5H-08

GPU を用いたポラード・ロー法の実装

中村 優† 久保田 光一‡

中央大学大学院 理工学研究科 情報工学専攻†‡

要約: GPU はその構造から高い計算能力をもち, 様々な分野において現在使用されている. 今回, 計算処理の対象として楕円曲線離散対数問題の解法として幅広く用いられているポラード・ロー法の計算を行うプログラムを作成し, CPU 上で動作するプログラムと実行性能の比較・評価を行った.

キーワード: GPGPU, CUDA, 並列計算

1 背景

近年, 各種計算に GPU(Graphics Processing Units) を用いる動きが盛んになっている. これは CPU の性能向上が各種問題により頭打ちになっているのに対し, GPU は SIMD 型という単純な構造になっているため, 現在においても性能の向上率が高いのが特徴である. このため GPU を画像処理だけではなくその他の汎用計算に用いる GPGPU に関連する技術が開発されている.

2 目的

現在, 複数の計算機を用いた並列化による実装が多用されている楕円曲線離散対数問題の解法であるポラード・ロー法由来の手法 [1][2] を GPU の特性に合わせた実装を行い, CPU との計算実験を行うことでその性能の評価を実施することを目的とする.

3 GPGPU

GPGPU(General Purpose computing on Graphics Processing Units) は GPU を画像処理以外の処理・計算に用いることである.

各種処理を行うコアが CPU では最大でも十数個であるのに対し, GPU では数十〜数千と多数搭載されている他, スレッドと呼ばれるプログラムの実行の最小単位の数も CPU に比べ, 数十倍も多く実行できる.

GPGPU に関連する技術は様々なものがあるが今回は NVIDIA 社の CUDA(Compute Unified Device Architecture) を用いて, プログラムを作成した.

4 楕円曲線離散対数問題

素体 $\mathbb{F}_p$ 上の楕円曲線を $E : y^2 = x^3 + ax + b$ $(4a^3 + 27b^2 \neq 0$ かつ $a, b \in \mathbb{F}_p)$ と定義された曲線を楕円曲線と呼び, $E(\mathbb{F}_p)$ と表す. その曲線上の点の集合に仮想的な点である無限遠点 O を加えたものを有理点と呼ぶ.

楕円曲線上の点 P を l 回加算する演算をスカラー倍算と呼び, 演算規則を利用することで容易に行うことができる. しかしその逆の演算は楕円曲線離散対数問題 (ECDLP) と呼ばれ, 楕円曲線の位数が大きくなるほど解くのが難しくなる.

4.1 ポラード・ロー法

ポラード・ロー法は楕円曲線離散対数問題に対して経験的に優れているものとして知られている解法である [1][2].

これは位数 n の楕円曲線 $E(\mathbb{F}_p)$ とその点 P, Q に対して,

$$c' \times P + d' \times Q = c'' \times P + d'' \times Q \quad (1)$$
$$(c', c'', d', d'' \in \{0, 1, \cdots, l-1\})$$

を満たす自然数 c', c'', d', d'' を見つけたとき,

$$Q = (c' - c'')(d'' - d')^{-1} \times P \quad (2)$$

が得られるのでスカラー倍算の特性より $Q = l \times P$ を満たす l を求めることができる.

5 実装したプログラムの概要

今回実装したプログラムでは GPU 上にスレッドを M 個用意し, それぞれを異なる 1 台の計算機とみなし, ランダムに選ばれた初期値を与え, 値を更新させ式 (1) を満たす c', c'', d', d'' が見つけるまで繰り返し実行させる. また, GPU が実行する前に行う各種処理や各スレッドが更新した値の比較・記憶の処理をメインマシンの CPU 上で実行させることにした.

5.1 探索部分

探索部分は GPU のスレッドとして多数実行される部分で最初の一回目の動作では各スレッドに対する初期値の設定を行い GPU 側のメモリであるグローバルメモリに保存, それ以降は式 (1) を満たす点が見つかるまでグローバルメモリに保存されている値を用いて計算・値更新を行い, 保存する処理を繰り返し行う.

