B-040

GPU のための回路方程式求解における命令レベル並列度の評価
Evaluation of Instruction Level Parallelism in Circuit Equations for GPU.

富永　浩文 †　中村　あすか †　篠塚　研太 †　前川　仁孝 †
Hirobumi Tominaga　Asuka Nakamura　Kenta Shinozuka　Yoshitaka Maekawa

1. はじめに

回路方程式の求解は，ランダムスパースな係数を持つ連立一次方程式を解く必要があり，従来より直接法による求解が行われている [1]．直接法による求解には，$O(n^3)$ の計算量が必要となることから，演算を高速化するために並列化の研究が行われている．近年，並列計算機として注目されているアーキテクチャに GPU がある．GPU を，効率良く用いるためには，並列化可能な個所を多くしてベクトル化率を高くする必要がある．しかし，回路方程式は高いスパース性を持つため，高いベクトル化を得ることが難しい．本問題を解決するために，命令レベルの並列性を利用し高速化する手法の一つとして，拡張ベクトル化 LU 分解法が提案されている [2]．本手法は直前の命令の実行に必要なデータの依存関係を解析し，同時に実行可能な命令をベクトル化することで高い並列性が得られる．しかし，GPU は，限られたメモリリソースで多くのスレッドを実行するため，拡張ベクトル化 LU 分解法によってベクトル化された命令が一度で処理ができないことがある．このような場合，複数のベクトル命令が発行されメモリアクセス回数が増え，メモリアクセスがボトルネックとなり処理効率が低下する．

そこで，本稿では，GPU を用いた回路方程式求解に対して，スタティックスケジューリングを用いた近細粒度並列処理を行う手法を提案し，従来ベクトルプロセッサにおいて有効とされていた拡張ベクトル化 LU 分解法に対して提案手法の有効性を評価する．近細粒度並列処理は，タスクグラフにより命令間の依存関係を抽出し，プライオリティリストを用いて並列化を行うことで，演算リソースに応じて効率的な回路方程式求解を行うことができるため，GPU を用いた回路方程式求解において高い高速化が期待できる．

2. 回路方程式求解の並列化

本章では，従来回路方程式求解に有効とされている命令レベルの並列性を利用した拡張ベクトル化 LU 分解法とスタティックスケジューリングによる近細粒度並列処理について述べる．

2.1 拡張ベクトル化 LU 分解法

拡張ベクトル化 LU 分解法は，LU 分解法で用いる除算演算と積差演算における要素の参照関係に基づき，演算の実行可能な順序を表すレベル付けを行う．LU 分解法において計算に必要となる要素は，除算演算では更新要素と参照要素，積差演算では更新要素と 2 つの参照要素を用いて計算する．参照要素は，次に更新されるまでは計算に用いても解が変わることがない．このため，計算に必要な参照要素が，更新要素として更新されるまでは，その要素のみを用いた計算は並列に計算することができる．拡張ベクトル化 LU 分解法によるベクトル化手法の手順は以下の通りである．まず，LU 分解をシミュレートするシンボリック分解を行う．このとき，各要素で実行可能な順番を表すレベルを各演算毎に設定し，最も最大のレベルが設定されている要素のレベルを増やし，そのレベルを演算レベルとする．次に，各命令で設定されたレベルの低い順から，同一のレベルとして設定されている命令をベクトル化し実行する．

2.2 スタティックスケジューリングによる近細粒度並列処理

近細粒度並列処理は，ステートメントレベルの並列性を抽出し並列化することで高い並列性を得ることができる [3]．本手法は，実行に必要な命令全ての依存関係からタスクグラフと呼ばれる無サイクル有向グラフを生成することで，全ての命令の依存関係を明確にする．生成したスクグラフより，各タスクの実行順序を決定する CP/MISF 法などの効率的なスタティックスケジューリングを用いてプライオリティリストを作成する．生成したプライオリティリストを用いて実行可能なタスクを割り当てることで，最適なタスクの実行順序を決定する [4]．

