能予測と最適化手法	ベンチマーク化、プロファイル測定、プロファイル検討、最適化手法検討		

平成25年度

	上半期	下半期
システムソフトウェア	Proof of Concept実装（スパイラル開発モデル）	
性能予測と最適化手法	ターゲットアプリケーション拡大検討 ベンチマーク化、プロファイル測定、プロファイル検討、最適化手法検討	

- 性能予測と最適化手法
  - ターゲットアプリケーションから抽出した演算と通信カーネルをベンチマークプログラム化
  - 東京大学情報基盤センターのFX10および京を用いて性能予測のための性能パラメータの抽出
  - 最適化手法の検討
  - 平成25年度ターゲットアプリケーション拡大のために理研AICSと連携
- システムソフトウェア
  - ヘテロOSカーネルおよび低レベル通信機構のproof of concept実装し、proof of concept実装に基づきハードウェア概念設計に反映するとともに、システムレベルの性能パラメータ（通信、ファイルI/O)を示す
    - システムソフトウェア設計では理化学研究所AICSシステムソフトウェア研究チームと連携

共同研究者(アプリ評価WG)

- アプリ最適化、性能評価
  - 大島聡史、中島研吾 (東大基盤センター)
  - 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) 、黒木聖夫 (海洋研究開発機構)
- アプリ開発者
  - 藤堂眞治、岩田潤一、内田和之、佐藤正樹、羽角博康 (東大)
- 概念設計マシン評価連携
  - 富士通社の皆様

Co-designの進め方

重要ループコード

詳細プロファイラ7回実行で
performance counter取得

詳細プロファイラの出力結果(csvファイル)

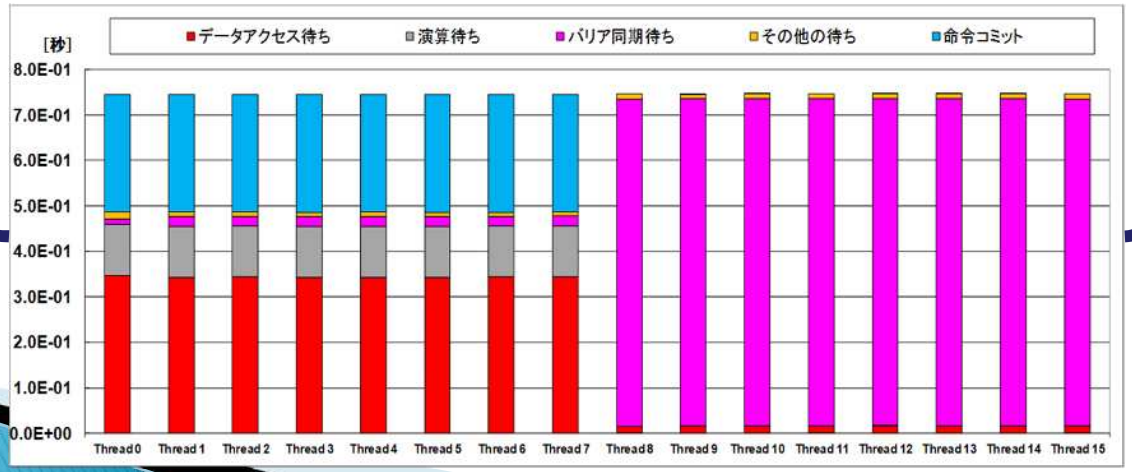
アーキテクチャ概念設計
プロセッサ・ノード・
インターコネクト・
階層化ストレージ

アプリケーション
評価・チューニング

性能予測ツール

ターゲットCPUパラメータ
(クロック、SIMD幅、
各種バンド幅)

性能予測値

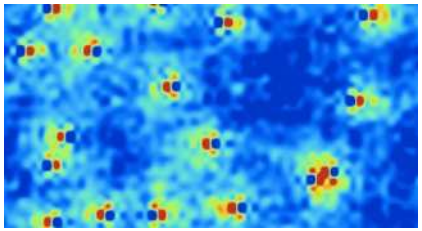


話の流れ

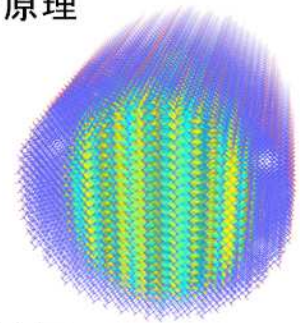
- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- 異機種環境での評価
- おわりに

ターゲットアプリケーションと要求性能

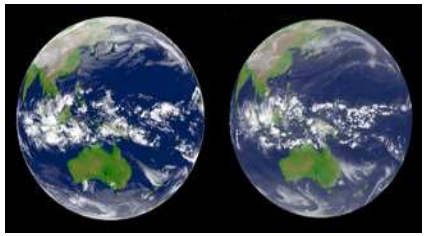
- **ALPS/looper**
  - 新機能を持った強相関･磁性材料の物性予測･解明。虚時間経路積分にもとづく量子モンテカルロ法と厳密対角化
  - 総メモリ：10～100PB
  - 整数演算、低レイテンシ、高次元のネットワーク
  - 利用シナリオ：1ジョブ当たり24時間、生成ファイル：10GB. 同時実行1000ジョブ、合計生成ファイル：10TB.



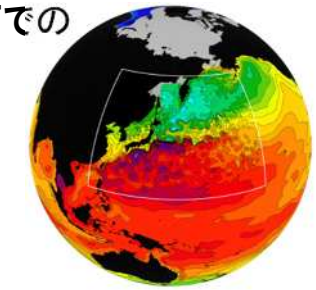
- **RSDFT**
  - Siナノワイヤ等、次世代デバイスの根幹材料の量子力学的第一原理シミュレーション。実空間差分法
  - 総メモリ：1PB
  - 演算性能：1EFLOPS (B/F =0.1以上)
  - 利用シナリオ：1ジョブ当たり10時間、生成ファイル：500TB. 同時実行10ジョブ、合計生成ファイル5 PB.



- **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.



