1J-05

OpenACCを用いた単精度LU分解のGPU並列化

高山 智礼　久木田 仁　矢島 雄河　藤井 昭宏　田中 輝雄
工学院大学

1. はじめに

近年，科学技術計算は倍精度演算が主流である．しかし，現在盛んに行われている機械学習では低精度演算が注目されている．スーパーコンピュータのベンチマークHPL-MxP [1] は，対角優位な行列に対する単精度LU分解と倍精度反復改良からなる混合精度演算を行っている．計算の主要部分は単精度のLU分解である．

一方，GPUはもともと画像処理向けで単精度演算が強力である．しかし，GPUで演算させるにはGPU専用にプログラムの書き換えや関数の追加を必要とる．また，新たにCPUとGPU間のデータ転送のプログラムを追加で記述する必要がある．一方で，CPUプログラムに指示句を追加するだけで，GPUで動作するコードを生成するディレクティブベースのGPU向けプログラミングOpenACC [2] の利用が広がっている．

本研究では，HPL-MxPの中で多くの計算時間の割合を占める単精度LU分解についてOpenACCを用いてGPUを実行し，その性能を評価する．

2. 対象とするプログラムHPL-MxP

HPL-MxPは，倍精度の対角優位な行列をpivotingなしの単精度LU分解と倍精度GMRESを利用した反復改良による混合精度アルゴリズムで連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く．HPL-MxPはもともとHPL-AIと呼ばれていたが，AIに限らない混合精度演算の指標として扱えるため，名称が変更された．

本研究では多くの計算時間の割合を占める単精度LU分解を対象とした．HPL-MxPのLU分解は再帰的なアルゴリズムになっている．その中で基本的な線形代数ルーチンが用いられており，ifやreturnが途中で含まれる複雑なプログラムになっていた．これを評価のため，プログラムの構造を維持して単純化した．またこのプログラムはCPUで最適化されていたため，最内側がメモリの連続方向になっていた．

GPU parallelization of single precision LU decomposition using OpenACC
Chihiro Takayama　Jin Kukita　Yuga Yajima
Akihiro Fujii　Teruo Tanaka
Kogakuin University

```
1 #pragma acc kernels
2 #pragma acc loop independent
3 for (int j = kiten+nb; j < kiten+nb+nj; j++)
4 #pragma acc loop independent
5   for (int i = kiten; i < kiten+ni; i++)
6     for (int k = kiten; k < i;  k++)
7       A(i,j) = A(i,j) - A(i,k) * A(k,j);
```

図1: OpenACCを用いたLU分解のプログラムの一部

3. OpenACC

OpenACCはディレクティブベースのプログラミング手法を採用した，GPUだけに限らずアクセラレータ・デバイス上の並列プログラミングを行うための標準規格である [3]．HPL-MxPのLU分解部分にOpenACCの指示句を記述したプログラムの一部を図1に示す．OpenACCの主な指示句として，GPUで並列化する部分を指示する#pragma acc kernelや，CPUとGPU間のデータ転送を行うcopyやupdate等がある．OpenACCの並列化の対象はfor文などの構造化ブロックに限られる [3]．データ転送を行う際はGPU上のメモリ領域の確保を意識する必要がある．

しかし，OpenACCの指示句をOpenMP [4] と同じ場所に記述するだけでは性能がでない．そこで以下の条件を満たすようにチューニングを行った．

- 可読性が低くならないよう，プログラムの構造を根本的に書き換えない
- 同一プログラムがGPUの有無によらず動く
- 将来的にコンパイラが高機能化すれば，コンパイラで対応可能と考えられる

以上の3条件をもとに基本的な高速化手法であるループ交換，アンローリング，ブロッキングを用いた．まずGPUのメモリアクセス最適化手法の一つにコアレスアクセスの活用がある．コアレスアクセスは並列スレッドがメモリ上の連続領域へ同時にアクセスすることであり，ループ交換によりこれを活用できる．次に，GPUはコア数が多いため，大量の並列化が必要となる．これは，複数の多重ループをそれぞれ並列化指示することで対応可能である．また，GPUは多量のスレッドが同時に動くため，内部メモリとキャッシュに収まるように行列サイズを抑える必要がある．ブロッキングを行うことで，LU分解中の行列を適切な大きさに分割して演算できる．

Copyright ©2024 Information Processing Society of Japan.
All Rights Reserved.

