数理解析研究所講究録
第1815巻 2012年 163-167

有理区間数とGPU並列処理による陰関数描画について

近藤 祐史*
YUJI KONDOH
香川高等専門学校
KAGAWA NATIONAL COLLEGE OF TECHNOLOGY

兵頭 礼子
NORIKO HYODO
アルファオメガ
ALPHAOMEGA INC.

村尾 裕一
HIROKAZU MURAO
電気通信大学
UNIVERSITY OF ELECTRO-COMMUNICATIONS

齋藤 友克
TOMOKATSU SAITO
アルファオメガ
ALPHAOMEGA INC.

1 はじめに

筆者らは，陰関数描画の高性能な実装へ向けて，Risa/Asir で ifplot として実装された描画法 [1] を発展させるべく，並列処理の導入の検討を行った [3]．そこでは，近年，高い処理性能がもてはやされている GPU 並列処理を用いることを検討した．実際に，NVIDIA GeForce GTX-285 を用い，共有メモリの使用，レジスタ数の制限，高占有率の確保などの最適化を行うことにより，GPU 並列処理が可能であることを示した．

本稿では，文献 [3] で用いている有理区間演算のアルゴリズムを見直すことにより高速化が可能となったので報告する．

2 有理区間数と RNS(Residue Number System)

陰関数描画法の内，最も単純な区間数による判定法を用いる場合を考える．一般には区間数の上下限値には浮動小数点数が用いられる．しかし， 数式処理的な正確さを追求するために，区間の上下限値を有理数で表現する有理区間数を用いる．演算コストは，通常の区間数や単なる多倍長整数に比べ増大するが，ここに並列処理を適用し解決を図る．

区間数の演算そのものは既知のとおりである．多項式の変数に区間数を代入して評価する際，通分した分子のみを計算することとし，区間の両端の値を RNS で表現することとする．その結果，必要な整数計算は一定の精度（ビット数）で行うことが可能となり，並列性も自動的に導かれる．RNS 表現された多倍長整数 $u = (u_1, u_2, \cdots, u_r)$ は，法として素数 $p_1, p_2, \cdots, p_r$ とすると，

$$u_k = u \mod p_k, \quad (1 \leq k \leq r)$$

*kondoh@di.kagawa-nct.ac.jp
0)本研究の一部は，科学研究費補助研究 (C) 22500011 の補助を受けて行われた．

となる．一般に，RNS 表現された整数間の演算は，モジュラー算法となり，

$$c = a + b \text{ のとき}, c_k = a_k + b_k \mod p_k, \quad (1 \leq k \leq r)$$

$$c = a - b \text{ のとき}, c_k = a_k - b_k \mod p_k, \quad (1 \leq k \leq r)$$

$$c = a \times b \text{ のとき}, c_k = a_k \times b_k \mod p_k, \quad (1 \leq k \leq r)$$

で行われる．最大の問題は RNS 表現された数の大小比較と符号判定であり，Garner による RNS 表現から混合基数表現への変換法に基づく方法を用いる．

3 計算手順の検討

文献 [2][3] では，各セルの区間演算に対し，

$$f(x,y) = \sum_{k=1}^{n} c_k \times x^{e_{xk}} \times y^{e_{yk}}$$

の x と y に有理区間数を代入することにより計算することを考える．実際には，通分して，

$$x = \frac{x_{nume}}{l_{deno}}, \qquad x_{nume} : \text{整数区間}$$

$$y = \frac{y_{nume}}{l_{deno}}, \qquad y_{nume} : \text{整数区間}$$

$$f(x,y) = \sum_{k=1}^{n} c_k \times x_{nume}^{e_{xk}} \times y_{nume}^{e_{yk}} \times l_{deno}^{o_f-(e_{xk}+e_{yk})}$$