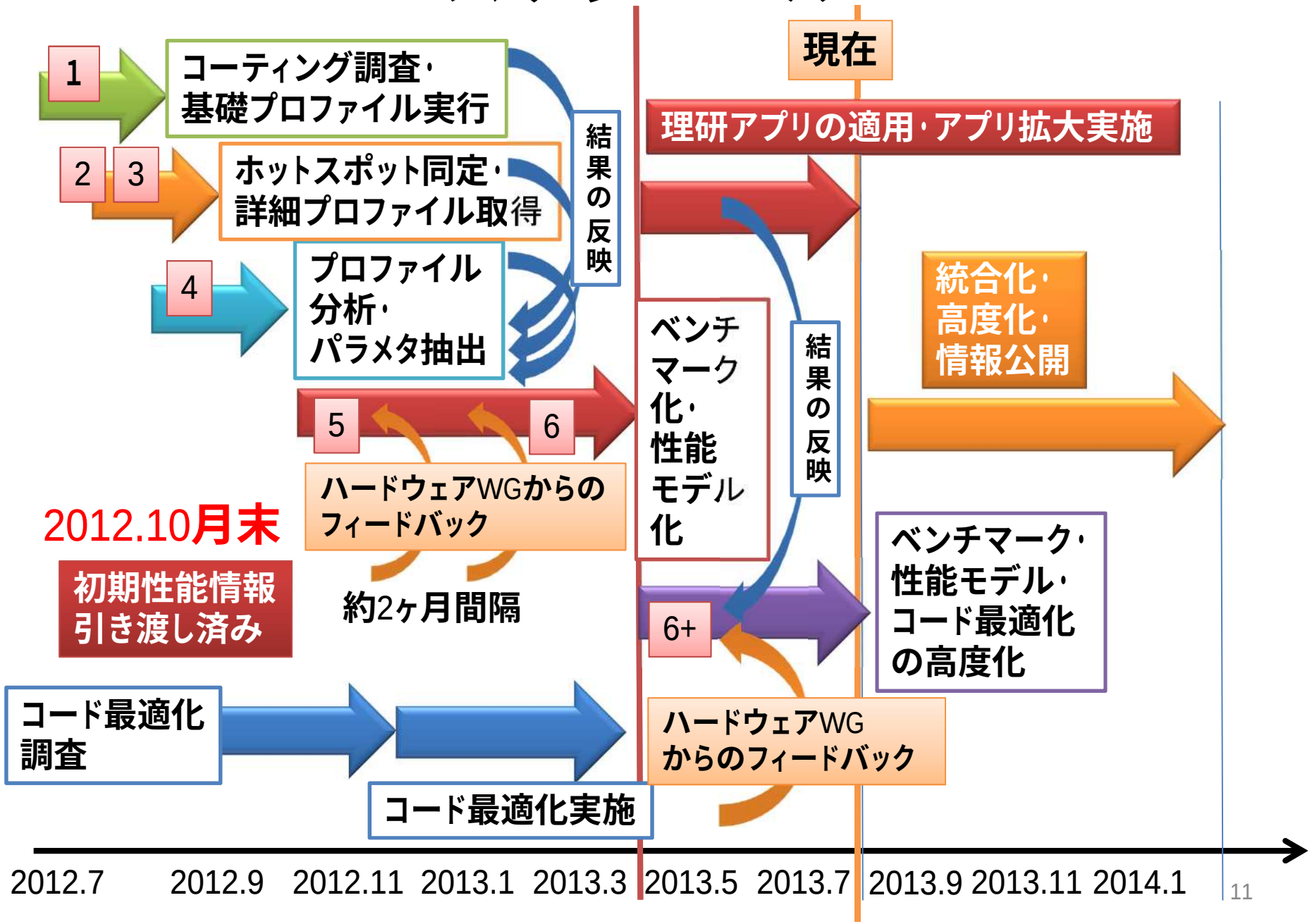
利用シナリオ
アンサンブル型
全系の1/10～1/100資源を用いた1ジョブを、複数同時実行することで、全資源を使い切る形態。

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

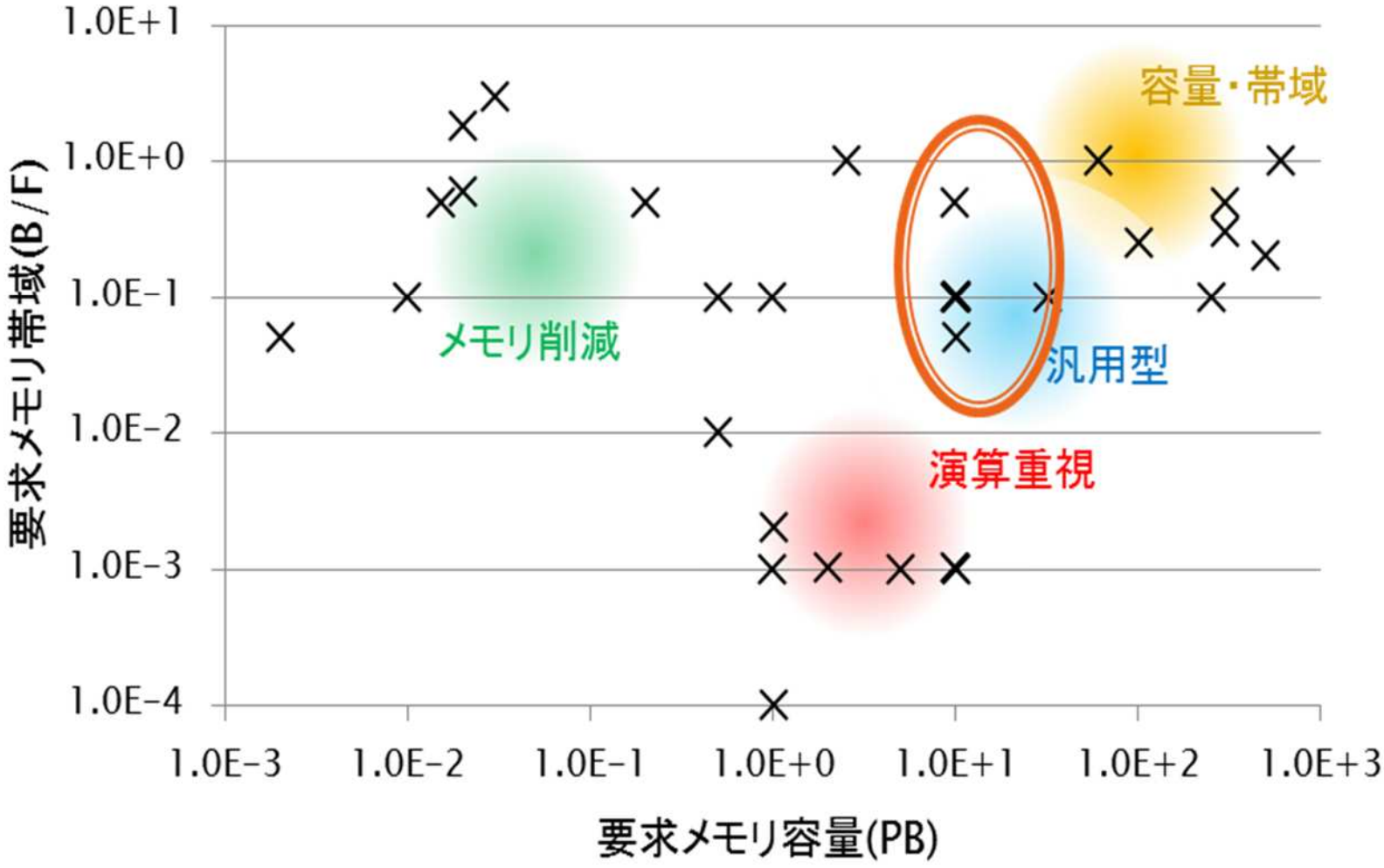
1. **ホットスポット同定**：**富士通社の**基本プロファイラで複数の**ホットスポット**（ループレベル）を同定、全体性能の予測をホットスポットのみで行う
   - ホットスポットの部品化
2. **カーネル分離**：（目視により）計算部分、通信部分、I/O部分の分離
   - 計算部分：**演算カーネル**
   - 通信部分：**通信カーネル**
   - I/O部分：**I/Oカーネル**
3. **通信パターン確認**
4. **詳細プロファイルと分析**：　**富士通社の**詳細プロファイラを用い、ホットスポットごとにハードウェア性能情報（**＝性能パラメタ**）を取得し分析
   - **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**：ハードウェア因子による実行時間を近似。
   - 富士通社の**性能予測ツール（FX10の性能プロファイルデータを基にする）**により、概念設計マシンでの実行時間を予測

