数理解析研究所講究録
第1148巻 2000年 206-211

離散対数問題への PVM(Parallel Virtual Machine) の適応

森田 雄介 (Yusuke Morita), 茂木 和弘 (Kazuhiro Motegi), 五十嵐 善英 (Yoshihide Igarashi)

群馬大学工学部情報工学科
〒 376-8515 桐生市天神町 1-5-1
tel: 0277-30-1829
email: {morita,motegi,igarashi}@comp.cs.gunma-u.ac.jp

1 はじめに

公開鍵暗号方式の多くは，有限体上の乗法群における素因数分解および離散対数問題の計算困難性に基づいている．素因数分解と離散対数問題ともに効率的なアルゴリズムを用い，さまざまなプロジェクトで数値実験が行われ成果を挙げている．離散対数問題は位数が大きくなると計算量が多くなり，実際に計算することが困難になる．このため数値実験には，高性能な計算機を利用しなければならない．本研究では，複数台の計算機をネットワークで結合し，仮想的に並列計算を行う PVM(Parallel Virtual Machine) を用いて離散対数問題の数値実験を行った．PVM を用いた大規模な分散システム上で離散対数問題の数値実験を行う場合，計算の精度，誤差，信頼性や動的なネットワーク構成などの問題を解決しなければならない．

2 離散対数問題

q を素数，g を有限体の非ゼロ元の集合 ${F_q}^*$ における生成元 (位数 $q-1$ の元) とする．q を法として $y=g^x$ とするとき，$x=\log_g y$ を y の g に対する離散対数という．つまり，離散対数問題とは (y,g,q) が与えられたとき $x\in F_q^*$ を求める問題である．

離散対数を求めるアルゴリズムは，次の 2 つに大別できる．

1. ${F_q}^*$ 上の離散対数の計算時間が ${F_q}^*$ の位数 $q-1$ の最大素因数のサイズ N に依存するアルゴリズムで，その計算時間は N に関して指数時間のオーダの実行時間になるもの．
2. 指数計算法 (index calculus method) とよばれる手法で，${F_q}^*$ 上の離散対数の計算時間が，q のサイズの準指数時間オーダの実行時間となるもの．

前者のアルゴリズムとしては Shanks[1], Pohlig-Hellman[2], Pollard[3] のアルゴリズムが有名である．後者としては，Coppersmith-Odlyzko-Schroeppel[4], Gordon[5], Adleman[6] のアルゴリズムが知られている．

2.1 定理

2.1.1 Chinese remainder theorem

整数 $m_1,\cdots,m_r$ をそれぞれ互いに素であるとする．つまり，$i\neq j$ に対して 最大公約数 $g.c.d.(m_i,m_j)=1$. $a_1,\cdots,a_r$ を整数とし，次の連立合同式を考える．

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

このとき，$M = m_1 m_2 \cdots m_r$ を法として一意な解 x が存在し，その解は

$$x \equiv \sum_{i=1}^{r} a_i M_i y_i \pmod M$$

により与えられる．ただし，$1 \leq i < r$ に対して，$M_i = M/m_i$ かつ $y_i = M_i^{-1} \pmod{m_i}$ である．

2.2 Pohlig-Hellman のアルゴリズム

$y \equiv g^x \pmod q$ を満たすような x を $0 \leq x < q-1$ の範囲で見つける代表的なアルゴリズムに，Pohlig-Hellman のアルゴリズムがある．Pohlig-Hellmanのアルゴリズムは，$q-1$ が小さな素数の積で表せるとき，底 g における $y \in F_q^*$ の離散対数を効率よく計算する (g は y の生成元)．このアルゴリズムは，素因数分解された各素因数ごとに処理を行なう．それらの処理は互いに依存関係が無いため，並列化が可能であると考えた．そのため本研究では，Pohlig-Hellman のアルゴリズムを用いた．

2.2.1 アルゴリズム

[Pohlig-Hellman のアルゴリズム]

Input: y, g, q;

