Wisteria/BDEC-01（Odyssey）における OpenMP によるプログラミング入門（その 3）

中島研吾[(a)]，笠井良浩[(b)]，坂口吉生[(b)]

(a)東京大学情報基盤センター，(b)富士通株式会社

本稿では，前々回，前回［4,5］に引き続き，Wisteria/BDEC-01（Odyssey）上でのプログラム最適化，性能評価について紹介する。プログラム類は Wisteria/BDEC-01 の **`/work/share/ompw/ompw.tar`** から取得できるので，興味ある方は試してみられると良い。また，プログラムのより詳細な説明については［2,3］を参照されたい。

1. 様々な問題規模における性能評価

前回［5］では，主として総メッシュ数＝128^3 の場合についての実行結果を紹介した。以下に問題規模を最大 256^3 とした場合の実行結果を示す。

表 1　問題規模を変えた場合の各プログラミング言語・各実装における計算時間（CG 法部分）

（a）総メッシュ数＝128^3=2,097,152

	Fortran	C (clang)	C (trad)
src0（初期設定）	1.671	1.564	2.354
src1（First Touch）	1.480	1.122	1.720
src2（+ELL）	0.747	0.809	1.127
src3（+omp-parallel削減）	0.707	0.834	0.854

（b）総メッシュ数＝160^3=4,096,000

	Fortran	C (clang)	C (trad)
src0（初期設定）	3.610	3.484	4.067
src1（First Touch）	2.993	2.228	3.425
src2（+ELL）	1.534	1.690	2.340
src3（+omp-parallel削減）	1.556	1.693	1.742

（c）総メッシュ数＝200^3=8,000,000

	Fortran	C (clang)	C (trad)
src0（初期設定）	7.666	8.321	9.397
src1（First Touch）	6.952	5.102	8.008
src2（+ELL）	3.421	3.910	5.381
src3（+omp-parallel削減）	3.440	3.920	3.824

（d）総メッシュ数＝256^3=16,777,216

	Fortran	C (clang)	C (trad)
src0（初期設定）	34.308	24.772	25.547
src1（First Touch）	32.202	22.172	23.814
src2（+ELL）	8.916	10.761	14.566
src3（+omp-parallel削減）	8.915	10.764	10.415

全体的な傾向としては，総メッシュ数＝128^3 の場合と変わらないが，以下の傾向が見られる。

- Fortran が最も効率が良く，以下 C（clang），C（trad）と続く
- src2⇒src3 による速度向上は C（trad）で顕著である他，Fortran，C（clang）ではほとんど効果が無い
- src3 では C（clang）と C（trad）はほぼ同じ性能であり，問題規模が大きくなると，C（trad）の方がむしろ高速となる
- Fortran は src0，src1 については，総メッシュ数＝256^3 の場合は，C（clang），C（trad）よりもむしろ遅いが，src2，src3 では 4 倍近い性能向上が見られ，C（clang），C（trad）より高速となる

2. 詳細プロファイラの適用

詳細プロファイラ［2,6］を適用することにより，計算性能，消費電力等詳細なデータを得ることができる。詳細プロファイラの使用手順は以下の通りである：

① プログラムの性能測定部分の最初と最後に fapp_start，fapp_stop を挿入する（図 1）。C 言語の場合は「#include “fj_tool/fapp.h”」を挿入する。複数箇所の測定も可能である［6］。コンパイルは通常の実行時と同じである。図 1 に示す例では，CG 法のループの前後に fapp_start, fapp_stop を挿入している。

② “-Hevent=pa1～pa17”として，計算を 17 回実行する（図 2）。”-d repo01～repo17”で指定される各ディレクトルに分析結果が出力される。各ディレクトリが既存の場合は中身を空にしておく必要がある。

③ ②で生成した各ディレクトリ内ファイルに対して処理を実施する（図 3）。実行ディレクトリに”pa1.csv”～”pa17.csv”の 17 個のファイルが生成される。

④ ③で生成した”pa1.csv”～”pa17.csv”を PC にコピーし，同じディレクトリに詳細プロファイラ用の Excel マクロファイル［6］を置き，ダブルクリックする。

⑤ 以下，指示に従うことによって解析結果が Excel ファイルとして出力される（図 4，表 2）。出力は各 CMG に対して行われる。

```
#include "fj_tool/fapp.h"

        fapp_start ("CG",1,0);
        Stime = omp_get_wtime();

for(L=0; L<(*ITR); L++) {
...
        if(ERR < EPS) {
           *IER = 0;
           goto N900;
        } else {
           RHO1 = RHO;
        }
}

        *IER = 1;
N900:
        Etime = omp_get_wtime();
        fapp_stop ("CG",1,0);

        return 0;
}
```

```
      call fapp_start ("CG",1,0)
      Stime = omp_get_wtime()

      do L= 1, ITR
...
        if (ERR .lt. EPS) then
          IER = 0
          goto 900
        else
          RHO1 = RHO
        endif

      enddo
      IER = 1
  900 continue
      Etime= omp_get_wtime()
      call fapp_stop ("CG",1,0)

      return
      end
```

