システム評価： ミニアプリ

丸山直也、鈴木惣一朗

第7回アプリFSミーティング

２０１３年７月９日

目次

1. アプリケーション受け付け
   - 6月末〆切
   - 新規4本受け付け

2. アプリ調査＆ミニアプリ化現状

3. ミニアプリ利用事例

フルアプリ リスト #1/2

feram	西松@東北大	強誘電体MD(長距離相互作用する擬スピン系)	OpenMP(MPI使用はパラメータ並列版のみ), 3次元FFT	
MARBLE	池口@横市大	生体高分子MD (クーロン力はPME)	MPI+OpenMP, 3次元FFT	◎
para-TCCI	石村@分子研	電子状態密度計算 (Hartree-Fock計算)	MPI+OpenMP, 密行列対角化を逐次計算(並列化予定なし)	
FFVC	小野@AICS /東大	差分非圧縮熱流体 (直交等間隔格子)	MPI+OpenMP, SOR or GMRES	◎
pSpatiocyte	岩本@QBIC	細胞内シグナル伝搬計算 (反応拡散系)	MPI+OpenMP, 六方細密充填格子上のモンテカルロ, 反応過程は未実装	
NEURON_K+	加沢@東大	神経回路シミュレーション (NEURONのカスタマイズ)	MPI+OpenMP, ALL_GATHER多用, 専用のモデル記述言語からCへのトランスレータ	
GT5D	井戸村@JAEA	5次元プラズマ乱流 (5次元差分+2次元FEM)	MPI+OpenMP, 共役残差法, 1次元FFT	
MODYLAS	安藤@名大	汎用古典MD (クーロン力はFMM)	MPI+OpenMP	◎
STATE	稲垣@阪大	第一原理MD (密度汎関数法)	MPI+OpenMP(レプリカ並列、k点並列,バンドor平面波並列), FFT, 固有値計算(RMM)	
FrontFlow/blue	山出@みずほ	有限要素法非圧縮熱流体	MPI+自動並列(京対応版), BiCGSTAB	
SiGN-L1	玉田@東大	遺伝子ネットワーク推定 (L1正則化法)	MPI+OpenMP, ファイル出力が律速	
NTChem/RI-MP2	河東田@AICS	電子相関計算 (分子軌道法)	MPI+OpenMP, DGEMM, 逐次計算部分が残っている, メモリ量O(N^3)	△

(ミニアプリ化: ◎=完了、〇=作業中、△=作業開始予定)

フルアプリ リスト #2/2

OpenFMO	稲富@九大	Hartree-Fock法を基にしたFMO第一原理計算	MPI+OpenMP, 動的負荷分散	
CONQUEST	宮崎@物材研	密度汎関数法による第一原理計算、O(N)法	MPI+OpenMP 疎行列x疎行列, FFT	
NGS Analyzer	玉田@東大	次世代シークエンサーの出力データ解析	MPI I/O律速	○
NICAM-DC	八代@AICS	全球雲解像モデルNICAMの力学コア部（有限体積法)	MPI 2～4段ルンゲクッタ	△
DCPAM	西澤@AICS	全球気象・気候シミュレーション スペクトル法(球面調和関数展開)	MPI+OpenMP リープフロッグ	△
FFVC-LPT	小野@AICS	流体解析＋粒子追跡練成、 流体解析部分はFFVC	MPI+OpenMP	

(ミニアプリ化: ◎=完了、○=作業中、△=作業開始予定)

ミニアプリ(開発者側でミニアプリ化) リスト

fft_check	西松@東北大	3次元FFTベンチマーク	
ALPS/looper	藤堂@東大	量子モンテカルロ法、リンクリスト操作、整数演算、MPI+OpenMP	藤堂先生に整備依頼予定
CCS QCD Solver Benchmark test program	石川@広大	格子QCDのClover型フェルミオンの伝搬関数を求めるソルバーのベンチマーク、 疎行列連立一次方程式、BiCGStab	アプリFS側での整備作業完了
ZZ-EFSI	杉山@東大	固体流体練成コードのホットスポット、MPI+OpenMP	
rmcsm bench nocore	清水@東大	モンテカルロ殻模型計算のベンチマークコード、OpenMP	
GCEED	信定@分子研	時間依存密度汎関数法、 時間依存コーンシャム方程式の差分解法における複数のホットスポットを統合	
？	今田@東大	多変数変分モンテカルロ法	未提供
RSGDX	兵藤@JAMSTEC	地震発生サイクルシミュレーション	未提供
Barista	坂下@東大	スピン系ハミルトニアン行列の厳密対角化計算	未提供