スケジュール


現在
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
理研アプリの適用・アプリ拡大実施
ベンチ
マーク
化・
性能
モデル
化
結果の反映
統合化・
高度化・
情報公開
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

アプリケーションの特徴

東大FSのアプリ要求性能

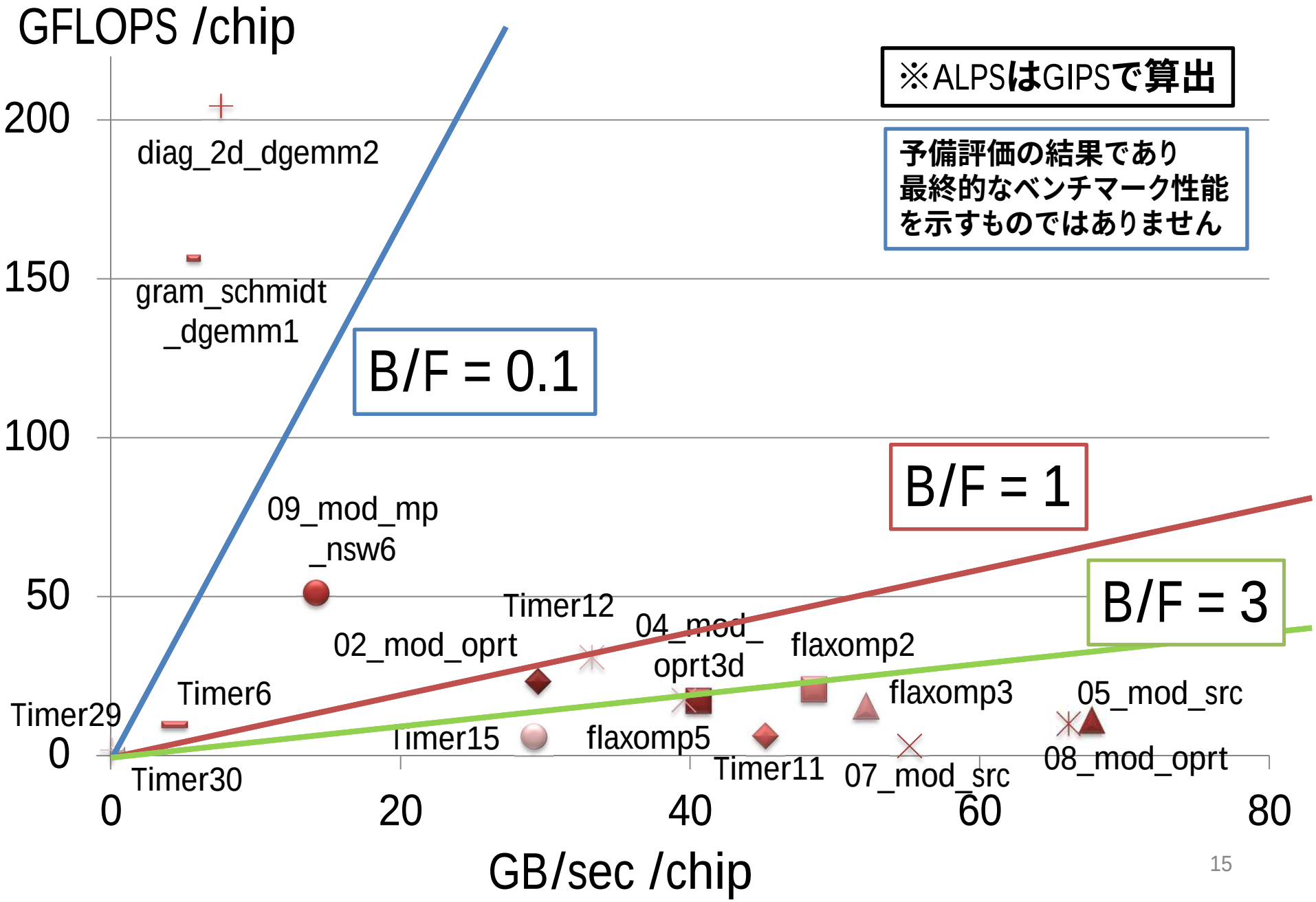


出典：今後のHPCI技術開発に関する報告書p.13から

アプリケーション特性のまとめ

アプリ名	演算カーネル			通信カーネル	
	演算種類	配列アクセスパターン	特徴	通信パターン	特徴
ALPS/ Looper	整数	探索	IF文成立は事前予測不可能	規則通信	バタフライ通信
RSDFT	浮動小数点	密行列-行列積	● 固有値ソルバー（対称実数） ● Gramm-Schmidt 直交化	同時集団通信	● 2Dプロセスグリッド ● 行／列方向の同時集団通信 ● MPI_Bcast と MPI_AllReduce
NICAM	浮動小数点	ステンシル演算	● 力学過程（最内IF文なし、ステンシル演算） ● 物理過程（ループ内演算多数）	隣接通信	主に6方向、最大で15方向
COCO	浮動小数点	ステンシル演算	● 最内IF文 ● 同時参照配列多数	隣接通信	最大で4方向