3. CUDA による命令レベル並列性を用いた回路方程式求解手法

NVIDIA 製 GPU は，数百のプロセッサ (CUDA Core:CC) を持ち，高いメモリバンド幅と高速なメモリを持つことで高速にデータを処理する．しかし，GPU は一度に多くのデータへアクセスするため，メモリアクセスに時間がかかる．このため，各 CC は，Streaming Multiprocessor(SP) と呼ばれる単位でまとめられ，各 SP の内の CC は，SP 内の CC で共有可能な高速なシェアードメモリを用いて，メモリアクセスを減らす．しかし，データを処理する際に演算レジスタなど計算リソースを上回るデータを処理する場合，レジスタに比べ低速なメモリへの書き戻しが行われる．このため，メモリアクセスの削減をリソース量に合わせたデータの割り当てをシェアードメモリを用いて，メモリアクセスの最適化を行うことが重要である．

このような GPU を汎用目的に用いるための API として Compute Unifide Device Architecture(CUDA) がある．CUDA は，効率的に命令の実行を行うために，Single Program Multiple Data(SPMD) のプログラミングモデルを採用し，多くのデータが，単一のプログラムによって処理される．データの処理を行うときに，CUDA は Warp と呼ばれる単位で SIMD 演算のように処理が行われる．このため，同一の Warp 内で一度に処理するデータに異なる処理を必要とする命令が実行されると，分岐処理が発生し，処理効率が低下する．処理効率の低下を防ぐため，同一の命令を実行するようにベクトル化する．

†千葉工業大学情報工学科, Department of Computer Science, Chiba Institute of Technology

表 1: 評価に用いた問題の詳細と生成されたベクトル命令数

Problem No. : Name	Size	Non-Zero	演算命令数	拡張 最大ベクトル命令数 (制限 有:無)	提案 最大ベクトル命令数 (制限 有:無)
1: Hamm add32	4,960x4,960	19,848	58,060	171 : 170	140 : 140
2: Rajata rajat12	1,879x1,879	12,817	31,076	1,807 : 1807	1,767 : 1,767
3: Bomhof Circuit2	4,510x4,510	21,199	2,613,021	2,495 : 2,266	2,219 : 2,219
4: Bomhof Circuit3	12,127x12,127	48,137	2,307,224	3,883 : 3,740	3,425 : 3,395
5: Hamm memplus	17,758x17,758	99,147	1,238,173	1,184 : 1,111	1,052 : 1,052

拡張ベクトル化 LU 分解法を用いた同一演算命令同士の命令のベクトル化は，抽出されたレベル毎に同一演算同士で行う．同一レベルで異なる命令を抽出した場合，異なる命令のベクトル命令を生成する．また，演算リソースを超えた命令を抽出した場合，超えた命令を後続のベクトル命令として生成する．生成したベクトル命令を GPU を用いて処理する場合，ベクトル命令を CPU から GPU 上へ転送，またはオンチップメモリからロードする．このため，ベクトル命令数を低く抑えることでメモリアクセスが減り，演算を高速化できる．そこで，本提案では，リストスケジューリング時に一番プライオリティが高く初めに実行が可能となっているタスクの命令と同一命令のうち，実行可能になっているタスクを演算リソースで計算できる分だけ並列化可能タスクとしてベクトル化することで，GPU のアーキテクチャを考慮した効率的なベクトル命令の実行を行う．

4. 並列化率の評価

提案手法によるベクトル化の有効性を示すために，実問題の例として The University of Florida Sparse Matrix Collection[5] を用いて評価する．評価を行う上で，GPU の計算リソースを考慮し同時実行可能な命令数を決定する必要がある．同時実行可能な命令数として一度に抽出する命令数は，CUDA の各ブロックが使用するレジスタ数とシェアードメモリの容量に依存する．本研究では，GTX480 を用いて全ての SP で実行可能な命令数を計測した結果，約 7500 であった．このため，同時実行命令数を 7500 とした．

