5A-01

マルチプラットフォーム対応連立一次方程式のソルバー開発と実行性能評価

川口優樹 † 宮島敬明 † 藤田直行 †

† 国立研究開発法人 宇宙航空開発機構

1 序論

近年, HPC 分野では FX100, Knights Landing (KNL), Tesla P100 など様々なアーキテクチャが登場している. HPC アプリ開発者はマルチプラットフォーム対応やハードウェアごとに最適化が必要となりつつある. しかしながら, 各アーキテクチャを考慮したアプリ開発はコーディング量が増加するなど敷居が高い.

工学的, 物理的現象を定式化すると偏微分方程式の初期値境界値問題が得られ, 差分法や有限要素法を用いて離散化すると大規模な連立一次方程式を解くことに帰着する. 多くの数値シミュレーションでは連立一次方程式の求解に時間を要するため, 連立一次方程式ソルバーを様々なアーキテクチャに実装し求解の高速化を図る研究が行われている [1]. しかしながら, HPC アプリ開発者が各アーキテクチャ向けに最適な実装をするのは容易ではないため, ポータブルに導入することができるマルチプラットフォーム対応された数値計算ライブラリが必要であると考えられる.

本研究の目的は, マルチプラットフォーム対応連立一次方程式ソルバーのライブラリを開発することである. 本ライブラリを Numerical Library for Multi-Platform (NLMP) と呼称する. また, 開発するライブラリの実行性能の評価を各アーキテクチャで行う.

2 連立一次方程式ソルバーのライブラリ

工学的, 物理的現象から得られる連立一次方程式の係数行列は零要素を多く含む疎行列になることが極めて多い. 一般的に疎行列を係数行列とする連立一次方程式に対しては Krylov 部分空間解法が用いられる. Krylov 部分空間解法には様々な解法があり, 共役勾配法が広く利用される. 本研究では Krylov 部分空間解法のマルチプラットフォームに対応したライブラリ化を行う.

Development and Performance Evaluation of Linear Equation Solver for Multi-Platform
Yuki KAWAGUCHI†,Takaaki MIYAJIMA†,Naoyuki FUJITA†
†Japan Aerospace Exploration Agency
kawaguchi.yuuki@jaxa.jp

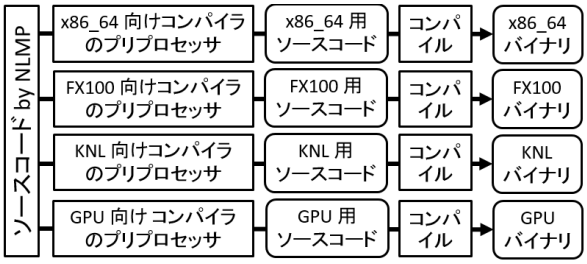

図 1: NLMP のコンパイルフロー図.

Krylov 部分空間解法の主要な演算は疎行列ベクトル積であり, 係数行列の非零要素のみを効率的にメモリに格納及び計算することで, 演算量とメモリ容量の削減が可能である. 疎行列の格納形式として Compressed Row Storage (CRS) 形式や Ellpack-Itpack (ELL) 形式がよく利用されるため本ライブラリでも対応する.

3 マルチプラットフォーム対応

NLMP の概念図を図 1 に示す. NLMP はコンパイル時に各コンパイラが定義している固有マクロを用いて, 利用するコードやライブラリを自動的に処理を切り替える. アプリ開発者は NLMP が提供する関数を記述し, 利用したいアーキ用のコンパイラでコンパイルするだけでマルチプラットフォーム対応が可能である.

Krylov 部分空間解法のアルゴリズムはベクトル同士の加算, 内積, 行列ベクトル積で構成されているものが多い. 上記の各演算をマルチプラットフォーム対応及び最適化することで, マルチプラットフォーム対応連立一次方程式ソルバーライブラリが開発可能となる [2]. 各コンパイラの固有マクロを用いて処理系を切り替えるベクトル同士の加算の例を Listing 1 に示す.

本研究で開発した NLMP の共役勾配法は約 2000 行のソースコードでできている. アプリ開発者がマルチプラットフォーム対応をする場合は同等以上のコーディングが必要となるが NLMP を利用することで利用したい機能を 1 行記述するだけで済む. すなわち, NLMP を利用することで開発コストの低減が可能である.

表 1: 計算機環境とコンパイラ環境.

