分散並列処理システムにおける高速化と持続性の研究

分散並列処理システムにおける高速化と持続性の研究

鹿児島大学工学部研究報告　第44号（2002）

分散並列処理システムにおける高速化と持続性の研究

鶴沢　偉伸*　　中山　茂**

Studies on Speeding and Persistency in Distributed Parallel Process System

Hidenobu TSURUSAWA and Shigeru NAKAYAMA

Distributed parallel process is easily realized because of increasing computer's performance and popularization of computer networks. In general, distributed parallel process is effective in speeding up processes. Because prime factorization takes long time in finding the prime factors of large composite numbers, we apply distributed parallel process to prime factorization in the elliptic curve method. The experiments on speeding up are carried out with 50 PC units connected to a network. Following the experiments, we do experiments to verify performance of different OSs; Windows, Linux, MacOS. In order to guarantee persistency of continuously longtime process, we combine ORDB (Object Relational DataBase) with the distributed parallel process. The experimental results indicate that the distributed parallel process is obviously effective in speeding up process and ORDB affects the system performance.

Keywords: distributed parallel process, object relational database, prime factorization, elliptic curve method

1. はじめに

ハードウェア技術は目覚しく進歩しているが、計算機の処理能力を飛躍的に向上させることは難しい。ネットワークが普及した現在では、複数の計算機を接続して分散処理システムを構築し、高速化する環境が整ってきた。分散処理システムの実現方法も色々と研究されており、最近はインターネット環境を利用した分散処理システムも実現している。分散処理の目的はいくつかあるが、処理を高速化する手段として利用されることが多い。分散処理システムを実現する方法としては、各計算機に用意したプログラム同士で通信を行うプロセス間通信や、クライアント・サーバ方式などがこれまで採用されている。さらに、業務アプリケーションのようにデータ中心の処理が主体であれば、リレーショナル・データベースを用いることが多い。プログラム言語Javaの普及により分散オブジェクトが容易に実現できるようになり[1),2)]、Sun Microsystems社などではオブジェクトを共有するシステムとしてオブジェクト共有空間を研究している[3)]。また、従来のリレーショナル・データベースと同様に、オブジェクトがデータベースで管理できるオブジェクト・リレーショナル・データベース(ORDB)も製品化されている。

2002年8月31日受理

*　宮崎産業経営大学経営学部経営学科
**　鹿児島大学工学部情報工学科

素因数分解は古くから数論の分野で研究されており、実用的な応用のないものとされていたが、セキュリティ技術で公開鍵暗号方式が提案されて、素因数分解の難しさを応用した RSA 暗号が広く使用されるようになったため[4]、素因数分解が注目されるようになった。オブジェクト共有空間を用いた分散並列処理システムを、分散並列処理が可能な楕円曲線法による素因数分解に応用し、計算の高速化を検証する。

分散並列処理システムをインターネット環境へ応用するには、異機種の計算機に対応することが重要であるが、すべてのオペレーティング・システムに対応したプログラムを用意することは簡単ではない。Java は異機種の計算機でそのまま動作するように、インタープリタ方式が採用されている。オブジェクト共有空間による分散並列処理システムを異機種の計算機を用いて構築し、素因数分解の実験で計算機ごとの処理能力を検証する。

オブジェクト共有空間は、計算機のメモリ上でオブジェクトを管理しており、高速であるが、システムが停止するとオブジェクトがすべて失われてしまう。オブジェクト共有空間のオブジェクトの持続性を保証するために、ORDB を組み込んだシステムを提案し、処理時間に及ぼす影響を検証する。従来のリレーショナル・データベースはデータの管理を目的としたものであるが、ORDB はデータ以外にオブジェクトをデータベースで管理する機能を持っている。オブジェクトの持続性を保証するには、ディスク装置の使用が不可欠で、処理速度への影響は避けられない。