表 1 に，評価に用いた問題と，拡張ベクトル化 LU 分解法と近細粒度並列処理によって抽出された最大の並列度を示す．ここで，演算命令数は，求解に必要な演算数を示し，最大発行ベクトル命令数はそれぞれの手法で並列化した際に，求解に必要なベクトル命令数を示す．最大ベクトル命令数は，ベクトル命令数の発行回数が高ければ高いほど多くなるため，低い数値であればあるほど高速に求解できる．また，並列度の制限は，GPU のリソースを考慮して抽出を行った場合を制限有とし，考慮せずに同時に実行可能な命令を最大限抽出した際のベクトル命令数を制限無とする．

評価の結果，近細粒度並列処理により生成したベクトル数は，拡張ベクトル化 LU 分解法に対して，最大で問題 1 において約 1.22 倍，最小でも約 1.02 倍以上のベクトル数が減少することが確認できた．また，ベクトル化されたベクトル命令数は，求解に必要な演算数が多いほど，拡張ベクトル化 LU 分解法よりもベクトル数の減少が確認できた．これは，演算数が多いほど並列可能な箇所が多いので拡張ベクトル化 LU 分解法では，リソース量を上回る並列度を抽出した回数が増え，ベクトル命令数が増加したと考えられる．逆に，近細粒度並列処理では，スタティックスケジューリングの特徴を利用したため，ベクトル命令数を抑えられたと考えられる．

次に，リソースの制限を考慮した結果，拡張ベクトル化 LU 分解法では，リソースを考慮していない場合と比べ，問題 2 以外で増加がみられ，最大で問題 3 において約 9%のベクトル命令数の増加が確認できた．近細粒度並列処理では，リソースを考慮していない場合と比べ，問題 4 において約 0.8%ベクトル命令数が増加したが，他の問題では増加が見られなかった．これは，拡張ベクトル化 LU 分解法では，リソースを超えたベクトル命令は，リソースに合わせてベクトル命令を単純にベクトル化したため，ベクトル命令数が増えたと考えられる．拡張ベクトル化 LU 分解法に比べ，本手法がリソースの制限を加えた際に，ベクトル命令数の増加を抑えられたのは，従来の CP/MISF 法に条件を加えることで，CP/MISF 法の利点を生かしたまま同一演算のスケジューリングを行い，少ないベクトル命令で高い並列化が可能になったと考えられる．

5. おわりに

本稿では，GPU による回路方程式求解を高速化するために，同一演算命令のベクトル化を行う近細粒度並列処理の手法を提案し，拡張ベクトル化 LU 分解法と近細粒度並列処理における並列度について評価した．評価の結果，近細粒度並列処理の方がベクトル命令発行回数が最大で約 1.22 倍の減少が確認できた．

参考文献

[1] 前川 仁孝，高井 峰生，伊藤 泰樹，西川 健，笠原 博徳：スタティックスケジューリングを用いた電子回路シミュレーションの階層型並列処理手法：情報処理学会論文誌．Vol.37，No.10，pp1859-1868，1996.

[2] 鹿毛 哲郎，下郡 慎太郎：ベクトル計算機による高速回路シミュレーションのためのベクトル化処理手法：電子情報通信学会論文誌 D，Vol.J70-D，No.8，pp1597-1606，1987.

[3] 笠原 博徳：マルチプロセッサシステム上での近細粒度並列処理：情報処理学会誌，Vol.37，No.7，pp651-661，1996.

[4] H. Kasahara，S. Narita：Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing：IEEE Trans. Comput.，Vol.C-33，No.11，pp1023-1029，1984.

[5] The University of Florida Sparse Matrix Collection：http://www.cise.ufl.edu/research/sparse/matrices/