2B-4

GPU を用いたブール行列の積および関係の推移的閉包の計算

大塚 寛†

愛媛大学 大学院理工学研究科(理学系) 数理物質科学専攻†

はじめに

線形時相論理(Linear Temporal Logic)に基づくモデル検査システム[1]では、システムが仕様を満たすことを Buchi オートマトンの空言語判定問題に帰着して示す。空言語判定問題は状態空間に対する 2 重の深さ優先探索(nested DFS)を用いて解かれる。この問題は有向グラフにおける推移的閉包を用いても解くことができ、またグラフの操作はその隣接行列に対する演算で行うことができる。そこで近年注目されている GPU (Graphics Processing Unit) を汎用目的の計算に使用する GPGPU (Geneal Purpose computing on GPU)、具体的には nVIDIA 社が提供する CUDA[2]を利用して、隣接行列のべき乗により推移的閉包を求めることで空言語判定問題を解く実験を行ったので、その結果を報告する。

背景

モデル検査はソフトウェアの正しさを示すための形式手法の一種である。このうち LTL モデル検査では、システムの振る舞いを Buchi オートマトン M で、システムの満たすべき仕様を LTL 式 ϕ で表し、システムが仕様を満たすこと($M \vDash \phi$)を以下の等価性により、Buchi オートマトンの空言語判定問題に帰着する。

$$\begin{aligned} M \vDash \phi &\Leftrightarrow L(M) \subseteq L(B_\phi) \\ &\Leftrightarrow L(M) \cap \overline{L(B_\phi)} = \varnothing \\ &\Leftrightarrow L(M \times B_{\neg\phi}) = \varnothing \end{aligned}$$

Buchi オートマトン $M = (S, S_0, L, T, F)$ によって無限長の語 $l_0 l_1 l_2 \cdots \in L^\omega$ が受理されるとは、$l_0 l_1 l_2 \cdots$ による実行 $s_0 s_1 s_2 \cdots$ において、

$$\exists f \in F \; s.t. \forall i \geq 0, \exists j \geq i \; s.t. \; s_j = f$$

ただし $s_0 \in S_0, (s_i, l_i, s_{i+1}) \in T \; (i \geq 0)$。すなわち、ある受理状態 $f \in F$ が無限回現れることである。

LTL モデル検査システム(例えば著名なシステムである SPIN など)では、この受理条件に基づく空言語判定問題を以下の方針で解く。

1. 初期状態 $s_0 \in S_0$ から F の要素まで深さ優先探索でたどる。このときその状態までに訪れた状態を保存しておく。
2. その状態から深さ優先探索を行い、1.で保存されている状態に到達したら受理されることになるので、”not empty”を出力して終了する。到達できなければ 1.に戻り、別の候補となる受理状態を探す。
3. 1.の深さ優先探索が途中で終了することなしに完了したら、受理される語がないので、”empty”を出力して終了する。

すなわち Buchi オートマトンの状態遷移のなす状態空間を nested DFS で探索し、必要であれば 2.の”not empty”を出力する際に反例として受理される語を出力する。

ところで nested DFS は計算量的には最適だが、並列化が困難であることが知られている。そこでこの問題を、状態空間を有向グラフとみなしたときの推移的閉包と捉え、これを隣接行列を用いて解くことを考えた。

A を状態集合 S の遷移集合 T による隣接関係による有向グラフを表す隣接行列

$$A = (a_{i,j}), a_{i,j} = \begin{cases} true & \exists l \; s.t. (s_i, l, s_j) \in T \\ false & otherwise \end{cases}$$

とすると、状態数 n に対して $A(I+A)^{n-1}$ は A の推移的閉包となる。ただし I は対角成分が *true* その他の成分が *false* である単位行列であり、行列の和とは成分ごとの論理和、行列の積とは成分ごとには論理値ベクトルの成分ごとの論理積の論理和である。

CUDA の特徴

CUDA について詳しくは述べないが、SIMD 型または SIMT 型(Single instruction multiple thread)の処理に適しており、一度に数千から数万のスレッドを実行できる一方、実装上注意すべき点がある。