演算カーネルの実測B/F値：最適化後



演算カーネルの性能(ＡＬＰＳ)
最適化後 (FX10、384MPI実行)

予備評価の結果であり
最終的なベンチマーク性能
を示すものではありません

カーネル名	スレッド数	GIPS/chip	% to Peak	SIMD %	Memory Throughput (GB/sec)/chip	% to Peak	B/I	Time [s]	% to Total Time
		1.63x			1.75x			2.09x	
全体	16	7.61	6.43	0.19	11.58	13.62	1.52	33.88	100
Timer6	16	9.69		0.04	4.37	5.14	0.45	4.46	13.16
Timer11	16	6.28		1.43	45.21	53.19	7.20	2.48	7.32
		4.91x			13.9x			13.9x	
Timer12	16	31.06		0.00	33.21	39.07	1.07	1.39	4.10
Timer15	16	6.07		0.00	29.23	34.39	4.82	0.58	1.71
Timer29	16	1.82		0.00	0.07	0.08	0.04	0.58	1.71
Timer30	16	1.75		0.00	0.05	0.06	0.03	1.66	4.90

合計：32.9%　4.68x

演算カーネルの性能(RSDFT)
最適化後 (FX10、96MPI実行)

予備評価の結果であり
最終的なベンチマーク性能
を示すものではありません

カーネル名	スレッド数	GFLOPS/chip	% to Peak	SIMD%	Memory Throughput(GB/sec)/chip	% to Peak	B/F	Time [s] 1.10x	% to Total Time	備考
全体	16	90.04	38.07	73.31	6.52	7.67	0.07	637.87	100	
diag_2d_dgemm1	16	185.12	78.28	81.26	9.09	10.69	0.05	46.53	7.29	固有値ソルバー
diag_2d_dgemm2	16	204.38	86.43	81.19	7.58	8.92	0.04	91.96	14.42	固有値ソルバー
gram_schmidt_sub_blkcyc_dgemm1	16	156.55	66.20	76.78	5.32	6.26	0.03	90.07	14.12	直交化
gram_schmidt_sub_blkcyc_dgemm2	16	191.24	80.87	78.21	6.04	7.11	0.03	73.75	11.56	直交化

合計：47.3%

演算カーネルの性能(NICAM)：
最適化後 (FX10、160MPI実行)

予備評価の結果であり、最終的なベンチマーク性能を示すものではありません

計算種別	カーネル名	スレッド数	GFLOPS/chip	% to Peak	SIMD%	Memory Throughput (GB/sec)/chip	% to Peak	B/F	Time [s]	% to Total Time	備考
	全体	16	11.09	4.69	61.04	34.37	40.43	3.10	173.52	100	160MPI実行
力学過程	02_mod_oprt (oprt_divergence)	16	23.3	9.85	45.95	25.06	29.48	1.08	4.84	2.79	IF文なし、ステンシル
	04_mod_oprt3d (oprt3d_divdamp)	16	17.19	7.27	78.83	40.57	47.73	2.36	4.14	2.39	IF文なし、ステンシル、複数配列参照
	05_mod_src (src_flux_convergence_PRL_8_)	16	11.08	4.69	83.22	67.69	79.64	6.11	3.13	1.80	IFなし、ステンシル、配列初期化あり
	07_mod_src (src_flux_convergence_PRL_17_)	16	3.05	1.29	79.84	55.16	64.89	18.09	1.91	1.10	IF文なし、内部演算少ないループが複数
	08_mod_oprt (oprt_divergence2_rev)	16	10.08	4.26	60.82	66.14	77.81	6.56	2.26	1.30	IFなし、複数配列参照
物理過程	09_mod_mp_nsw6	16	51.2	21.65	71.64	14.16	16.66	0.28	1.51	0.75	IF文あり、最適化済み

1.15x

合計：10.25%

演算カーネルの性能(COCO)
：最適化後 (FX10、192MPI実行)

予備評価の結果であり
最終的なベンチマーク性能
を示すものではありません

カーネル名	スレッド数	GFLOPS /chip	% to Peak	SIMD%	Memory Throughput (GB/sec) /chip	% to Peak	B/F	Time[s]	% to Total Time	備考
								1.08x		
全体	16	1.57	0.67	29.21	4.11	4.84	2.61	269.72	100	
		2.47x			2.25x			2.29x		
flxomp2	16	20.94	8.85	37.33	48.51	57.07	2.32	4.90	1.82	IF文あり
		4.53x			2.19x			2.19x		
flxomp3	16	15.47	6.54	57.15	52.12	61.32	3.37	3.61	1.34	IF文あり
		3.34x						2.21x		
flxomp5	16	17.41	7.36	53.70	39.58	46.56	2.27	4.95	1.84	IF文あり、ブロック化

※I/Oのための通信(MPI_Allgather)が入っている。エクサ環境向きコードではI/O方式を変更するため、カーネル実行時間の比率は、この値より高くなる見込み。(予想：50%以上)

合計：4.99%

話の流れ

- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- 異機種環境での評価
- おわりに

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	

Many many
More to come

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、
概念設計マシンにおける
性能予測のための基礎データ
を妥当にするための作業