5.2 計算結果の比較・記憶

グローバルメモリに保存された各スレッドの計算結果は全スレッドが探索部分を実行後, 同期が取られてからメインマシン側のメモリに転送される. その後, 計算結果の比較を行うがメインマシン側に比較用のリストを用意しておき, 2 分探索木を用いて比較及び記憶を行う.

この比較・記憶処理は GPU 内でも実装は可能であるがアクセス速度が低速なグローバルメモリに頻繁にアクセスするため, 処理に時間がかかってしまう可能性がある. 実験を行ったところ, 全ての処理を GPU で行う場合の処理時間は値の比較・記憶を CPU で行う場合の処理時間に比べ最大で約 8 倍程度, 時間がかかることがわかった. そのため, 今回は GPU からメインマシンへのデータの転送や動作の切り替えのオーバーヘッドが余分にかかってしまうが値の比較・記憶はメインマシン上で行うことにした.

5.3 プログラムの流れ

以下の手順は [2] に基づく.

1. 任意の分割数 L, 分割関数 $H(X)$, 特徴点の条件, 位数 n の楕円曲線 $\mathbb{F}_p$, 楕円曲線上の点 P, Q を決定する.
2. ランダムウォーク関数に用いる L 個の楕円曲線上の点 R_i をランダムに選択する.

$$\begin{aligned} R_0 &= s_0 \times P + t_0 \times Q \\ &\vdots \\ R_{L-1} &= s_{L-1} \times P + t_{L-1} \times Q \\ &(s_0, \cdots, s_{L-1}, t_0, \cdots, t_{L-1} \in \{0, 1, \cdots, n-1\}) \end{aligned}$$

Implementation of the Pollard's rho algorithm with GPU

† Masaru NAKAMURA, Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University

‡ Koichi KUBOTA, Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University

Copyright ©2016 Information Processing Society of Japan. All Rights Reserved.

3. M 個のスレッドにそれぞれ初期値 X_i を決定する.

$$\begin{aligned} X_0 &= \quad c_0 \times P + d_0 \times Q \\ &\vdots \\ X_{M-1} &= c_{M-1} \times P + d_{M-1} \times Q \\ &(c_0, \cdots, c_{M-1}, d_0, \cdots, d_{M-1} \in \{0, 1, \cdots, n-1\}) \end{aligned}$$

4. GPU 上でスレッド番号 i の点 X_i に対して, 分割関数 $H(X)$ とランダムウォーク関数を用いて点を更新し, 特徴点の条件に当てはまるまで繰り返す.

$$\begin{aligned} j &= H(X_i) \\ X_i &= X_i + R_j \\ c_i &= c_i + s_j \bmod n,\ d_i = d_i + t_j \bmod n \end{aligned}$$

5. 全スレッドの点の更新が終了したら, 全ての特徴点をメインマシンのメモリに転送する.
6. メインマシン側でそれぞれ特徴点を比較する.
7. $X_i = X_j$, かつ $d_i \neq d_j$ を満たすような点が見つかれば 8 へ. そうでなければ点を記憶し, 4. へ
8. $c' = c_i$, $c'' = c_j$, $d' = d_i$, $d'' = d_j$ とし, $l = (c' - c'')(d'' - d')^{-1} \bmod n$ を計算, 結果 l を出力して終了.

6 計算機実験

実験環境及び, 共通のパラメータを以下の通りとした.

表 1 実験環境

CPU	Core i7-4770 @3.40GHz
GPU	NVIDIA Quadro 600(1280MHz, 1GB)
GPU コア数	96 コア
CUDA バージョン	5.0
GCC バージョン	4.4.7

表 2 共通パラメータ

L(分割数)	16 または 32
H(分割関数)	与えられた点の x 座標を L で剰余した値
特徴点の条件	なし (更新した点を全て記憶する)

実験に用いた楕円曲線は表 3 の通りとし, 楕円曲線 $E(\mathbb{F}_p)$ の有理点の個数を $\#E(\mathbb{F}_p)$ と表す.

またそれぞれの楕円曲線に対して, ランダムに (P, Q) の組を 100 個用意し, 一つの組に対してその答えが求まるまでの平均時間を評価基準とする.