進捗現状

- ミニアプリ化第一版完成
  - MDミニアプリ
    - MARBLE
    - MODYLAS
  - QCDミニアプリ
    - CCS-QCD
  - 流体ミニアプリ
    - FFVC（最終確認中） → 東大FSチーム近日提供予定
- 作業中
  - シーケンスマッチング
    - NGS Analyzer → 東大FSチーム８月末提供予定
  - 全体性能調査

FFVC_MINI
FFV-Cミニアプリ

FFV-C (FrontFlow/Violet Cartesian)

- 東大生産技術研究所で開発中の三次元非定常非圧縮熱流体ソルバー
- 直交等間隔格子、有限体積法、Fractional Step法
- 設定可能な境界条件
  - 外部境界条件: 計算領域の周囲
  - 内部境界条件: 計算領域の内部

    固体境界、沸きだし/吸い込み口の他に、熱交換器のモデル化などに対応可能な圧力損失境界条件も
- ロードストア量の削減

  各セルでの媒質情報，境界条件情報などを、ビット単位のフラグとして32ビット整数内に圧縮格納
  - Intelコアではピーク性能の50%
  - 京/FX10では、10～20%？

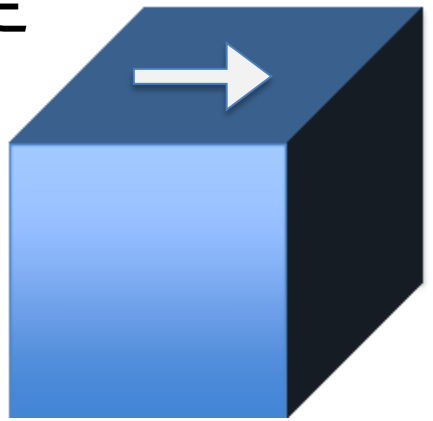
プログラム制御構造
（Euler陽解法の場合）

```
時間ステップループ {

    - 対流項，拡散項計算
    - Poisson方程式ソース項計算

    V-P反復ループ {

        Poisson反復ループ {
            - SORコア計算
        }

        - 速度更新

        /*
        - 圧力損失境界条件によるPoissonソース項の更新
          （ミニアプリでは削除）
        */
    }
}
```

オリジナルコードからの変更点

- 三次元キャビティフローのベンチマーク計算(組み込み例題)に特化
  - 使用する境界条件は外部境界のみ
  - ただし、制御構造(V-P反復とPoisson反復の二重ループ)はオリジナルコードのまま
- 使用アルゴリズムを固定
  - 時間発展: Euler陽解法
  - 圧力Poisson方程式: ストライドアクセス型2色SOR
- コマンドライン引数による計算条件指定
- 設定ファイル、入力データは不要
- コードサイズ

  10万行 → 1万行 (sloccountによる)



コマンドライン引数 #1/3

計算サイズの指定

--scale=str　サイズ指定モードをstrongまたはweakで指定
--size=int　計算領域サイズを整数(一辺のセル数)で指定
--division=str　領域分割をLxMxN形式で指定

- Strong Scaling

  --scale=strong --size=128 --division=2x2x2

  128x128x128セルを全計算領域として、それを2x2x2に分割して、8プロセスで計算

- Weak Scaling

  --scale=weak --size=128 --division=2x2x2

  128x128x128セルの計算領域を基本単位として、それを2x2x2に積み上げた領域を、8プロセスで計算

コマンドライン引数 #2/3

反復回数の制御

--step=int　　計算ステップ数を整数で指定 (ディフォルト 20)
--p_itr=int　　Poisson反復の最大反復回数 (ディフォルト 30)
--vp_itr=int　　V-P反復の最大反復回数 (ディフォルト 20)
--practical　　practical計算フラグ (ディフォルト 未指定)

- 性能測定モード(ディフォルト)

  Poisson反復、V-P反復とも、収束状況によらず常に最大反復回数の反復を行う
- Practical計算モード

  実際の流体シミュレーションに対応し、収束条件が満たされた時点で反復から抜ける

コマンドライン引数 #3/3

その他のパラメータ

--dt=float　時間刻み幅を実数(CFL数単位で)で指定 (ディフォルト0.2)
--comm=str　通信方法をsync(同期)またはasync(非同期)で指定
　　(ディフォルト async)
--output_interval=int
　　出力ステップ間隔 (ディフォルト 0(出力なし))

