C-11

並列ベクトルプロセッサとしてのグラフィクスプロセッサ

Graphics Processor as Parallel Vector Processors

森 眞一郎†, 五島正裕†, 中島康彦‡, 富田眞治†
Shin-ichiro Mori, Masahiro Goshima, Yasuhiko Nakashima, Shinji Tomita

１．はじめに

コンピュータゲームに代表されるエンターテイメント系アプリケーションからの需要に支えられ, 汎用グラフィクスカードに登載されるグラフィクスプロセッサ(Graphics Processing Unit, 以下 GPU と呼ぶ.)の性能は著しい向上を続けている. さらに, フォトリアリズムを追究する CG 系アプリケーションの要求に答えるため, 近年の GPU は浮動小数点演算を正式にサポートするとともに, 従来固定であったグラフィクス演算処理フローのプログラム制御までも可能にした[25,26,27]. このような GPU のハードウェア面での高性能化と汎用化, さらには, Cg 言語\cite{nv1,nv2}に代表される GPU 向けプログラミング環境の整備は, GPU 上でのアプリケーションとして従来のグラフィクスプログラムに留まらない汎用数値計算処理にも道を開き始めている.

本稿では, GPU 上にいかに数値計算をマッピングするかを調査し GPU をベクトルプロセッサと見做したときに, どの程度の潜在能力をもっているかを考察する.

2．グラフィクスプロセッサユニット(GPU)

2.1 基本アーキテクチャ

図１に, レンダリング処理の流れを, GPU 内の主なハードウェア資源に対応づけた概念図を示す. 視点等の初期パラメータを設定した後, CPU から描画すべき物体(例えば三角形ポリゴン)のデータを GPU に送ると, GPU ではまず頂点座標の幾何変換を行ない, 物体がスクリーンに投影された時の位置および形状を求める. 次に, 投影物体をラスタライズ処理により, スクリーンの個々のスキャンラインに対応するフラグメント(投影物体をスキャンラインで切った時の断片)を計算する. 個々のフラグメントに対して, テクスチャの張り付けやフィルタ処理, シェディング処理を行なった後, 既に描画済のデータと合成し, フレームバッファに格納する. この処理の前半部分は主に頂点データに対する処理であり, 図１の頂点プロセッサ(Vertex Processor)に相当する部分で処理が行なわれる. 一方, 後半部分はピクセルデータに対する処理であり, 図１のフラグメントプロセッサ(Fragment Processor)が担当する. それぞれのプロセッサが処理の対象とするデータは, 頂点座標(x,y,z,w)およびピクセルデータ(色情報:RGBα)であり, 処理の形態も異なるため, 両プロセッサは各々に特徴的な性質をもったハードウェア構成となる.

例えば, 前者は 4×4 の座標変換行列と 4 次元座標ベクトルとの積の計算を高速に行なう少々複雑な専用ハードウェアをもつとともに分岐処理をサポートするのに対して, 後者では分岐処理はサポートせず, 2 枚の画像の色情報をピクセル単位で合成するための比較的単純なハードウェアを多数備えるという具合いである.

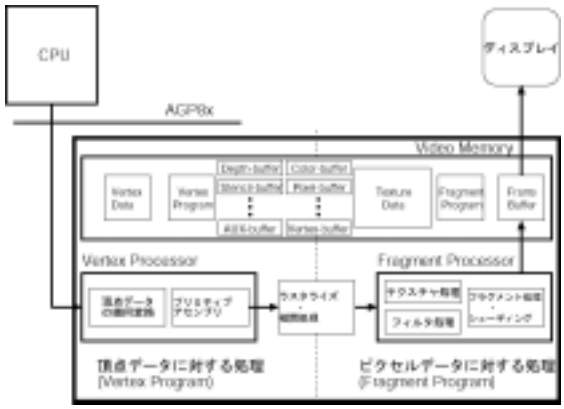

図１．レンダリング処理の概念図

また, 両プロセッサとも, 高い描画性能を達成するために複数の演算パイプラインをもっており, 入力データ列に対して SIMD 的な動作を行なう. なお, 個々のパイプライン内でも頂点データ, ピクセルデータ共に４次元のベクトルとして取り扱われる. 例えば, NVIDIA 社の GeForce6800Ultra は, RGBa それぞれの要素に対して 32bit 浮動小数点演算を行うフラグメントプロセッサを 16 台実装している.