2. 楕円曲線法による素因数分解の原理

$p-1$法は Pollard が考案したもので、楕円曲線法による素因数分解の雛型になったアルゴリズムである。素数 p で割り切れない整数 a に対して、

$$a^{p-1} \equiv \mathrm{mod}\ p \tag{1}$$

が成り立つ。ここで、合成数 n の素因数を p とし、$p-1$ が小さな素数の積であるとすれば、$a^{p-1}-1$ は p で割り切れるので、最大公約数 $\gcd(a^{p-1}-1, n)$ を p が割り切ることになる。実際には、素因数 p がわからないため、$a^{p-1}-1$ を計算することはできない。そこで、素数のべき乗の積を選んで、

$$k = 2^{e_2} \cdot 3^{e_3} \cdot 5^{e_5} \cdots r^{e_r} \tag{2}$$

とし、$\gcd(a^{k-1}-1, n)$ を計算する。合成数 n が $p-1 \mid k$ となる素因数 p を持てば、p は a^{k-1} を割り切り、

$$\gcd(a^k-1, n) \geq p > 1 \tag{3}$$

となる。$\gcd(a^k-1, n) \neq n$ のとき、n の因数が見つかり、$\gcd(a^k-1, n) = n$ ならば、別の a を選んで計算をやり直す。処理手順をまとめると、次のようになる。

(1) 上限の素数 K を決め、小さな素数のべき乗を因子に持つ整数 k を選ぶ。

$$k = LCM[2, 3, \cdots, K] \tag{4}$$

(2) $1 < a < n$ を満たす任意の整数 a を選ぶ。

(3) $\gcd(a, n)$ を計算する。1 より大きい場合、n の因数が見つかる。

(4) $D = \gcd(a^k-1, n)$ を計算する。$1 < D < n$ であれば、D は n の因数である。

(5) $D = 1$ ならば、k の値を大きくし、(1)項から計算をやり直す。$D = n$ であれば、(2)項へ戻り、別の a を選んで計算をやり直す。

楕円曲線：$y^2 = x^3 + ax + b$ では、楕円曲線上の 2 点 P_1, P_2 を加算した点 $P_1 + P_2$ は、2 点 P_1, P_2 を通る直線と楕円曲線の交点 $P_1 * P_2$ を見つけ、点 $P_1 * P_2$ と無限遠点 O を通る直線（x 軸との垂直線）が交差する点となる。点 $P_1 * P_2$ と x 軸に関して対称になる点である（図－1 参照）。2 点 $P_1(x_1, y_1), P_2(x_2, y_2)$ に対して、加算点 $P_3 = P_1 + P_2$ の座標を (x_3, y_3) とすると、次のようになる。

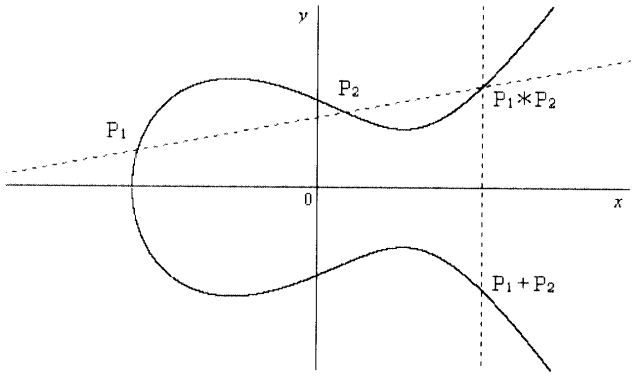
$x_1 \neq x_2$ の場合：

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \tag{5}$$

$x_1 = x_2, y_1 = y_2 \neq 0$ の場合：

$$\lambda = \frac{3{x_1}^2 + a}{2y_1} \tag{6}$$

とおくと、



図－1 楕円曲線の群法則

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \quad (7)$$

$x_1 = x_2, y_1 = -y_2$ の場合： $P = O$

となる。ここで、$P_2 = P_1$ とすると、$P_1 + P_2 = 2P_1$ となり、点 P_1 を 2 倍する計算となる。これが 2 倍公式と呼ばれている。

Lenstra は、$p-1$ 法が p を法とする剰余類が群構造を持つという事実に基づいていることに注目し、楕円曲線が同様の目的に使える群構造を持つことに気づいた。したがって、楕円曲線：$y^2 = x^3 + ax + b$ を使って、次の手順で合成数 n の素因数 p を求めることができる。

(1) 乱数より 1 と n の間にある整数 a, x_1, y_1 を選ぶ。

(2) a, x_1, y_1 を使って、楕円曲線： $y^2 = x^3 + ax + b$ より b の値を求める。

$$b = y^2 - x^3 - ax \quad (8)$$

(3) $\gcd(4a^3 + 27b^2, n) \neq 1$ であることを確認する。n に等しい場合、(1)項に戻って値を選び直す。1 から n の間であれば、因数である。