ALPSの事例

ALPS: Timer12のチューニング（1／3）

オリジナルコード（京コンピュータ上では問題なし）

```
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。

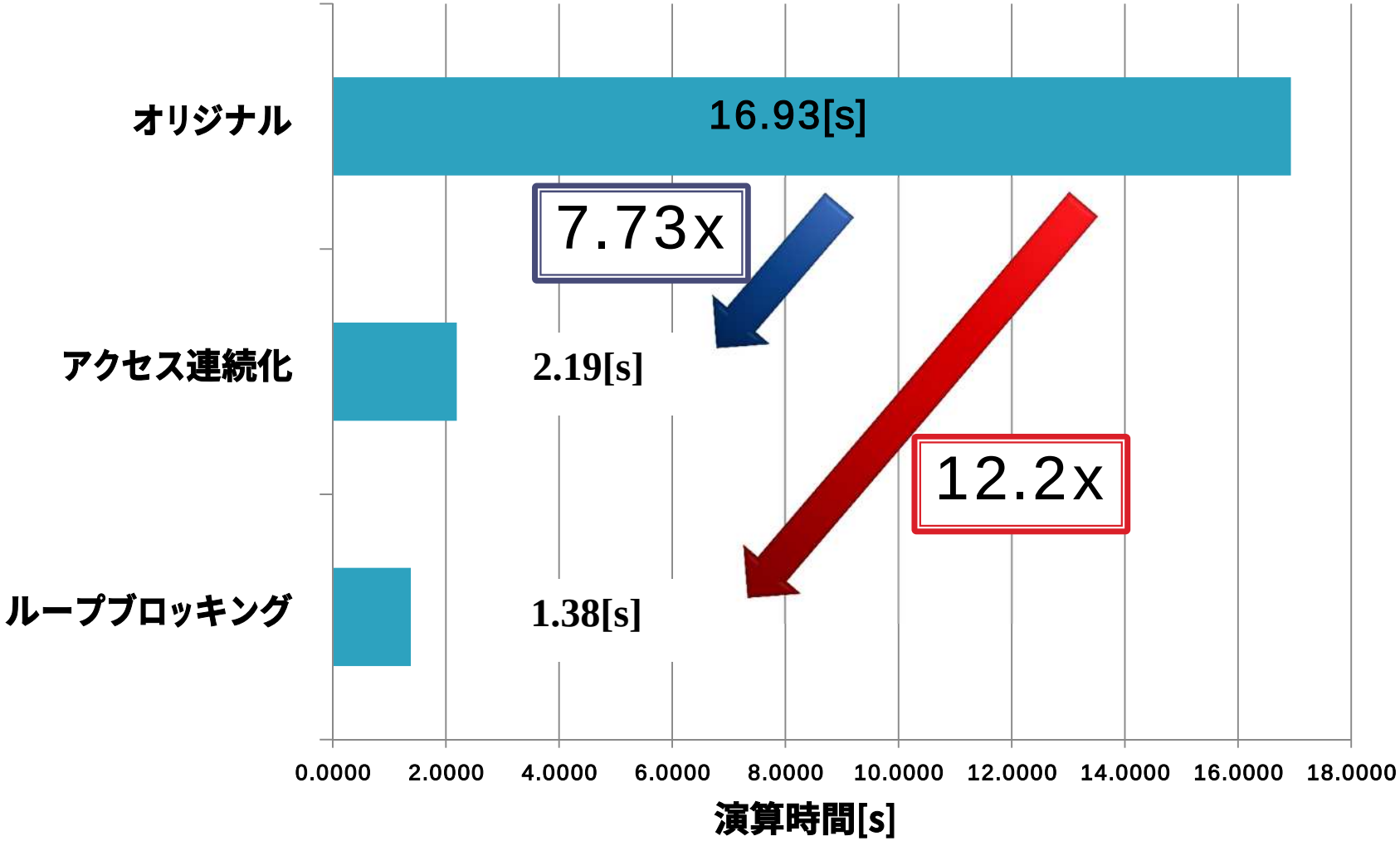
ALPS: Timer12のチューニング（3／3）

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

NICAMの最適化(3/6) -mod_oprt3d_4

●最適化①：kで4展開

->ループ展開の指定

```
!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)&
    ) * 0.5D0 * VMTR_GSGAMH(ij,k,l) * VMTR_GZXH(ij,k,l)&
   + ( GRD_afac(k) * VMTR_RGSGAM2(ij,k,  l) * rhovy_in(ij,k,  l)&
   + GRD_bfac(k) * VMTR_RGSGAM2(ij,k-1,l) * rhovy_in(ij,k-
1,l)&
    ) * 0.5D0 * VMTR_GSGAMH(ij,k,l) * VMTR_GZYH(ij,k,l)&
   + ( GRD_afac(k) * VMTR_RGSGAM2(ij,k,  l) * rhovz_in(ij,k,  l)&
   + GRD_bfac(k) * VMTR_RGSGAM2(ij,k-1,l) * rhovz_in(ij,k-
1,l)&
    ) * 0.5D0 * VMTR_GSGAMH(ij,k,l) * VMTR_GZZH(ij,k,l)&
    ) + rhow(ij,k,l) * VMTR_RGSH(ij,k,l)
    enddo
  enddo
```

●最適化②：nのループ融合

```
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
        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
```

2文を削除して
ループ融合

- kのループで4展開で、Byte/Flopが4.86から4.0に減少
- nのループ融合で、2ループの同一配列のロードを削減、Byte/Flopが5.41から3.78に減少
- ①と②の効果で1.20倍性能が向上

NICAMの最適化(4/6) -mod_src5

●最適化：キャッシュブロック化

```
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
   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
```

キャッシュブロックのループibを追加

ijのアクセス範囲を
キャッシュブロック長に変更

- rhogvx、rhogvy, rhogvz はk,k-1を参照
  ijのキャッシュブロック化でキャッシュヒット率向上の効果がある
- ijループのblock化で10%性能向上

NICAMの最適化(6/6) -mod_oprt8

●最適化：ループ2分割、8展開

最適化前

```
do k = 1, ADM_kall
  nstart = suf(ADM_gmin-1,ADM_gmin-1 )
  nend  = suf(ADM_gmax, ADM_gmax )
  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_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)
=(0.5D0+sign(0.5D0,c(1,n,k,l)))*sa_p+ &
      (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
```

最適化後

```
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
```

- nで8展開してByte/Flopを減らす
- 8展開でスピルコードが出るため、sa_pの式とsa_mの式でループを分割
- この最適化で1.36倍性能向上

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

ここでの最適化前は、コンパイラオプションの最適化、
ブロック化（タイリング）サイズの調整済みの
準最適化されたコード

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

COCOの最適化(3/3) - flxtrc._OMP_5

●最適化：ループ融合

最適化前