表 1: 実験環境: 不老 Type II サブシステム

	CPU	GPU
モデル コア数 メモリ キャッシュサイズ 理論演算性能	Intel Xeon Gold 6230 20 コア× 2 (DDR4 2933MHz) 384GiB 27.5MB 1.344TFLOPS × 2 ソケット	NVIDIA Tesla V100 (Volta) 2560 FP64 コア (HBM2) 32GiB × 4 ソケット 6144 KB 7.8TFLOPS × 4 ソケット
コンパイラ コンパイルオプション	gcc 11.3.0 -fopenmp -O3 -std=c99	nvc 12.0.76 -mp -O3 -Minfo=accel -acc -ta=tesla:cc70 -tp=skylake

表 2: 実験パターン

パターン 1	CPU での逐次実行
パターン 2	OpenMP を用いた CPU の並列実行
パターン 3	OpenACC を用いた GPU の並列実行
パターン 4	パターン 3 の jki ループを jik に変更 (i がメモリアクセス連続方向)
パターン 5	パターン 4 に 8 段のアンローリングを追加
パターン 6	パターン 5 に 1024 のブロッキングを追加

4. 実験

4.1 実験概要

前章に記述した 3 つの手法を段階的に適用し，行列サイズを 64 から 8192 まで 2 のべき乗で変化させた．計測部分は LU 分解の関数のみである．実験には名古屋大学のスーパーコンピュータ不老 Type II サブシステムの 1 ノードを用いた．実験環境を表 1 に，実験パターンを表 2 に示す．

4.2 結果と分析

実行結果を図 2 に示す．横軸は行列の一辺の長さ，縦軸は性能である．

アンローリングにより (パターン 4 と 5 の比較) 性能が約 2 倍になった．これはスレッドあたりの演算量の増加が要因である．次にブロッキングの効果を示す (パターン 5 と 6 の比較)．V100 が持つ 6144KB の L2 Cache に収まる行列サイズを計算する．計算式は $\sqrt{6144[\mathrm{KB}] \times 1000/4} = 1239.3$ となる．この行列サイズをもとに 1024 でブロッキングを行う．LU 分解中の行列が L2 Cache に収まるようになり，問題の行列が大きい場合でも性能の低下を防ぐことができた．

以下では行列サイズを 4096 として考察する．NVIDIA が開発した GPU 向けライブラリ cuSOLVER の LU 分解関数で計測をした結果，行列サイズ 4096 で 2.1TFLOPS であった．パターン 6 の結果より，ハードウェアに特化した cuSOLVER 性能の 11% までだすことができた．また，GPU の特性に合ったチューニングを行ったパターン 6 は最終的に CPU での逐次実行 (パターン 1) の 14 倍程度となった．

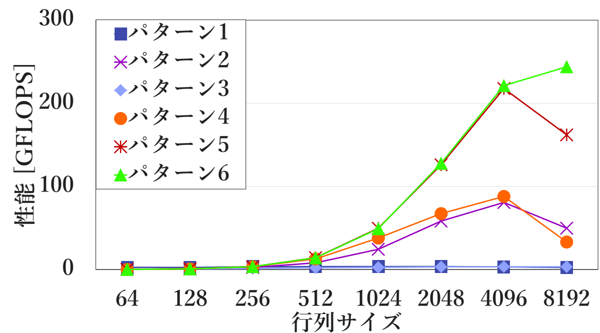

図 2: 不老 Type II サブシステムの単精度 LU 分解性能

5. おわりに

本研究では，OpenACC を用いて HPL-MxP の単精度 LU 分解を GPU 上で並列化した後，ループ交換，アンローリング，ブロッキングを適用し評価した．ディレクティブベースを用いたことで，GPU 専用に大幅な書き換えを行う必要がなかった．このプログラムは，ディレクティブを無視することで CPU でも実行することができる．数行の記述とコンパイルオプションの変更のみで実行することができ，管理，計測を行うことが容易となる．

今後の課題として，今回の実験の中でパターン 5 と 6 は固定値で行った．これらの値を変数としてチューニングを行うことで，さらなる性能向上が期待できることがあげられる．

謝辞

本研究の一部は JHPCN jh230045 及び JSPS 科研費 JP18K11340，JP23K11126 の助成を受けた．

参考文献

[1] HPL-MxP，*https://hpl-mxp.org* (2023-12-21).

[2] Application Programming Interface, Open ACC，*https://www.openacc.org* (2023-12-21).

[3] アクセラレータデバイス OpenACC Programming ディレクティブによるプログラミング，HPC world，*https://hpcworld.jp/archive/SPG/Pgi/OpenACC/index.html* (2023-12-21).

[4] OpenMP，*https://www.openmp.org* (2023-12-21).

Copyright ©2024 Information Processing Society of Japan.
All Rights Reserved.