(4) 上限の素数 K を決め、小さな素数のべき乗を因子に持つ整数 k を選ぶ。

$$k = LCM[2,3,\cdots,K] \quad (9)$$

(5) $P(x_1, y_1)$ の k 倍点を計算する。

$$kP = \left(\frac{a_k}{d_k^2}, \frac{b_k}{d_k^3}\right) \quad (10)$$

(6) $D = \gcd(d_k, n)$ を計算する。$1 < D < n$ であれば、D は n の因数である。

(7) $D = 1$ ならば、(1)項に戻って別の楕円曲線で計算をやり直す。$D = n$ であれば、(4)項へ戻り、k を減らして計算をやり直す。

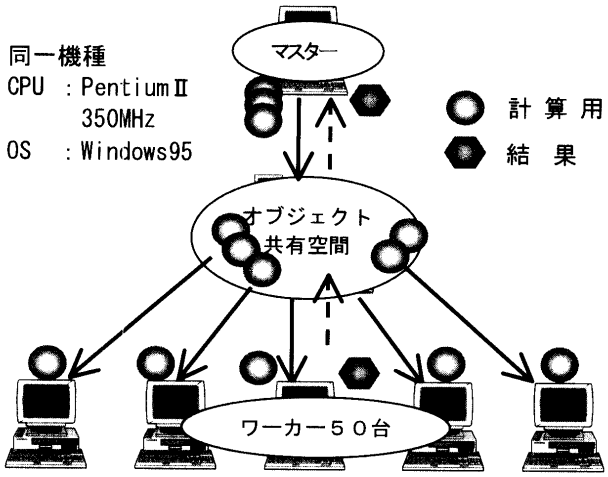
kP の計算は k を 2 進展開し、

$$k = k_0 + k_1 \cdot 2 + k_2 \cdot 2^2 + k_3 \cdot 2^3 + \cdots + k_r \cdot 2^r \quad (11)$$

と表す。ここで、k_i は 0 か 1 である。点 P の k 倍は、

$$kP = k_0 \cdot P + k_1 \cdot 2P + k_2 \cdot 2^2 P + \cdots + k_r \cdot 2^r P \quad (12)$$

となるため、$2P$ の計算を 2 倍公式で行う。

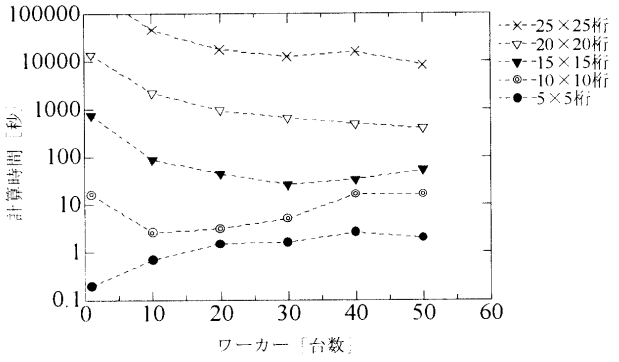


図－2 オブジェクト共有空間のシステム構成

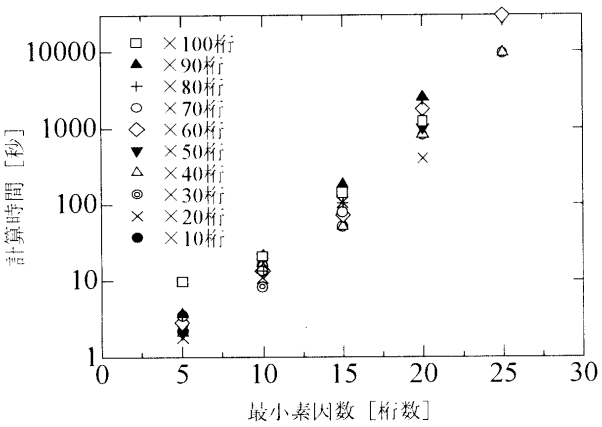
3. 分散並列処理システムによる素因数分解の実験

3.1 オブジェクト共有空間を用いた分散並列処理による素因数分解

同じ計算機(PentiumII 350MHz, Windows95)を通信速度 100Mbps のネットワークに接続し、オブジェクト共有空間による分散並列処理システムで素因数分解の実験を行う。実験システムは図－2 に示すように、オブジェクト共有空間を実装するサーバ 1 台と素因数分解の計算を行うワーカーを最大 50 台、素因数分解に必要なオブジェクトをオブジェクト共有空間に書き込むマスター1 台で構成する。オブジェクト共有空間は Jini1.0.1 の JavaSpaces サービスを使用し、ワーカー及びマスターのプログラムは Java の JDK1.3 で開発した。マスターが計算に必