```
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
```

ループ#1

```
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#2

最適化後

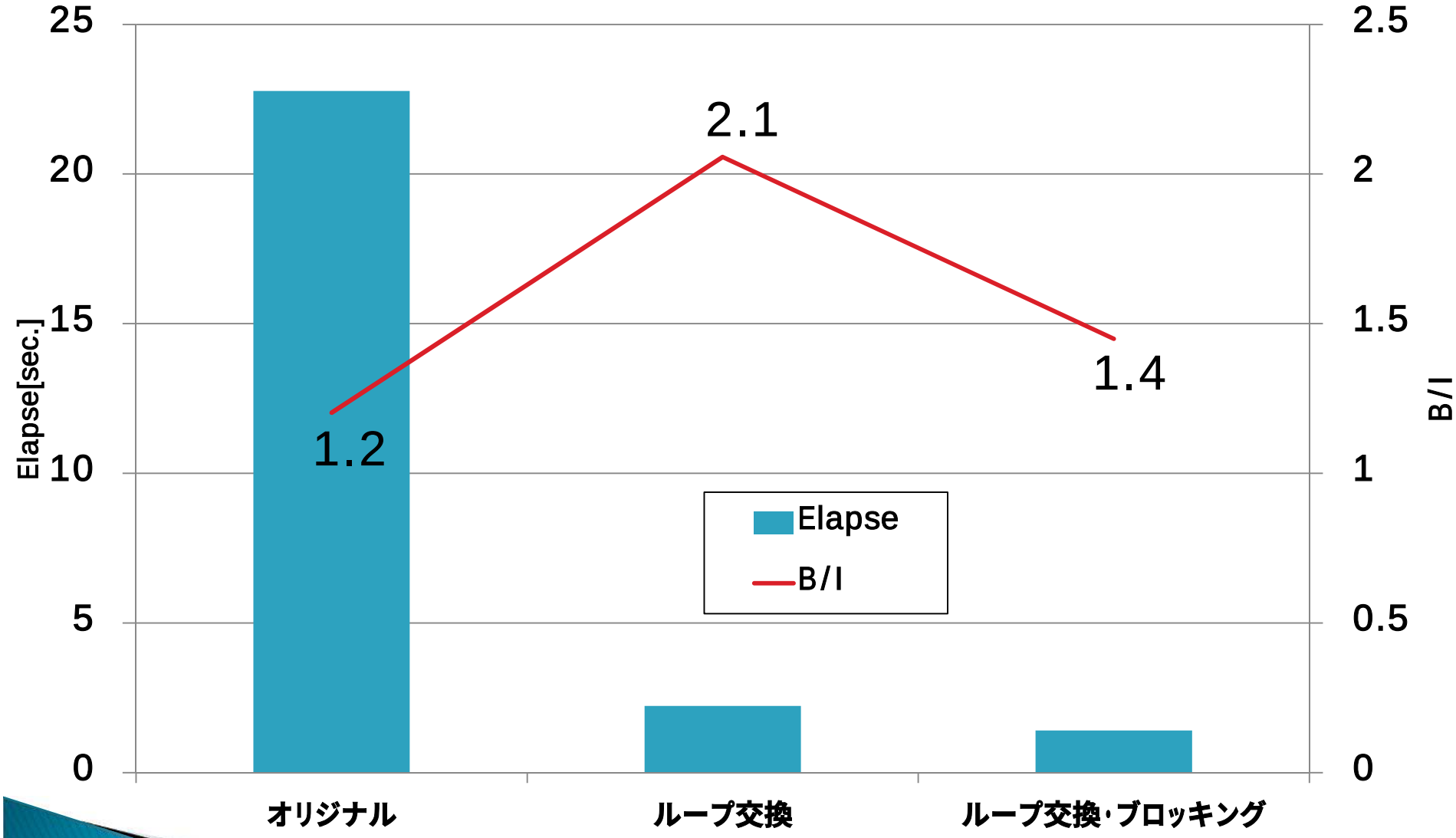
```
DO K = KSTR, KEND
  DO IJ = IJ1, IJ2
      IF ( UV(IJ, K) .GT. 0.D0 ) THEN
        ZF0(IJ-IJ1+1, KU) =

        FTZ (IJ, K, N) =
        SM (IJ, KU, N) =

        SXY(IJ, KU, N) =
      ELSE
        ZF0(IJ-IJ1+1, KU) =

        FTZ (IJ, K, N) =
        SM (IJ, KU, N) =

        SXY(IJ, KU, N) =
      END IF
  END DO
END DO
```

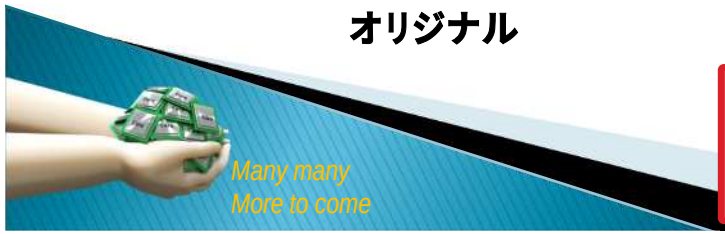
ループ#1 #2を融合

- 似た構造のループを融合
- ループ融合で1.12倍性能向上

実測B/F値の変化

ALPSの実行時間評価 (最適化効果(Timer12))