Output: x;

step1: $q - 1 = \prod_{k=0}^{n} p_k^{\alpha_k}$ に素因数分解する．

step2: 各素数 $p_k (0 \leq k \leq n)$ について，$x \pmod{p_k^{\alpha_k}}$ を求める．

step2-a: 1 の p_k 乗根 $r_{p_k,j} = g^{j(q-1)/p_k}$ $(j = 0, 1, \ldots, p_k - 1)$ を計算し，
$\{r_{p_k,j}\}$ のテーブルを作成する．

step2-b: $x \equiv x_0 + x_1 p_k + \cdots + x_{\alpha-1} p_k^{\alpha_k - 1} \pmod{p_k^{\alpha_k}}$
$(0 \leq x_i < p_k)$ とし，テーブルを利用し，$x_1, x_2, \ldots, x_{\alpha_k - 1}$ を求める．

step3: Chinese remainder theorem により x を求める．

"step2-b" $x \pmod{p_k^{\alpha_k}}$ の各 x_i の求め方を詳しく示す．
$p_k = p, \alpha_k = \alpha$ とし，各式は p を法とする．

- x_0 の計算

  1. $y^{(q-1)/p}$ を計算する．($y^{q-1} = 1$ なので $y^{(q-1)/p}$ は 1 の p 乗根の 1 つ．)
  2. $y = g^x$ より，
     $y^{(q-1)/p} = g^{x(q-1)/p} = g^{(x_0 + x_1 p + \cdots + x_{\alpha-1} p^{\alpha-1})(q-1)/p} = g^{x_0(q-1)/p} = r_{p,x_0}$
  3. $y^{(q-1)/p}$ と テーブル $\{r_{p,j}\}_{0 \leq j < p}$ を比較する．
  4. $y^{(q-1)/p} = r_{p,j}$ となるテーブル $\{r_{p,j}\}_{0 \leq j < p}$ のインデックス j の値を x_0 とする．

- x_1 の計算

  1. y を $y_1 = y/g^{x_0}$ で置き換える．
  2. $x - x_0 \equiv x_1 p + \cdots + x_{\alpha-1} p^{\alpha-1} \pmod{p^\alpha}$ とする．
  3. y_1 は p 乗数より ${y_1}^{(q-1)/p} = 1$ かつ，
     ${y_1}^{(q-1)/p^2} = g^{(x-x_0)(q-1)/p^2} = g^{(x_1 + x_2 p + \cdots)(q-1)/p} = g^{x_1(q-1)/p} = r_{p,x_1}$
  4. ${y_1}^{(q-1)/p^2}$ と テーブル $\{r_{p,j}\}_{0 \leq j < p}$ を比較する．
  5. ${y_1}^{(q-1)/p^2} = r_{p,j}$ となるテーブル $\{r_{p,j}\}_{0 \leq j < p}$ のインデックス j の値を x_1 とする．

- x_i の計算
  1. $y_i = y/g^{x_0+x_1p+\cdots+x_{i-1}p^{i-1}}$ 置き換える.
  2. $x - x_0 - \cdots - x_{i-1}p^{i-1} \equiv x_ip^i + \cdots + x_{\alpha-1}p^{\alpha-1} \pmod{p^{\alpha}}$
  3. y_i は p^i 乗数より ${y_i}^{(q-1)/p^i} = 1$ かつ,
     ${y_i}^{(q-1)/p^{i+1}} = g^{(x_i+x_{i+1}p+\cdots)(q-1)/p} = g^{x_i(q-1)/p} = r_{p,x_i}$
  4. ${y_i}^{(q-1)/p^{i+1}}$ と テーブル $\{r_{p,j}\}_{0\leq j<p}$ を比較する.
  5. ${y_i}^{(q-1)/p^{i+1}} = r_{p,j}$ となるテーブル $\{r_{p,j}\}_{0\leq j<p}$ のインデックス j の値を x_i とする.

この結果, $x \pmod{p^{\alpha}}$ を得る.

3 実験と評価

3.1 仮想並列計算機

実際の並列計算機と, 仮想並列計算機では以下の相違点がある.

表 1: 並列計算機の比較

	並列計算機	仮想並列計算機
コスト	大	小
通信速度	高速	多様
プロセッサ	同一	多様
計算機の追加, 削除	困難	容易
負荷	一定	変動
障害発生確率	低い	高い

仮想並列計算機では, 安価で拡張性の高いシステムの構築が可能である. この利点を生かすために, 通信速度, プロセッサの多様性, 負荷の変動や障害発生などの問題点の改良が不可欠である.
今回はこのうち, プロセッサについて考察した.