図－3 オブジェクト共有空間での台数効果



図－4 ワーカー50台による高速化

要なオブジェクト（計算用Entry）を乱数よりまとめて生成し、オブジェクト共有空間に書き込む。計算用 Entry がオブジェクト共有空間に書き込まれると、ワーカーがそのオブジェクトを取り込み、素因数分解を試みる。素因数を見つけたワーカーは、その結果をオブジェクト共有空間にオブジェクト（結果Entry）として書き込む。その結果Entryをマスターが取り込んで、素因数分解が終了する。

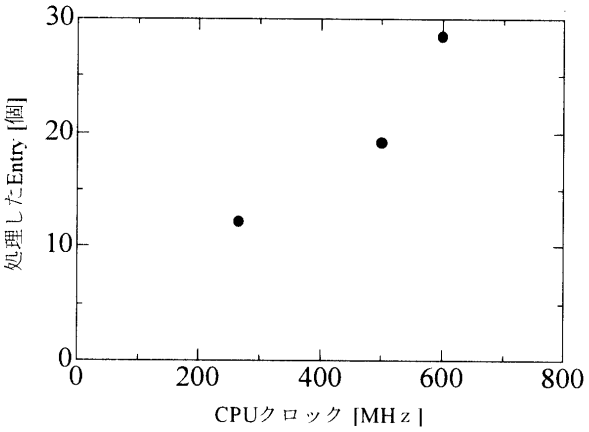
ワーカーの台数を 10 台、20 台と 10 台ずつ増やし、5×5 桁、10×10 桁、・・・と異なる合成数でそれぞれ10回の素因数分解を行って、平均を取った。実験結果を図－3 に示す。ワーカー1 台の単独計算と比較すると、5×5 桁は平均 2 個目の計算用 Entry で素因数が見つかっているため、2 台のワーカーで十分であり、それ以上ワーカーを増やしても効果は現れない。また、10×10 桁は平均 14 個目で素因数が見つかっていることから、10 台から 20 台当たりがもっとも台数効果が期待できる。15×15 桁は平均 99 番目で素因数が求まっており、30 台または 40 台ではそれぞれ3個程度のEntryを計算すればよいことになる。ワーカー30 台のとき、約 29 倍の高速化が見られ、ワーカーの台数効果がもっとも現れている。20×20 桁と 25×25 桁の素因数分解では、それぞれ平均 330 番目、平均 6050 番目と計算する Entry の数が圧倒的に多くなるため、ワーカーがもっとも多い 50 台で最速となっている。ワーカー1 台の計算時間と比較すると、20×20 桁で約 36 倍、25×25 桁で約 40 倍の高速化が見られる。さらに大きな素因数を見つける場合、より多くの Entry が計算されるため、ワーカーを 50 台よりも増やした方が高速化を期待できる。

次に、同じ環境でワーカーを 50 台に固定して、大きな合成数で素因数分解を行い、合成数の大きさが計算時間に及ぼす影響を検証した。5×10 桁、5×20 桁、・・・、5×100 桁と片方の素因数を変え、素因数分解を行った。実験はそれぞれ 10 回行い、平均した結果が図－4 である。最小素因数が大きくなると、計算時間が大きく影響を受ける。計算時間は、合成数 15×50 桁と 20×50 桁では約 14 倍に、合成数 20×50 桁と 25×50 桁を比較すると約 28 倍になる。合成数 20×50 桁と 20×60 桁では約 2 倍の計算時間となり、最小素因数の大きさが計算時間に大きく影響している。

楕円曲線法による素因数分解は、計算時間が最小素因数の大きさに強く影響されることが知られており、Brent は素因数分解の予測時間 T を次式で表現している[5]。

$$T = O(\exp(\sqrt{c \cdot \ln p \cdot \ln \ln p}) \cdot (\ln N)^2) \qquad (13)$$