SYSTEM	ARCHITECTURE
SORA-PPi	Intel Xeon E5-2643 × 2
SORA-MA	SPARC64 XIfx 2.2GHz (FX100)
KNL	Intel Xeon Phi 7250 Quadrant + Flat mode
Skylake	Intel Xeon Gold 6150 × 2
P100	Skylake + NVIDIA Tesla P100

Compiler	Compile Option
GCC 7.2.0	`-fopenmp -Ofast -march=native`
Intel 18.0.0	`-fopenmp -Ofast -march=native`
Fujitsu GM-2.0.0-06	`-fopenmp -Kfast -Xg -Kocl` `-Kprefetch_sequential=soft`
PGI 17.10 (CPU)	`-fast -acc -ta=multicore`
PGI 17.10 (GPU)	`-fast -acc -ta=tesla,cc60`
CUDA 9.0	`-O3 -arch=sm_60`

Listing 1: ベクトル同士の加算の例.

```
template<class T> void axpy_kernel(...){
#if defined (_OPENACC)
# pragma acc kernels present(ret, x, y)
#elif defined (_OPENMP)
# pragma omp parallel for
#endif
 for(i=0; i<n; ++i){
  ret[i] = alpha*x[i] + y[i];
 }
}

void axpy(...){ // NLMP ユーザーが利用するベクトル同士の加算
#if defined (__INTEL_COMPILER)||defined (__FUJITSU)
 cblas_daxpy(...);
#elif defined (__NVCC__)
 cublasDaxpy(...);
#else
 axpy_kernel(...);
#endif
}
```

4 数値実験

本研究では, 表 1 に示す各計算機環境において NLMP の実行性能の評価を行う. 評価には HPCG から得られる連立一次方程式に対し NLMP より共役勾配法を用いて 300 回反復計算させる際の実行性能の比較を行う. 評価に利用する連立一次方程式の次元数は 8,000,000 元であり, 係数行列の非零要素数は 213,847,192 個の疎行列とする. また, 疎行列の格納形式は CRS 形式と ELL 形式を用いる. 得られた結果を表 2 に示す.

表 2 より, まず NLMP を用いて作成したソースコードはコンパイラを変えるだけで各アーキテクチャで実行可能なプログラムを生成できていることがわかる. また, CRS 形式の実行性能においては CUDA の P100 を用いるよりも Intel コンパイラの KNL を利用することで 1.3 倍高速に計算できることがわかる. 一方で ELL 形式においては Intel コンパイラの KNL よりも CUDA の P100 を用いることで 3 倍高速に計算できることがわかる. CPU は CRS 形式のほうが高速なのに対して GPU は ELL 形式のほうが高速な理由については, ELL 形式にすることによりパディングデータ分のデータ転送が増えたことでメモリバンド幅ネックになっていることが原因であると考えている.

表 2: CRS 形式と ELL 形式の共役勾配法の実行時間. ただし, 反復回数は 300 回とし, データの単位は秒とする. また, KNL は MCDRAM を利用している. なお, - は利用可能なコンパイラがないことを示す.

CRS	GCC	Intel	Fujitsu	PGI	CUDA
SORA-PPi	21.5	23.6	-	23.5	-
SORA-MA	31.6	-	9.7	-	-
KNL	3.9	3.4	-	-	-
Skylake	7.6	7.8	-	7.4	-
P100	-	-	-	8.5	4.7

ELL	GCC	Intel	Fujitsu	PGI	CUDA
SORA-PPi	35.6	40.1	-	56.7	-
SORA-MA	34.3	-	12.1	-	-
KNL	7.0	6.5	-	-	-
Skylake	19.2	22.3	-	14.4	-
P100	-	-	-	2.1	2.1

5 結論と今後の展望

本研究では, マルチプラットフォーム対応数値計算ライブラリ NLMP を開発し実行性能の評価を行った. 結果, 約 2000 行のソースコードで共役勾配法のマルチプラットフォーム対応をすることができ, CRS 形式と ELL 形式で実行性能の評価を完了した.

上記の結果から, HPC アプリ開発者に NLMP を利用してもらうことで効率的にアプリのマルチプラットフォーム対応を促進してもらえると考えられる. 今後の展望として MPI を NLMP に導入し, 複数ノードでの計算を可能にすることを検討中である.

謝辞

数値計算の実行に当たっては宇宙航空研究開発機構スーパーコンピュータ JSS2 を用いた [4].

参考文献

[1] 中島研吾, 大島聡史, 塙敏博, 星野哲也, 伊田明弘: ICCG 法ソルバーの Intel Xeon Phi 向け最適化, 情報処理学会研究報告 Vol.2016-HPC-157 No.16.

[2] 川口優樹, 宮島敬明, 藤田直行: OpenACC による共役勾配法カーネルコードの並列化と実行性能評価, 第 59 回プログラミング・シンポジウム, 2017/1/21.

[3] HPCG. http://www.hpcg-benchmark.org

[4] JAXA Supercomputer System Generation 2. https://www.jss.jaxa.jp