3.2 PVM

PVM[8, 9] は米国のオークリッジ国立研究所を中心に開発された, メッセージパッシングによる並列計算を行なうためのソフトウエアであり, 動作するマシンの種類が多いことから広く利用されてきている. PVM は Parallel Virtual Machine つまり仮想並列計算機を意味している. PVM は, ネットワークに接続された異機種コンピュータ群を, 単一の並列コンピュータとして利用することを可能にするソフトウエアシステムである. これにより, 多数のコンピュータの持つ計算パワーを, 1 つの大規模計算問題に結集して処理を行なうことができる. 本実験では, pvm-3.4.2 を使用した.

PVM では 2 つのプログラムを用意する必要がある. ひとつは全体を制御する "メインホスト" 用, もうひとつはメインホストから依頼を受け, タスクを行なう "ホスト" 用である.

3.3 実験環境

本実験では, 表 2 で示した 12 台の計算機 (ホスト) を使用し, "OS" は Solaris と Linux の 2 種類を用いた. 各計算機は, ハブを介して 10BASE-T で接続されている. 以下の実験ではホストを "No." で示した数字で表したが, この数字は計算速度が速い順に割り当てた. 本実験では, "No.2" の "beast" をメインホストとした.

表 2: 実験環境

No.	ホスト名	OS	CPU	Clock(MHz)	Memory(MB)
1	pinocchio	Linux	PentiumIII	500	192
2	**beast**	**Linux**	**PentiumII**	**450**	**128**
3	goofy	Linux	Celeron	433	128
4	dumbo	Linux	AMD-K6 2	400	128
5	sonic	Linux	PentiumII	366	128
6	hercules	Linux	PentiumII	333	256
7	snowwhite	Solaris7	UltraSPARC-II	360	640
8	aladdin	Solaris7	UltraSPARC-IIx2	360	256
9	minnie	Solaris2.6	UltraSPARC	248	256
10	mickey	Solaris2.6	UltraSPARCx2	168	256
11	tarzan	Linux	Pentium	200	64
12	taurus	Solaris2.4	SuperSPARC	50	128

4 提案手法

4.1 準備実験

Pohlig-Hellman のアルゴリズムは，全体の計算時間のうちテーブル $\{r_{p_k,j}\}_{0\leq j<p_k}$ の作成に最も時間がかかる．そこで，各素因数 p_k のテーブルの作成，探索を並列化した．準備実験では図 1 に示すように，プロセッサの性能にかかわらずテーブル $\{r_{p_k,j}\}_{0\leq j<p_k}$ の j の範囲を均等に分割した．つまり，Pohlig-Hellman アルゴリズムの "step2-a" のテーブルの作成と，"step2-b" の各 x_i を求める部分を並列化した．

4.2 提案手法

提案手法では、プロセッサの計算性能に応じてテーブルの分割した (図 2)．プロセッサの性能を比較するため，各ホストは起動と同時に計算速度の目安とする性能測定のルーチン (使用する多倍長演算ライブラリの四則演算を 150 万回繰り返す) を実行し，メインホストに返答する．メインホストは各ホストの応答時間に応じて性能比を求め，計算能力に応じてテーブルを割り当てる．