表 3 実験に用いた $\mathbb{F}_p$ 上の楕円曲線

a	b	p	$\#E(\mathbb{F}_p)$
0	3	7	13
6	4	13	17
0	6	31	43
20	57	61	67
47	82	127	139
199	136	251	257
288	144	509	521
245	629	1021	1039
1055	1267	2039	2099
112	1401	4093	4211
6755	5906	8191	8311
13025	7704	16381	16603
28602	30160	32749	32941

a	b	p	$\#E(\mathbb{F}_p)$
11891	53202	65521	65557
40076	725	131071	131357
116967	161973	262139	262733
321727	465094	524287	525377
0	55735	1048573	1049791
78563	1068023	2097143	2099479
1005643	1087588	4194301	4197161
7634512	2040187	8388593	8392141
8000000	12057779	16776289	16780201
12396	12110695	33524453	33532307
34012396	54247962	66898873	66913211
74119846	104260955	134137733	134152367

6.1 実験結果

プログラム内の各種変数を 64 ビットの long long int 型に設定し, 位数が 13〜134152367 のデータに対して CPU とスレッドの数を 2〜512 までそれぞれ設定して動作させた. 分割数を 16 に設定した実験結果を図 1 に, 32 に設定した実験結果を図 2 に示している. 横軸は位数の大きさ, 縦軸は (P, Q) の組を一つ与えて, その結果が出るまでの平均時間を表している. また両軸とも対数をとってある.

実験結果より位数が 8311 のときに CPU が 0.0036 秒に対し, 512 スレッドで動作させた GPU が 0.0027 秒で初めて性能を上回っていることがわかる.

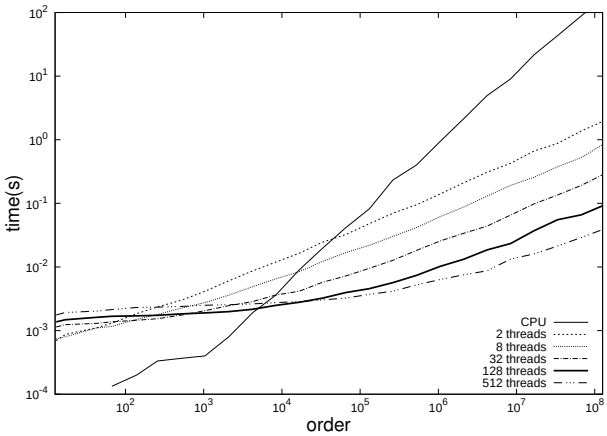

図 1 分割数を 16 に設定した時の CPU と GPU の実行結果

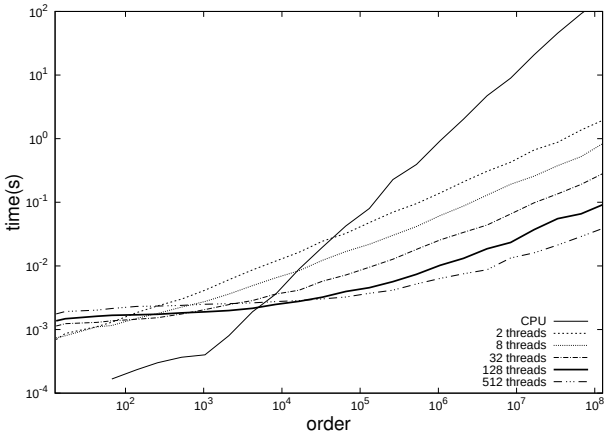

図 2 分割数を 32 に設定した時の CPU と GPU の実行結果

7 考察

今回, 楕円曲線離散対数問題に用いられているポラード・ロー法の GPU 上への実装, 及び作成したプログラムを用いて計算実験を行うことで CPU と比べて, 一番大きい位数である 134152367 の時には 512 スレッドで動作させた GPU が約 0.05 秒, CPU が約 220 秒と約 4400 倍の差が出ている.

参考文献

[1] 伊豆 哲也, **楕円曲線暗号入門**, 立教大学理学部数学科講義「計算機緒論 2」講義補助資料, 2013 年 8 月.
[2] Darrel Hankerson, Alfred Menezes, Scott Vanstone, **Guide to Elliptic Curve Cryptography**, Springer, 2004.
[3] 青木 尊之, 額田 彰, **はじめての CUDA プログラミング**, 工学社, 2009.

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.