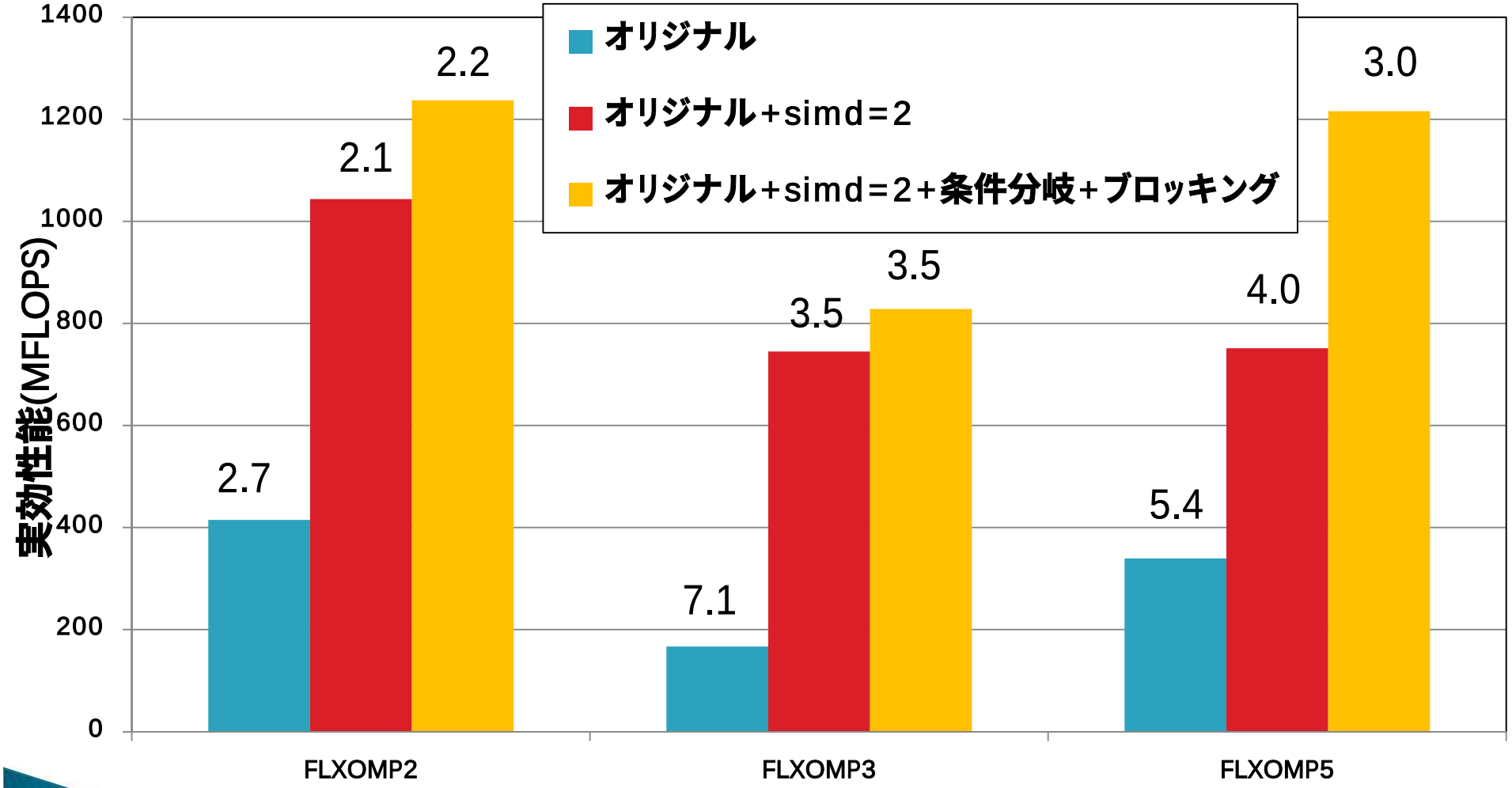




実効性能が高くなっても, Byte/Instruction値が低下していない場合がある

COCOの実効性能とByte/Flop

●COCOの主要ループに最適化を適用したときの実効性能とByte/Flop値



実効性能が高くなっても，Byte/Flop値が低下していない場合がある

話の流れ

- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- **異機種環境での評価**
- おわりに

検証事項

- FX10のチューニング済み性能が妥当か？
- FX10とは異なるプロセッサアーキテクチャでの性能差はどれくらいか？

性能評価環境

●FX10 スーパーコンピュータシステム (CPU: Sparc64 IV-fx)

OS	XTCOS
計算ノード数	4800
1ノード理論性能	236.5GFlops
総理論演算性能	1.13PFLOPS
1ノード主記憶容量	150TB
インターコネクト	6次元メッシュ/トーラス
ノード間ネットワーク性能	5GB/s×双方向

B/F=0.32

B/F比率では
1.44倍、
SR16Kの
ほうが良い

●HITACHI SR16000/モデルM1 (CPU: Power7)

OS	AIX7.1
計算ノード数	56
1ノード理論性能	980.48GFLOPS
総理論演算性能	54.906TFLOPS
1ノード主記憶容量	200 GB
ノード間ネットワーク性能	96GB/s (単方向) ×双方向

B/F=0.52

FX10に行った
同方式の
チューニングを
SR16Kでも
実施している

測定条件

●FX10とSR16000の性能を、同一データで比較
●ＳＲ１６０００の1ノード (メモリ上限200GB)で動作するようデータを縮小

ＳＲ１６０００とFX10の比較用データ

アプリ名	データ
ALPS/looper	実行パラメータ変更なし -l 262144 -t 0.00083 -n 16 -m 16
RSDFT	原子数1000、MB=2200に縮小
NICAM	5ノード用のデータを使用
COCO	メッシュ数を1440x1320x66から720x660x50に縮小し、ステップ数720

全体性能比較　–性能比較 (1/4)

ALPS/looperの性能比較

計算機	ノード数	MPI×OMP	実行時間(s)	実行効率比	理論性能 (GFLOPS)
FX10	16	16×16	48.8	1.00	3784.0
SR16000	1	4×16	142.3	1.32	980.48

$$\text{実行時間比} = \frac{[\text{FX10の実行時間}] \ / \ [\text{FX10の理論性能}]}{[\text{SR16Kの実行時間}] \ / \ [\text{SR16Kの理論性能}]}$$

SR16Kのほうが30%ほど効率が良い

理由 (解析中)

- 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。そのため、ハードウェアの性能差が出た？
- 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？

- 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

全体性能比較　–性能比較 (2/4)

RSDFTの性能比較

計算機	ノード数	MPI×OMP	実行時間(s)	実行効率比	理論性能(GFLOPS)
FX10	4	32×2	109.2	1.00	946.0
SR16000	1	32×1	69.4	1.52	980.48

SR16K のほうが50%ほど良い

理由 (解析中)

- RSDFTは通信時間の占める割合が高いアプリ
- 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？

- 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

全体性能比較　–性能比較 (3/4)

NICAMの性能比較

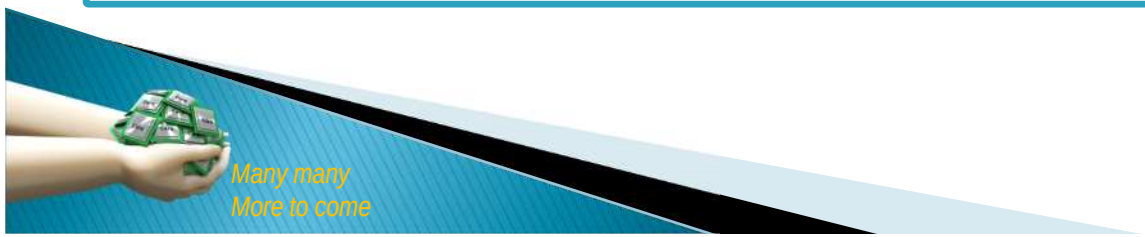
計算機	ノード数	MPI×SMP	実行時間(s)	実行効率比	理論性能 (GFLOPS)
FX10	5	5×16	281.9	1.00	1182.5
SR16000	1	5×4	608.3	0.89	612.8