- 出力データ

  圧力と速度の三次元データを、時間ステップ毎に、分散出力

ホットスポット

1. Poisson方程式の2色SOR計算
   – コア部分は、1時間ステップあたりN_{vp}*N_p*2回実行
   – 反復ごとに、袖通信と集約通信(残差値)あり
2. 圧力勾配計算と速度の更新
   – 1時間ステップあたりN_{vp}回実行
3. 対流項と粘性項の計算
   – 1時間ステップあたり1回実行

N_{vp}： V-P反復の反復数
N_p ：Poisson反復の反復数

NGS ANALYZER

プログラム構成

- 5つステージから構成
  1. インデックス作成
  2. 並列化のためのファイル分割
  3. アラインメント計算
  4. 重複削除
  5. 最終処理
- それぞれジョブファイルの自動作成、ジョブ投入、終了待ち、の一連の操作を行う（自前ワークフロー処理）

調査現状

- 未だ動作チェック完了せず
  - ステージ3の実行まで到達
  - 実行完了までは未達
- 不足情報
  - 複数種類の入力データおよびその実行見積もり時間
  - 実行結果検証手順
  - 計算内容の説明
  - 現在の実行規模、2018~2020年頃の想定実行規模の情報の提供
- 8月末までにミニアプリ化を終了し、東大チームに引渡し予定

その他調査中

- rmcsm
- pSpatiocyte
- GT5D
- para-TCCI
- OpenFMO
- CONQUEST
- STATE
- …

予定

- ミニアプリ化予定
  - 流体ミニアプリ
    - FFVC（数週間内完成予定）
    - もう1本程度
  - 気候ミニアプリ（第3四半期）
    - NICAM、MIROC（代替）
  - 量子化学（第3四半期）
    - NTCHEM
  - 物質科学系
    - ALPS/looper
    - CONQUESTとSTATEの調査開始済み

ミニアプリ利用事例

ミニアプリの目的

FSにおける
第一義的な目的

FSアーキテクチャ設計指針

- 東大チームにMODYLAS, CCS-QCD ミニアプリを提供済み、現在評価進行中
- 今後FFVC、NGS Analyzerを提供予定
- 東北大、筑波大チームにも情報提供済み

大局的な目的

アプリとシステムの共同作業の接点

アプリとシステムの共同作業の接点としてのミニアプリ

- アプリ側視点
  - 計算機システムに対する要求を具現化したもの
  - 要求
    - 性能
    - 使い勝手
    - 信頼性
- システム側視点
  - アーキテクチャ、システムソフトウェア、コンパイラ、ライブラリ等の実際的な評価
  - カーネルベンチマークを補完
    - 例１：　新プログラミング言語のアプリケーション全体の記述性評価
    - 例２：　耐故障チェックポイントのオーバーヘッド評価

事例:　気候シミュレーションコード SCALEを中心としたAICS内共同研究

気候シミュレーション
基盤ライブラリSCALE
の開発（富田チーム）

http://scale.aics.riken.jp/

京向け性能最適化（南チーム）

XcalableMPの評価（佐藤チーム）

通信機構の高度化（石川チーム）

MIC評価（石川チーム）

GPU評価（丸山チーム）

プログラム検証評価（前田チーム）

ミニアプリによる事例： ハイレベルフレームワークの評価

- （手前味噌ながら・・・）ステンシルフレーム Physisを構造格子流体ミニアプリ（FFVC）に適用
- Physis
  - C + ステンシル計算記述専用拡張
  - ソーストランスレータにより C/CUDA/MPI/MPI+CUDA等へ自動変換
  - 目標： 記述と性能のポータビリティ
    - 各種アーキテクチャ毎の個別最適化の自動化
  - 現状： GPUを主ターゲットとして開発中
  - http://github.com/naoyam/physis

FFVCのPhysisによる記述

- ひとまず・・・主要カーネルのポアソンソルバーステンシルをPhysisで記述
  - 全体の一部に過ぎないためデータコピーオーバーヘッド有り
  - 効率上はアプリケーション全体のデータモデルをフレームワーク上に移植するべき