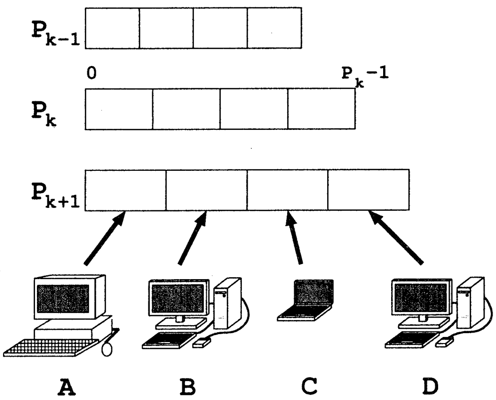

図 1: 準備実験

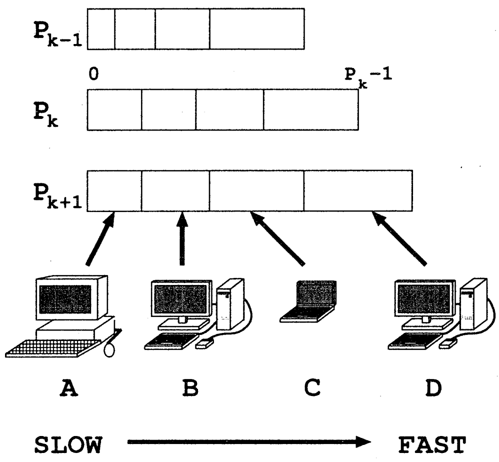

図 2: 提案手法

4.2.1 手順

以下に提案手法の処理手順を示す.
入力として y, g, q をとり，結果 x を出力する.

[提案手法のアルゴリズム]

$\boxed{\text{メインホスト}}$

step1: $q-1=\prod_{k=0}^{n} p_k^{\alpha_k}$ に素因数分解する.

- ホスト (m 台) を起動する.
- ホストの応答時間から性能比を計算する.

step2: 各 p_k について，$x \pmod{p_k^{\alpha_k}}$ を求める.

- 作成するテーブルの範囲 $(0 \leq j < p)$ を各ホストの能力に応じて分割し，それぞれの範囲を $j_l(0 \leq l < m)$ とする.

step2-a: データ (p_k, α_k, j_l) を各ホストへ送信する.

- * 全てのホストからテーブル完成の応答を待つ.

step2-b: $x \equiv x_0 + x_1 p_k + \cdots + x_{\alpha_k - 1} p_k^{\alpha_k - 1} \pmod{p_k^{\alpha_k}}$ とおく.
各 $x_i (0 \leq i < \alpha_k)$ について，
$y_i = y / g^{x_0 + x_1 p_k + \cdots + x_{i-1} p_k^{i-1}}$ を計算する.

- * データ $(y_i^{(q-1)/p_k^{i+1}})$ を全てのホストへ送信する.
- * 解の存在したホストからデータ (x_i) を受信する.

step3: Chinese remainder theorem により x を求める.

$\boxed{\text{ホスト}}$

step1: 起動と同時に性能測定のルーチンを実行する.

- メインホストにルーチンの終了を伝達する.

step2: データ (p_k, α_k, j_l) を受信する.

step2-a: 1 の p_k 乗根 $r_{p_k, j} = g^{j(q-1)/p_k}$ を計算し，担当範囲 j_l のテーブル $\{r_{p_k, j_l}\}$ を作成する.

- * メインホストに担当テーブルの完成を伝達する.

step2-b: データ $(y_i^{(q-1)/p_k^{i+1}})$ を受信する.
受信データとテーブルを比較する.

- * 担当テーブルに解が存在したとき，データ (x_i) を送信する.

5 結果

5.1 準備実験との比較

実験値には $q = 10^{20} + 42709$，$q-1 = 2^2 \cdot 13 \cdot 107 \cdot 113 \cdot 1609 \cdot 86771 \cdot 113921$ (最大素因数 6 桁) を用いた.
ホストは，速い計算機と遅い計算機を交互に No. 2, 7, 3, 8 の順に追加した．図 3 は data1 が準備実験の手法，data2 が提案手法である．準備実験の手法では，遅い計算機を追加すると全体の計算時間は遅くなったが，提案手法ではホストを増やすごとに計算時間を短縮することができた.

5.2 ホストを増やしての実験

次にホスト数と計算時間のグラフを図 4 に示す．実験値は $q = 10^{20} + 38583$，$q-1 = 2 \cdot 3^2 \cdot 643 \cdot 5483 \cdot 139703 \cdot 1127957$ (最大桁数 7 桁) を用いた.
ホストは，data_f2s では No. 2, 3, {1, 6}, {4, 5}, {7, 8, 9, 10, 11, 12} の順に速い計算機を中心に遅い計算機を追加し，data_s2f では No. 2, 7, {8, 10},{9, 12}, {3, 6}, {1, 11}, {4, 5} の順に遅い計算機を中心に速い計算機を追加した ({} は同時に追加).
ホストの追加の仕方によりグラフの傾向は異なるが，どちらの場合も計算機を増やすごとに計算時間を短縮できた.