により計算が行われた．これらをより，具体的な手順で示すと，

計算手順 1

(1) $t_1 \leftarrow x_{nume}^{e_{xk}}$ （区間モジュラー数の冪乗）

(2) $t_2 \leftarrow y_{nume}^{e_{yk}}$ （区間モジュラー数の冪乗）

(3) $T \leftarrow t_1 \times t_2$ （区間モジュラー間の乗算）

(4) $T \leftarrow T \times c_k$ （区間モジュラーとモジュラー数の乗算）

(5) $t_1 \leftarrow l_{deno}^{o_f-(e_{xk}+e_{yk})}$ （モジュラー数の冪乗）

(6) $T \leftarrow T \times t_3$ （区間モジュラーとモジュラー数の乗算）

となる．しかし，実際に代入する x や y の区間数値を考えると，0 を含む区間数は，軸上だけであり，その軸上も分割の取り方で下限か上限が 0 の場合のみとできる．よって，各象現での計算を考えると，区間（モジュラー）数の冪乗演算は，符号を調べる必要なく，下限のみと上限のみの演算でできる．また，その他のところも符号が分かっているところがある．格子を 32 スレッド（1 ワープ）の倍数に取れば，同じワープ内で軸をはさまない．これらを考慮すると，計算手順 1 を第 1 象現においては計算手順 2 のように変更できる．他の象現では，若干符号比較が必要のため分岐が入るが，同一ワープ内では同一方向に分岐するため，特に問題は起こらない．これにより，有理区間数の符号判定回数が大幅に減り，速度の向上が見込まれる．

計算手順 2

- 第 1 象限，$x = [x.lo, x.hi] \geq 0, y = [y.lo, y.hi] \geq 0$ の場合
  - **(1)** $X.lo \leftarrow x.lo_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(2)** $X.hi \leftarrow x.hi_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(3)** $Y.lo \leftarrow y.lo_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
  - **(4)** $Y.hi \leftarrow y.hi_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
  - **(5)** $T.lo \leftarrow X.lo \times Y.lo$ （モジュラー間の乗算）
  - **(6)** $T.hi \leftarrow X.hi \times Y.hi$ （モジュラー間の乗算）
  - **(7)** $t_1 \leftarrow l_{deno}^{o_f-(e_{xk}+e_{yk})}$ （モジュラー数の冪乗）
  - **(8)** $t_3 \leftarrow t_1 \times c_k$ （モジュラー数の乗算）
  - **(9)** $T \leftarrow T \times t_3$ （区間モジュラーとモジュラー数の乗算）
- 第 2 象限，$x = [x.lo, x.hi] < 0, y = [y.lo, y.hi] \geq 0$ の場合
  - **(1)** $X_1 \leftarrow x.lo_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(2)** $X_2 \leftarrow x.hi_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(3)** $Y.lo \leftarrow y.lo_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
  - **(4)** $Y.hi \leftarrow y.hi_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
    - if (e_{xk} が奇数) // $X_1 \leq X_2 \leq 0$
      - **(5)** $T.lo \leftarrow X_1 \times Y.hi$ （モジュラー間の乗算）
      - **(6)** $T.hi \leftarrow X_2 \times Y.lo$ （モジュラー間の乗算）
    - else // $0 \leq X_2 \leq X_1$
      - **(5)** $T.lo \leftarrow X_2 \times Y.lo$ （モジュラー間の乗算）
      - **(6)** $T.hi \leftarrow X_1 \times Y.hi$ （モジュラー間の乗算）
  - **(7)** $t_1 \leftarrow l_{deno}^{o_f-(e_{xk}+e_{yk})}$ （モジュラー数の冪乗）
  - **(8)** $t_3 \leftarrow t_1 \times c_k$ （モジュラー数の乗算）
  - **(9)** $T \leftarrow T \times t_3$ （区間モジュラーとモジュラー数の乗算）
- 第 3 象限，$x = [x.lo, x.hi] < 0, y = [y.lo, y.hi] < 0$ の場合
  - **(1)** $X_1 \leftarrow x.lo_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(2)** $X_2 \leftarrow x.hi_{nume}^{e_{xk}}$ （モジュラー数の冪乗）
  - **(3)** $Y_1 \leftarrow y.lo_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
  - **(4)** $Y_2 \leftarrow y.hi_{nume}^{e_{yk}}$ （モジュラー数の冪乗）
    - if (e_{xk} が奇数) // $X_1 \leq X_2 \leq 0$
      - if (e_{yk} が奇数) // $Y_1 \leq Y_2 \leq 0$
        - **(5)** $T.lo \leftarrow X_2 \times Y_2$ （モジュラー間の乗算）
        - **(6)** $T.hi \leftarrow X_1 \times Y_1$ （モジュラー間の乗算）
      - else // $0 \leq Y_2 \leq Y_1$
        - **(5)** $T.lo \leftarrow X_1 \times Y_2$ （モジュラー間の乗算）
        - **(6)** $T.hi \leftarrow X_2 \times Y_1$ （モジュラー間の乗算）
    - else // $0 \leq X_2 \leq X_1$