ただし、合成数を N、合成数 N の素因数を p、$c \cong 2$ とし、$O(t)$ は桁数 t の 2 つの数値を加算するための作業量を表す。Brent の式は、計算時間は素因数の大きさが強く影響し、合成数の大きさは若干の影響を及ぼすことを表しており、オブジェクト共有空間による分散並列処理システムでも同様な傾向が見られる。



図－5 異機種による処理能力の違い

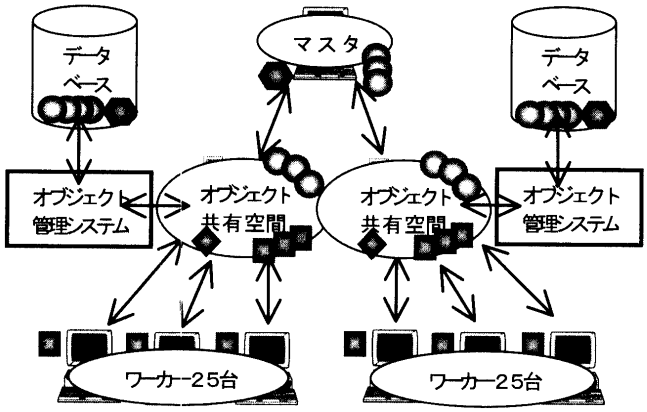


図--6 ORDB を組み込んだシステム

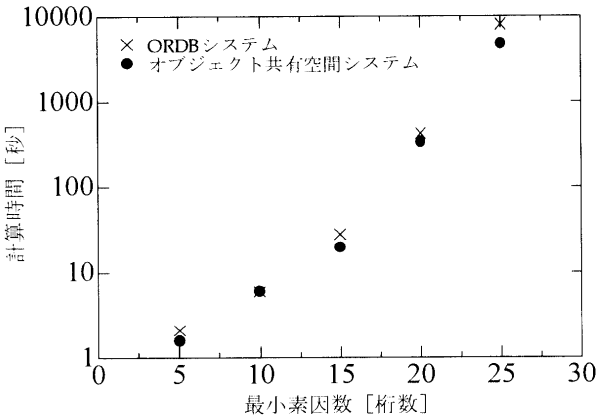
3.2 異機種分散並列処理による素因数分解

ワーカーに異なる機種 3 台を用意し、それぞれ Windows98(Pentium Ⅲ 600MHz), Linux(Celeron 500MHz), MacOSX(G3 266MHz)で Java(JDK1.3)を使って実験を行った。Java はインタープリタ方式であり、クラスファイルはどの環境でも動作するため、Windows98 でコンパイルしたクラスファイルを別の計算機へ転送して実験を行った。合成数 15×15 桁を素因数分解し、10 回測定して平均を取った。素因数が見つかるまでに各ワーカーで処理した Entry の数と CPU のクロック周波数の関係を図－5 に示す。CPU の種類と Java 仮想マシンが異なるため、単純に CPU のクロック周波数だけで処理能力は判断できないが、最速周波数である Windows98 が同じ時間内にもっとも多くの Entry を処理しており、処理能力はクロック速度に準じる結果となった。