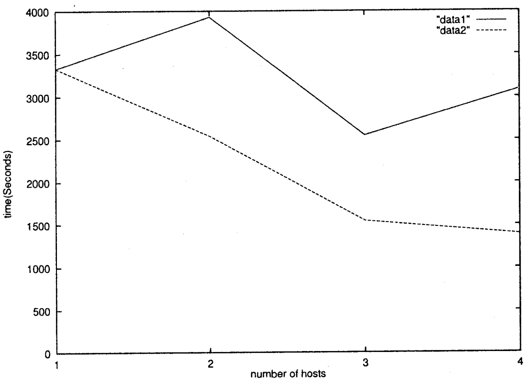

図 3: 準備実験との比較 (ホスト台数，計算時間)

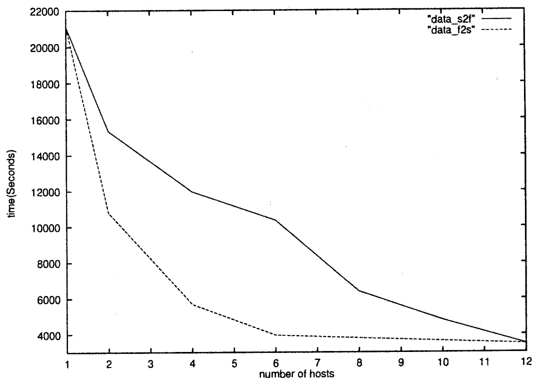

図 4: ホスト 12 台での実験 (ホスト台数，計算時間)

6 まとめ

Pohlig-Hellman のアルゴリズムで最も計算時間の必要なテーブル $\{r_{p_k,j}\}$ を並列に利用できるようにし，PVM を用いて数値実験を行った．このとき，計算速度に差がある計算機の利用を考慮しタスクの割り当てを行った．その結果，台数を増加させることにより並列の効果が上がる結果が得られた．今回の結果を踏まえ，今後以下の点を考慮に入れシステムを構築していく．

・高並列化: アルゴリズムの特性上，計算時間は $q-1$ の最大素因数に依存する．そのため，最大素因数が小さければ (10 桁前後)，q が 100 桁でも処理は可能である．しかし，それ以上の数を考慮すると現状のアルゴリズムでは膨大な時間がかかる．そのため大規模分散システムに適したアルゴリズムを提案する必要がある．
・動的なシステム: 計算の規模が大きくなるほど障害が発生しやすくなる．その場合，障害の発生した計算機を検出，削除する必要がある．また長時間の処理に対応するため，計算の途中でも計算機の追加，削除ができるようにする．このようにシステムに拡張性を持たせる必要がある．
・動的なスケジューリング: 動的なシステムを導入することで，タスクの分割にも改良が必要になる．障害などで削除する場合には，その計算機に割り当てたタスクを別の計算機に振り分けたり，追加する場合には，タスクの再分割を行なう必要がある．そのためには，動的なシステムに対する性能評価を行ないタスクを分割する必要がある．

参考文献

[1] D. R. Stinson, "Cryptography: Theory and Practice", CRC Press, Boca Raton, Florida, 1995.

[2] S. C. Pohlig, M. E. Hellman, "An improved algorithm for computing algorithms over $GF(p)$ and its cryptographic significance", IEEE Transactions on Information Theory, pp.106-110, 1978.

[3] J. M. Pollard, "Theorems on factorization and primality testing", Proc. Cambridge Philos. Soc. pp.521-528, 1974.

[4] D. Coppersmith, A. Odlyzko, R. Schroeppel, "Discrete logarithms in $GF(p)$", Algorithmica 1, pp.1-15, 1986.

[5] D. Gordon, "Discrete logarithms in $GF(p)$ using the number field sieve", SIMA J. Discrete Math. 6, pp.124-138, 1993.

[6] L. M. Adleman, "A subexponential algorithm for discrete logarithms over all finite fields", CRC Press, Boca Raton, Florida, 1995.

[7] N. Koblitz, "A course in Number Theory and Cryptography", Springer-Verlag, pp.97-106, 1987.

[8] "PVM 公式ホームページ", http://www.epm.ornl.gov/pvm/

[9] "PVM3 ユーザーズガイド＆リファレンスマニュアル日本語版", http://phase.etl.go.jp/contrib/PVM-j.