オリジナルFortranループ

```
 do k=1,kx
   do j=1,jx
   do i=1+mod(k+j+color+ip,2), ix, 2
     idx = bp(i,j,k)
     ndag_e = real(ibits(idx, bc_ndag_E, 1))  ! e, non-diagonal
     ndag_w = real(ibits(idx, bc_ndag_W, 1))  ! w
     ndag_n = real(ibits(idx, bc_ndag_N, 1))  ! n
     ndag_s = real(ibits(idx, bc_ndag_S, 1))  ! s
     ndag_t = real(ibits(idx, bc_ndag_T, 1))  ! t
     ndag_b = real(ibits(idx, bc_ndag_B, 1))  ! b
     dd = 1.0 / real(ibits(idx, bc_diag, 3))  ! diagonal
     pp = p(i,j,k)
     ss = ndag_e * p(i+1,j  ,k  )  + ndag_w * p(i-1,j  ,k  ) &
       + ndag_n * p(i  ,j+1,k  ) + ndag_s * p(i  ,j-1,k  ) &
       + ndag_t * p(i  ,j  ,k+1) + ndag_b * p(i  ,j  ,k-1)
     dp = ( dd*ss + b(i,j,k) - pp ) * omg
     p(i,j,k) = pp + dp
     res = res + dble(dp*dp) * dble(ibits(idx, Active, 1))
   end do
   end do
end do
```

Physisステンシル関数

```
static void sor2sma_kernel(const int i, const int j, const int k,
                           PSGrid3DReal p, REAL_TYPE omg, PSGrid3DDouble res,
                           PSGrid3DReal b, PSGrid3DInt bp) {
  int idx = PSGridGet(bp, i, j, k);
  REAL_TYPE ndag_e = real(ibits(idx, BC_NDAG_E, 1)); //  ! e, non-diagonal
  REAL_TYPE ndag_w = real(ibits(idx, BC_NDAG_W, 1)); //  ! w
  REAL_TYPE ndag_n = real(ibits(idx, BC_NDAG_N, 1)); //  ! n
  REAL_TYPE ndag_s = real(ibits(idx, BC_NDAG_S, 1)); //  ! s
  REAL_TYPE ndag_t = real(ibits(idx, BC_NDAG_T, 1)); //  ! t
  REAL_TYPE ndag_b = real(ibits(idx, BC_NDAG_B, 1)); //  ! b
  REAL_TYPE dd = 1.0 / real(ibits(idx, BC_DIAG, 3)); //  ! diagonal
  REAL_TYPE pp = PSGridGet(p, i, j, k);
  REAL_TYPE ss = ndag_e * PSGridGet(p, i+1, j  ,k  ) + ndag_w * PSGridGet(p, i-1,j  ,k  )
     + ndag_n * PSGridGet(p, i  ,j+1,k  ) + ndag_s * PSGridGet(p, i  ,j-1,k  )
     + ndag_t * PSGridGet(p, i  ,j  ,k+1) + ndag_b * PSGridGet(p, i  ,j  ,k-1);
  REAL_TYPE dp = ( dd*ss + PSGridGet(b, i, j, k) - pp ) * omg;
  PSGridEmit(p, pp + dp);
  PSGridEmit(res, dble(dp*dp) * dble(ibits(idx, ACTIVE_BIT, 1)));
}
```

Physisステンシル制御コード

```
...
  PSGridCopyin(pg, p);
  PSDomain3D dom = PSDomain3DNew(g, sz[0]+g, g, sz[1]+g, g, sz[1]+g);
  if ((color + ip) % 2 != 0) {
    PSStencilRun(PSStencilMapRed(
        sor2sma_kernel, dom, pg, omg, resg, bg, bpg));
  } else {
    PSStencilRun(PSStencilMapBlack(
        sor2sma_kernel, dom, pg, omg, resg, bg, bpg));
  }
  PSGridCopyout(pg, p);
```

コード生成

```
> physisc-ref ffv_physis.C
```

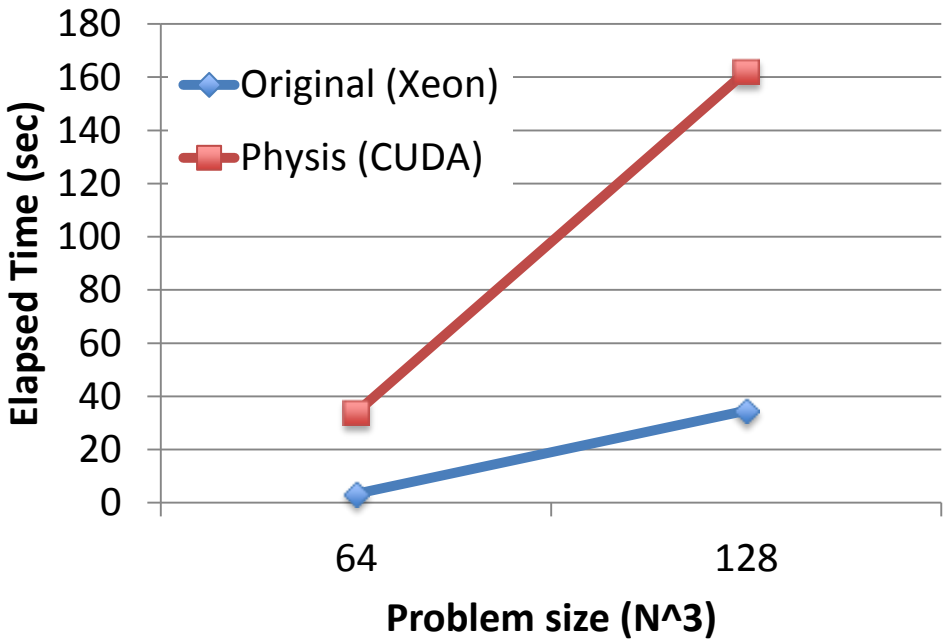
- Physis拡張構文を通常のC/C++コードに変換したものを生成
- 通常のC/C++コンパイラでコンパイル
- 主に動作確認用

```
> physisc-cuda ffv_physis.C
```