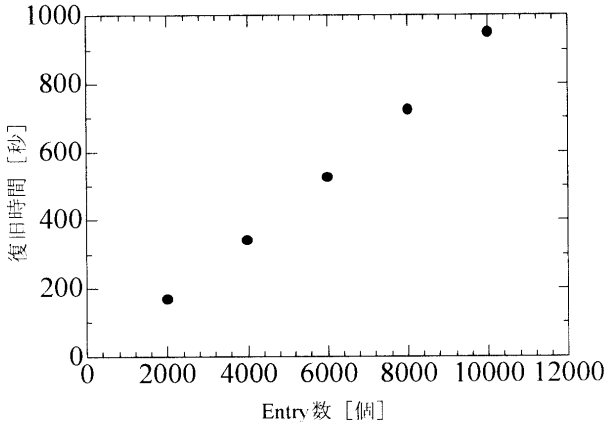
3.3 ORDB によるオブジェクトの持続

オブジェクト共有空間に貯蔵されているオブジェクトを保証するため、サーバに ORDB を実装し、オブジェクトをデータベースに格納する。ORDB を組み込んだシステム構成を図－6 に示す。ORDB に Informix 社の製品 Cloudscape3.5 を使用し、ORDB によるオブジェクト共有空間サーバの負荷が増大するため、サーバを 2 台構成とした。ワーカー50 台は各サーバに 25 台ずつ割り当て、サーバの負荷が均等になるようにした。マスターは素因数分解で使用するデータベース登録用 Entry をオブジェクト共有空間に書き込み、素因数が見つかったときにワーカーが書き込む結果 Entry を待つ。マスターはデータベース登録用 Entry をオブジェクト共有空間に 1 個ずつ交互に書き込み、オブジェクトが同数になるようにした。オブジェクト管理システムはこの登録用 Entry が書き込まれると、取り込んでデータベースに登録し、計算用 Entry をオブジェクト共有空間に書き込む。この計算用 Entry をワーカーが取り込んで、素因数分解を試みる。素因数が見つかった場合、ワーカーは結果登録用 Entry をオブジェクト共有空間に書き込む。結果登録用 Entry をオブジェクト管理システムが取り込んでデータベースの更新を行い、マスターに渡す結果 Entry をオブジェクト共有空間に書き込む。マスターが結果 Entry を取り込んで、素因数分解が終了することになる。

実験は 5×5 桁、10×10 桁、15×15 桁、20×20 桁、25×25 桁の合成数でそれぞれ 10 回行い、平均した結果をオブジェクト共有空間システムの結果とともに図－7 に示す。小さい素因数を見つける場合、計算する Entry の数が少ないため、オブジェクト共有空間システムと ORDB システムの計算時間に大きな差は現れない。素因数が大きくなるにつれて、計算に費やされる Entry の数が膨大になり、データベースの処理に要する時間が計算時間に影響を与えるようになる。25×25 桁の素因数分解では、ORDB システムの方が約 1.7 倍の計算時間となった。さら



図－7 オブジェクト共有空間と ORDB の実験結果



図－8 オブジェクトの復旧時間

に大きい素因数を求める場合、計算される Entry が増えるため、計算時間が長くなることが予想される。

次に、ORDB のデータベースからオブジェクト共有空間へオブジェクトを戻す実験を行った。ワーカーを停止した状態で、マスターによりオブジェクト共有空間へ複数の Entry を書き込み、オブジェクトがデータベースに登録された後、サーバを強制的に停止させた。サーバを再起動し、データベースへ登録されている計算用 Entry をオブジェクト共有空間へ戻す実験を行った。データベースへ登録される Entry の数を 2000, 4000, 6000, 8000, 10000 と 2000 個ずつ増やし、オブジェクト共有空間へ戻す Entry の数が復旧に及ぼす影響を検証した。実験は 10 回行い、平均した結果が図－8 である。データベースからの Entry の取り出しは、記録順に実行されるため、復旧時間は Entry の数にほぼ比例する。Entry1 個当たりの復旧時間は 100 ミリ秒以内と高速であり、多くの Entry を復旧しても高速に処理できた。

4. まとめ

分散並列処理システムを素因数分解へ応用し、ワーカー50 台による合成数 25×25 桁の計算では約 40 倍の高速化が確認できた。さらに大きな合成数の素因数分解では計算量が増えるため、ワーカーの台数を追加することで高速化が可能である。また、Java の場合、ワーカーに使用できる計算機もいろいろな機種があり、選択肢が広い。異機種による分散並列処理システムでは、ほぼ CPU のクロック速度に見合った処理がワーカーで行われ、インターネット環境を利用することでワーカーの数をかなり増やすことが可能となる。ORDB システムで確認できたように、計算速度への影響は避けられないが、長時間の連続計算ではオブジェクトの持続性を確保することは重要である。計算時間への影響を最小限に抑え、オブジェクトの復旧も迅速に対応できるようなシステムの構築が大切である。

参考文献

1) 中山 茂、Java 分散オブジェクト入門、技術堂出版(2000)
2) 由井薗 隆也、中山 茂、オブジェクト共有空間 JavaSpaces における分散並列処理の実験評価、電子情報通信学会技術報告、vol. 99、no. 547、SS99-59、pp. 73-80(2000)
3) Sun Microsystems, JavaSpaces Service Specification, Sun Microsystems(2000)
4) R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryposystems, Communications of the ACM, vol.21, pp.120-126(1978)
5) R. P. Brent, Lecture 4 Uses of Randomness in Computation, Six Lectures on Algorithms, Oxford, May-June(1999)