- スレッド内の条件分岐で同じ制御フロー経路

Boolean Matrix Multiplication and Transitive Closure of Relations using GPU

†Hiroshi Ohtsuka

Graduate School of Science and Engineering, Ehime University

に従わないと Warp Divergence を生じる。
- 同期が取れるのは同一ブロック内のスレッドに限る。

実装

基本的には行列のべき乗を求め、得られた結果から初期状態から受理状態への実行および受理状態からそれ自身への実行があるか(対応する行列の成分が true か)を調べるだけである。なお行列の積は論理演算を用い、論理値を int 型として扱う。以下に挙げた疑似コードはべき乗 $A(A+I)^{n-1}$ を求め、それを R に返す部分である。**while** による繰り返しの回数は高々 $\lceil \log(n-1) \rceil$ である。

$R \leftarrow A, \quad P \leftarrow A+I, \quad i = n-1$
while $(i \neq 0)$ {
  if $(i \,\&\, 1 == 1)$ {
    $tmp \leftarrow R \times P$ // 行列の積
    $R \leftarrow tmp$ // 行列のコピー
  }
  $tmp \leftarrow P \times P$ // 同上
  $P \leftarrow tmp$
  $i >>= 1$
}

GPU による実装では、上に挙げた注意点から、行列の積と行列のコピーのみを別々の kernel 関数として実装し、上のコード自体は CPU 側で実装、積とコピーは同期をとりながら続けて呼び出す方針をとった。なお行列 *R, P, tmp* は GPU 側のメモリ上に確保する。

実行環境

実験に用いた環境を以下に示す。CPU 側は表 1 のとおりである。CPU 側でも並列化は可能で、比較のため Intel C コンパイラの CilkPlus の機能を用いて並列化(Cilk_for のみ利用, ワーカ数 8)を行った結果を挙げる。

表 1

CPU	Core i7 860 (2.80GHz)
Memory	8Gbyte
OS	Windows7 SP1 64bit
CCompiler	Intel CC 12.0

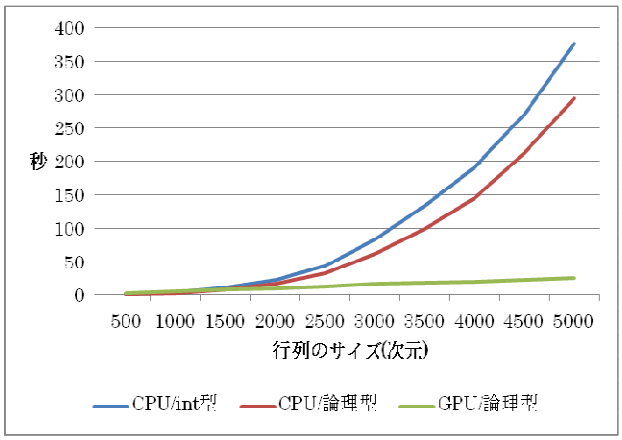
GPU 側は表 2 のとおりである。

表 2

GPU	GeForce GTX 285 (1.48GHz)
Memory	998Mbytes
Capability	1.3
SDK	CUDA 4.0

実験結果

ここでは、上のアルゴリズムを CPU のみおよび GPU 併用で実装した結果を挙げる。併せて通常の int 型の行列の演算の結果も挙げる。



行列の次元が 1500 以上となって初めて GPU が CPU を上回る。ちなみに疑似コード中の行列の積の回数は 15～19 回で、行列のコピーと併せてその倍の同期処理が行われる。

考察

数値型の行列演算での GPU の対 CPU 比での高速性は言及されていたが、論理型とその演算でも同様の高速化が得られた。一方で、論理演算の性質を利用した積和計算内でのループの脱出、推移閉包作成前での受理状態への到達確認など、CPU では当たり前に出来る高速化の処理が GPU には向かない、あるいはオーバーヘッドを伴うことも確認できた。

まとめと課題

モデル検査に現れる Buchi オートマトンは隣接行列としては疎行列であり、疎行列の GPU 上でのデータ構造としては CSR[3]が知られているが、今回は CSR による実装は採用しなかった。また nested DFS との系統的比較は出来なかったが、今回の方法は空言語判定問題を幅優先探索(BFS)を用いて解くことに対応しているとも言える。したがって BFS との比較も必要である。

参考文献

[1] G.-J. Holzmann, the Spin Model Checker
[2] NVIDIA CUDA C Programming Guide
http://developer.nvidia.com/page/home.html
[3] J. Barnat, M. Ceska, and T. Lamr, CUDA accelerated LTL Model Checking