- NVIDIA GPU向けにCUDAコードを生成

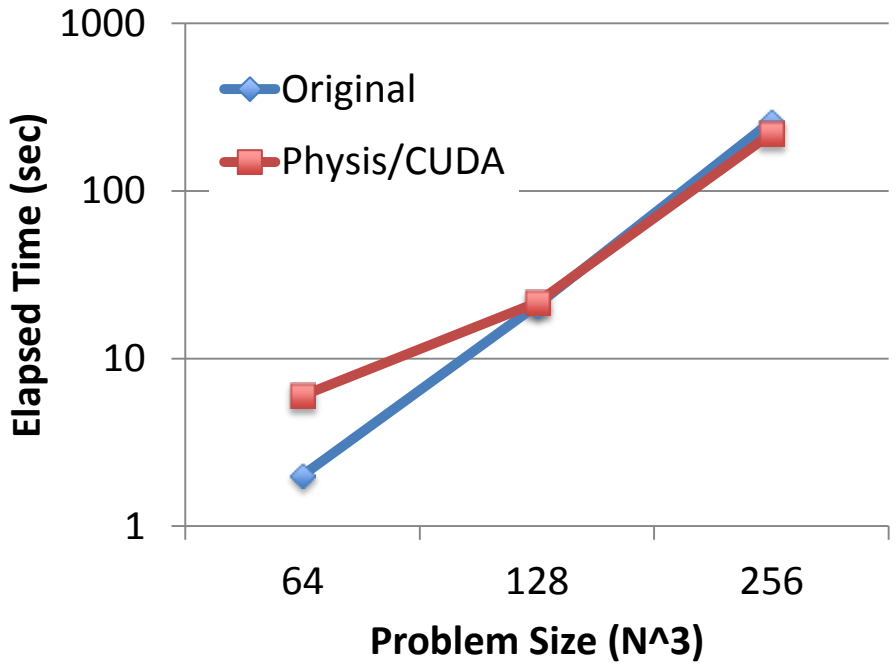
性能結果

- Tesla M2075 vs. Intel SNB Xeon E5-2670
  - M2075 BW: 90 GB/s ＞ Xeon: 35 GB/s
- メインループ全体実行時間
  - 圧倒的に負け
  - カーネル部分を
    オフロードしたのみ
    →PCI転送オーバ
    ーヘッド



性能結果

- Tesla M2075 vs. Intel SNB Xeon E5-2670
  - M2075 BW: 90 GB/s > Xeon: 35 GB/s
- ポアソンソルバーカーネルのみ
  - それでもなんとかトントン



知見

- ミニアプリへの適用は容易
  - アプリを理解してPhysisによるカーネルの実装に約半日
  - GPU依存コードの記載の必要なし
- 性能がでてない
  - PCI転送
    - アプリ全体のデータモデルをPhysis向けに変更すれば解決可能
  - カーネル性能
    - PhysisではCUDA向けにステンシル計算における典型的な最適化を実装済み
    - 単純なステンシルではチューニング済み人手実装と同等の性能
    - しかし、Red-black実行の場合の性能評価未実施
  - 今後コード生成の改良の必要あり

- 今後、元のアプリケーション開発元へのフィードバック
  - より高度なアプリケーション開発手法、指針を提示

まとめ

1. 提供アプリケーション本数 → 27本
2. アプリ調査＆ミニアプリ化現状
   - 新たに構造格子流体コードがほぼ完成
   - NGS Analyzerの調査進行中
   - 全体的なアプリケーション性能調査進行中
3. ミニアプリ利用事例
   - 東大チームにてミニアプリを用いた性能評価進行中
   - 計算機科学研究における利用事例