SR16K の実行時間のほうが10%ほど効率が悪い

理由 (解析中)

- 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
- FX10は、自動並列化のためのディレクティブが入っている
- 自動並列化コードの効率差が出た？

- 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



全体性能比較　–性能比較 (4/4)

COCOの性能比較

計算機	ノード数	MPI×OMP	実行時間(s)	実行効率比	理論性能 (GFLOPS)
FX10	4	4×16	196.8	1.00	946.0
SR16000	1	4×8	179.4	1.06	980.48

SR16K の実行時間のほうが6%ほど効率が良い

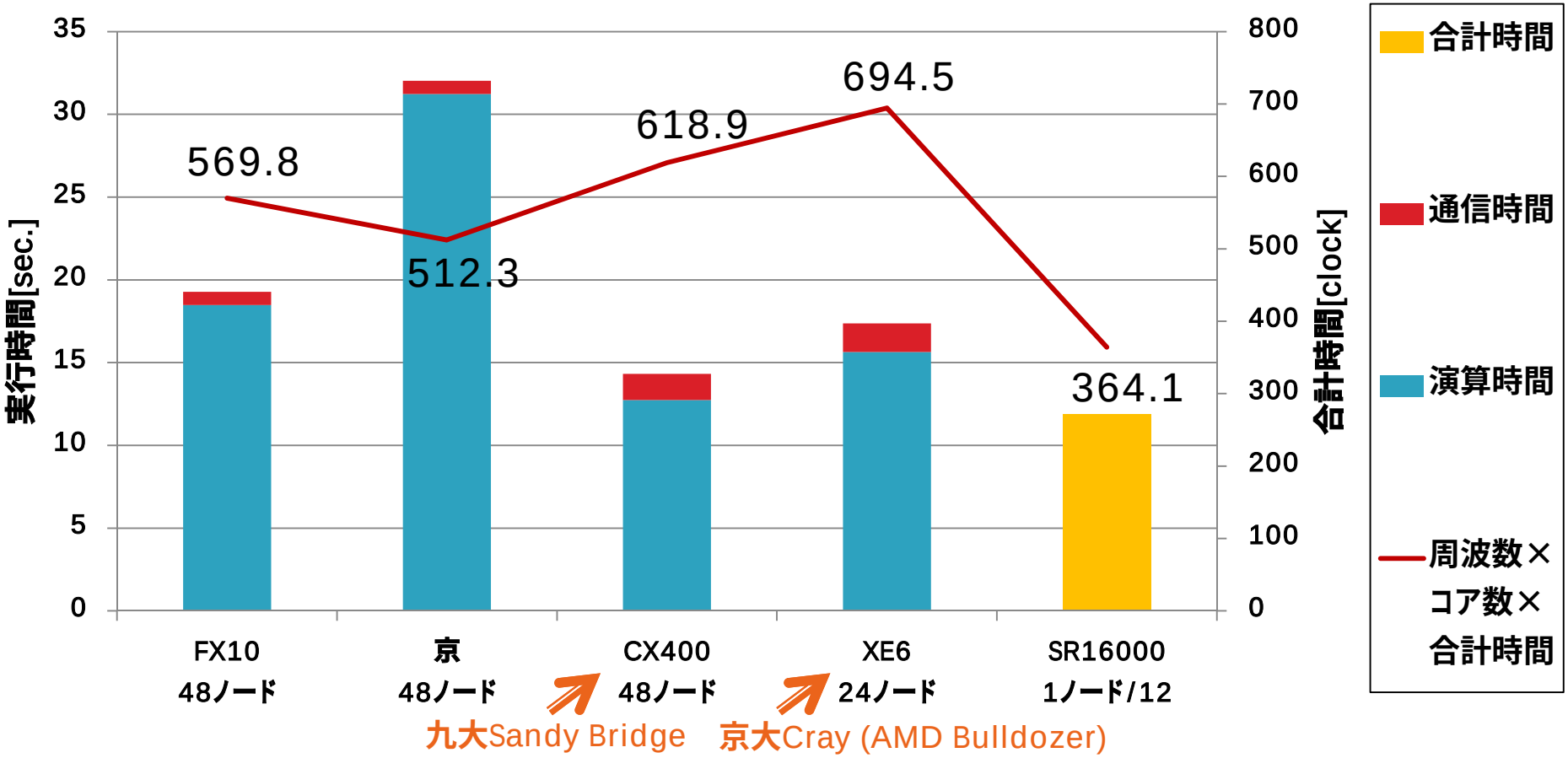
理由 (解析中)

- FX10 では1 ソケットあたり85GB/秒 (B/F=0.36)
- SR16K は1 ソケット当たり128GB/秒 (B/F=0.52)
- 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？

- 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

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 (ただし1ノードのため、通信は有利になっている)
- XE6は整数2パイプ低IPC (Instructions Per Cycle) の為、若干効率が低め

話の流れ

- プロジェクト概要
- 対象アプリケーションと性能モデル化手法
- FX10による性能チューニング
- 異機種環境での評価
- おわりに

おわりに (1/2)

- 概念設計マシンに対するコ･デザイン
  - アプリケーションの要求性能、エクサマシンにおける実行形態 (実行モデル) を、計算科学者にヒアリングの上で策定
  - アンサンブル型の実行を考慮
  - エクサマシンの実行モデルによる性能予測結果をもとに、概念設計に反映
    - 計算機アーキテクチャの構成
    - I/O性能
    - コード最適化
    - 従来アルゴリズムの限界の示唆と新規アルゴリズム開発

おわりに(2/2)

- **チューニング前後のB/F値 (B/I値)**
  - 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い
  - 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/I 値のほうが高い

- アプリケーションをB/F値のみで性能を議論することは危険
- 演算カーネルの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として性能を評価することが重要

今後の予定

- **評価アプリケーションの拡張**
  - 理研FSグループと連携しH25年度の評価アプリを選定（4種）
- **チューニング方法論の確立**
  - アプリケーションの分類手法の確立
    - CS的観点（メモリアクセス、通信のパターン）での特徴
  - 効果あるチューニング手法の系統化
  - コンパイラ最適化を補助する汎用的なチューニング機構（自動チューニング技術）の応用
- **エクサ向きのアルゴリズム開発と評価**
  - 概念設計マシンによる性能予測の結果、実行効率が悪いアプリケーションについては、新規アルゴリズムの開発検討
    - **通信回避アルゴリズム（直交化処理、Communication Avoiding QR）を、RSDFTの直交化処理に適用評価を予定**