if (e_{yk} が奇数) // $Y_1 \leq Y_2 \leq 0$

(5) $T.lo \leftarrow X_1 \times Y_2$ （モジュラー間の乗算）

(6) $T.hi \leftarrow X_2 \times Y_1$ （モジュラー間の乗算）

else // $0 \leq Y_2 \leq Y_1$

(5) $T.lo \leftarrow X_1 \times Y_1$ （モジュラー間の乗算）

(6) $T.hi \leftarrow X_2 \times Y_2$ （モジュラー間の乗算）

(7) $t_1 \leftarrow l_{deno}^{o_f-(e_{xk}+e_{yk})}$ （モジュラー数の冪乗）

(8) $t_3 \leftarrow t_1 \times c_k$ （モジュラー数の乗算）

(9) $T \leftarrow T \times t_3$ （区間モジュラーとモジュラー数の乗算）

- 第 4 象限，$x = [x.lo, x.hi] \geq 0, y = [y.lo, y.hi] < 0$ の場合

(1) $X.lo \leftarrow x.lo_{nume}^{e_{xk}}$ （モジュラー数の冪乗）

(2) $X.hi \leftarrow x.hi_{nume}^{e_{xk}}$ （モジュラー数の冪乗）

(3) $Y_1 \leftarrow y.lo_{nume}^{e_{yk}}$ （モジュラー数の冪乗）

(4) $Y_2 \leftarrow y.hi_{nume}^{e_{yk}}$ （モジュラー数の冪乗）

if (e_{yk} が奇数) // $Y_1 \leq Y_2 \leq 0$

(5) $T.lo \leftarrow X.hi \times Y_1$ （モジュラー間の乗算）

(6) $T.hi \leftarrow X.lo \times Y_2$ （モジュラー間の乗算）

else // $0 \leq Y_2 \leq Y_1$

(5) $T.lo \leftarrow X.lo \times Y_2$ （モジュラー間の乗算）

(6) $T.hi \leftarrow X.hi \times Y_1$ （モジュラー間の乗算）

(7) $t_1 \leftarrow l_{deno}^{o_f-(e_{xk}+e_{yk})}$ （モジュラー数の冪乗）

(8) $t_3 \leftarrow t_1 \times c_k$ （モジュラー数の乗算）

(9) $T \leftarrow T \times t_3$ （区間モジュラーとモジュラー数の乗算）

4 計算機実験

文献 [3] では，NVIDIA GeForce GTX-285 を使用していたが，本稿の計算方法の変更の利点を明確に調査するため，Mac Book を用いて実験を行った．使用した計算機は，CPU Core 2 Duo 2.4GHz, NVIDIA GeForece 320M 256MB（メインメモリ上）48core, CUDA 3.2 である．また，描画対象には筆者らがよく利用している Heart 型関数を用いた．描画領域を 384×384 格子に分割した計算時間は，従来の方法では 1.2 秒必要だったところ，提案した計算方法では 0.360 秒と大幅に改善することができた．計算能力の低い GPU での実験であるが，能力の高い GTX-285 や GTX480 などでも同様の結果が得られると期待できる．

5 まとめ

GPU 並列処理を用いた陰関数描画で使用してる有理区間演算の計算法を見直すことにより，大幅に計算時間を短縮することができることを示した．本稿での実験に用いた計算機は能力の低い GPU を搭載しているものであったが，この結果は，能力の高い GTX-285 や GXT480 などでも有効に役立つことが期待できる．今後は，より汎用性のある Fermi アーキテクチャである GTX480 やそれ以降の高速 GPU での実験を行い有効性を実証することが必要である．また，

(i) RNS 演算で特に時間が費やされている符号判定の見直し,

(ii) 多項式の区間数での（多点）評価法の検討,

(iii) Sturm 法による解の分離と存在区間の特定

なども行う必要があるだろう.

参 考 文 献

[1] 齋藤友克，竹島卓，平野照比古，グレブナー基底の計算 実践篇，東京大学出版会，2003.

[2] 鈴木省吾，GPU を使用した陰関数グラフ描画の高速処理法，電気通信大学大学院電気通信学研究科修士論文，2010.

[3] 村尾裕一，鈴木省吾，近藤祐史，齋藤友克，有理区間数と GPU 並列処理について，第 19 回日本数式処理学会大会報告，数式処理，**17**(2), 2011, 28–31.