図 1　詳細プロファイラ適用のための関数呼び出し［2,6］

fapp.sh

```
#!/bin/sh
#PJM -N "fapp"
#PJM -L rscgrp=lecture-o
#PJM -L node=1
#PJM -omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
export OMP_NUM_THREADS=48
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

fapp -C -d ./repo01 -Hevent=pa1 ./solf2
fapp -C -d ./repo02 -Hevent=pa2 ./solf2
fapp -C -d ./repo03 -Hevent=pa3 ./solf2
fapp -C -d ./repo04 -Hevent=pa4 ./solf2
fapp -C -d ./repo05 -Hevent=pa5 ./solf2
fapp -C -d ./repo06 -Hevent=pa6 ./solf2
fapp -C -d ./repo07 -Hevent=pa7 ./solf2
fapp -C -d ./repo08 -Hevent=pa8 ./solf2
fapp -C -d ./repo09 -Hevent=pa9 ./solf2
fapp -C -d ./repo10 -Hevent=pa10 ./solf2
fapp -C -d ./repo11 -Hevent=pa11 ./solf2
fapp -C -d ./repo12 -Hevent=pa12 ./solf2
fapp -C -d ./repo13 -Hevent=pa13 ./solf2
fapp -C -d ./repo14 -Hevent=pa14 ./solf2
fapp -C -d ./repo15 -Hevent=pa15 ./solf2
fapp -C -d ./repo16 -Hevent=pa16 ./solf2
fapp -C -d ./repo17 -Hevent=pa17 ./solf2
```

図 2 計算実行のためのシェルスクリプト例
（計算を 17 回実行する）

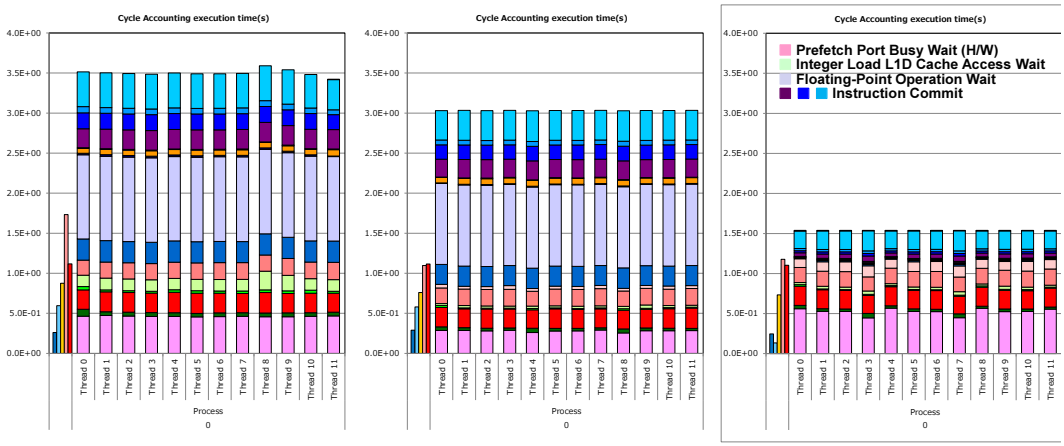
data.sh

```
#!/bin/sh
#PJM -N "data"
#PJM -L rscgrp=lecture-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o data.lst

module load fj
module load fjmpi

fapp -A -d./repo01 -Icpupa,mpi -tcsv -o pa1.csv
fapp -A -d./repo02 -Icpupa,mpi -tcsv -o pa2.csv
fapp -A -d./repo03 -Icpupa,mpi -tcsv -o pa3.csv
fapp -A -d./repo04 -Icpupa,mpi -tcsv -o pa4.csv
fapp -A -d./repo05 -Icpupa,mpi -tcsv -o pa5.csv
fapp -A -d./repo06 -Icpupa,mpi -tcsv -o pa6.csv
fapp -A -d./repo07 -Icpupa,mpi -tcsv -o pa7.csv
fapp -A -d./repo08 -Icpupa,mpi -tcsv -o pa8.csv
fapp -A -d./repo09 -Icpupa,mpi -tcsv -o pa9.csv
fapp -A -d./repo10 -Icpupa,mpi -tcsv -o pa10.csv
fapp -A -d./repo11 -Icpupa,mpi -tcsv -o pa11.csv
fapp -A -d./repo12 -Icpupa,mpi -tcsv -o pa12.csv
fapp -A -d./repo13 -Icpupa,mpi -tcsv -o pa13.csv
fapp -A -d./repo14 -Icpupa,mpi -tcsv -o pa14.csv
fapp -A -d./repo15 -Icpupa,mpi -tcsv -o pa15.csv
fapp -A -d./repo16 -Icpupa,mpi -tcsv -o pa16.csv
fapp -A -d./repo17 -Icpupa,mpi -tcsv -o pa17.csv
```

図 3　データ処理用シェルスクリプト例



(a) src0 (Initial)　(b) src1(+First Touch)　(c) src2（+ELL）

図 4　詳細プロファイラによる解析結果（0 番 CMG の 12 コアの計算時間内訳）
（総メッシュ数＝160^3=4,096,000），Fortran

表 2 は Fortran の場合の各実装における CG 法計算部分の詳細プロファイラによる分析結果の抜粋である。0 番 CMG の出力結果を 4 倍して 1 ノード，48 コアに換算している。SIMD 化率，メモリスループットともに，最適化によって向上していることがわかる。ELL を適用することにより，メモリ性能はピーク性能の 70%程度となっていることがわかる。それに反比例して命令数（Instruction）は減少している。消費電力（W）は「Node」はノードあたりの消費電力を示し，Core/L2/Memory はそれぞれ，計算コア（L1 キャッシュ含む），L2 キャッシュ，メモリの消費電力の内訳を示している。最適化により計算密度が上昇し，メモリもビジーとなるため，消費電力（W）の値は増加している。特にメモリ部分は src0⇒src2・src3 で 2.5 倍程度増加している。計算時間が半分以下となっているため，総消費エネルギ（J）では src0⇒src2・src3 で 64%程度となっている。

表 2　詳細プロファイラ出力結果抜粋（総メッシュ数＝160^3=4,096,000），1 ノード 48 コア換算

Fortran

	計算時間 (sec.)	対ピーク性能比 (%)	SIMD 化率 (%)	Memory Throughput (%)	Instruction		Power (W)	
					Effective	Load/ Store	Core L2 Memory	Node
solf0 (src0)	3.69	1.59	20.0	30.3	3.39×10^{11}	8.27×10^{10}	81.6 10.9 20.2	112.
solf1 (src1) (First Touch)	2.98	1.97	28.8	37.5	2.35×10^{11}	5.33×10^{10}	92.1 10.8 33.3	136.
solf2 (src2) (+ ELL)	1.58	3.73	49.7	70.0	1.19×10^{11}	4.17×10^{10}	104. 15.0 51.7	170.
solf3 (src3) (+ reduced “omp-parallel”)	1.58	3.72	48.4	69.8	1.22×10^{11}	4.10×10^{10}	101. 14.9 51.0	167.

3. まとめ

3 回にわたって，Wisteria/BDEC-01（Odyssey）を使用したプログラムの最適化，性能評価について紹介した。詳細プロファイラは，計算を 17 回実行しなければならないが，消費電力も含めた様々なデータを取得可能である。省電力化は今後のスーパーコンピュータの運用において重要な要素であり，読者の皆様も是非一回お手元のプログラムの消費電力を測定してみることをお勧めしたい。

参　考　文　献

[1] Wisteria/BDEC-01（「計算・データ・学習」融合スーパーコンピュータシステム）
https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria

[2] OpenMP によるマルチコア・メニィコア並列プログラミング入門（Wisteria/BDEC-01（Odyssey, A64FX 搭載），http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/

[3] P3D 関連資料
- ソースコード等： http://nkl.cc.u-tokyo.ac.jp/files/fvm.tar
- 解説資料（Fortran）：http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/omp-f-01.pdf
- 解説資料（C）： http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/omp-c-01.pdf

[4] 中島研吾，坂口吉生，笠井良浩，Wisteria/BDEC-01（Odyssey）における OpenMP によるプログラミング入門（その 1），スーパーコンピューテイングニュース（東京大学情報基盤センター）24-2，2022：https://www.cc.u-tokyo.ac.jp/public/VOL24/No2/11_202203Wisteria-2.pdf

[5] 中島研吾，坂口吉生，笠井良浩，Wisteria/BDEC-01（Odyssey）における OpenMP によるプログラミング入門（その 2），スーパーコンピューテイングニュース（東京大学情報基盤センター）24-3，2022：https://www.cc.u-tokyo.ac.jp/public/VOL24/No3/14_202205-Wisteria-3.pdf

[6] Wisteria/BDEC-01 利用支援ポータル：
https://wisteria-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal.ja/index.cgi