旧来の GPU では, これらのレンダリングパイプラインの構成は固定であり, レンダリング処理も固定的であったのに対して, 近年, 複雑なレンダリング処理を実現するという要求に答えるため, 頂点プロセッサとフラグメントプロセッサにおいて, レンダリング処理の流れを動的に制御するプログラマブルパイプラインの概念が導入された. さらに, より精度の高い計算を実現するため, 扱えるデータ形式として, 16bit あるいは 32bit の浮動小数点形式が追加された.

表 1 はグラフィクスプロセッサの Chip レベルの仕様を Pentium4 と比較したものである. 昨年リリースされた GeForceFX5900Ultra は, Prescott コアの Pentium4 とほぼ同じトランジスタ数であり, ピクセル計算時の理論最大性能は Puntium4 を上回るものである. また, 今年 5 月に発表された最新の GeForce6 では, トランジスタ数が Pentium4 の 2 倍弱になるとともに, 理論最大性能も GeForceFX5900Ultra の約 2 倍弱に達している. このような GPU の高い演算性能を,

表１　Chip 諸元比較

	Pentium4	GeForce FX5900Ultra	GeForce6 6800Ultra
Core Name	Prescott	NV35	NV40
Core Clock	3.6GHz	450MHz	400MHz
Process	90nm low-k	0.13μm 銅	0.13μm 銅
Transistor	125M	135M	222M
Cache Size	1MB	非公開	非公開(+外部 cache)
Die	112mm^2	???mm^2	300mm^2

†京都大学大学院情報学研究科、 Kyoto University
‡ 京都大学大学院経済学研究科/JST、 Kyoto University/JST

PC 内の第２のプロセッサとして利用しようという試みが, GPU 上での汎用計算(GPGPU)[1]である.

2.2 基本演算

前節でも述べた通り, GPU はグラフィクス処理に特化した専用のハードウェアが多く組み込まれており, これらのハードウェアをいかに有効に活用できるかで, GPU 上でのプログラムの実行効率が大幅に異なってくる.

以下では, 汎用計算に応用可能な代表的な機能について簡単に説明する.

(1) 内積計算

２つの 4 次元ベクトル間の内積計算は, グラフィクス処理で非常に多用される機能である. 例えば, 物体からの反射光の強度計算や距離を求める場合の自乗和計算で使用する.

(2) 座標変換

上記内積計算を 4 つ組合せた計算で, 行列(4x4)とベクトル(要素数 4)の積である. 頂点プロセッサに入力される座標データは基本的に全てが座標変換の対象となる.

(3) 正規化(Reciprocal of Square Root)

ベクトルの正規化等で必要な２乗根の逆数を計算する回路である. 内積計算と組合せることで, 距離や距離の逆数を計算することが可能である.

(4) テクスチャマッピング

ビデオメモリ内に格納されたテクスチャデータを, 幾何変換後の物体に張り付けて模様を表現する機能である. テクスチャデータとして 2 次元配列を与え, 同じ大きさの長方形ポリゴンにマッピングすれば, ２つの 2 次元配列に対する要素毎の演算が可能となる.

(5) フィルタ処理

2 次元画像のエッジ強調やノイズ除去に代表されるような(主に２次元の)空間微分や積分を行なう機能である.

(6) マスク付き演算

フラグメント処理においては, 個々のピクセルの{Z, Stencil, α}を比較することで, 当該ピクセルに対する処理を無効化することが可能である. フラグメント処理パイプラインでは分岐をサポートしないが, 予めマスクパターンが分る場合には, これらのテスト機能を使うことで, ベクトル計算機のマスク付き演算に相当する機能を実現可能である.

(7) 間接参照

ごく最近サポートされるようになった機能で, テクスチャデータを別のテクスチャデータを参照する際の間接アドレスのテーブルとして使用する機能である.

一方で, GPU であるが故の実装上の制約が多く存在する. そのうちでも重要なものが以下の３つである.

(A) 条件分岐

前述の通り, Fragment Processor はその性格上, 条件分岐をサポートしていない. これは, 汎用プログラムの実装においては極めて強い制約となる.

(B) ReadBack 性能

CPU から GPU へのデータの受け渡しは, AGP8X の場合 2GB/s に近い転送速度が得られるが, その逆方向の GPU から CPU へのデータ転送では, 数 100MB/s 程度の転送速度しか得られない. また, GPU のパイプラインを一旦フラッシュした後でなければ, GPU から CPU へのデータ転送が行なえないという制約がある. したがって, GPU から CPU へのデータ転送が頻繁に行なわれると, AGP バスのオーバヘッドと GPU のパイプラインフラッシュの為, GPU の性能を十分に発揮することができない.

(C) コードサイズの制限

Vertex Processor および Fragment Processor のプログラムは, 実行に先立ち GPU のビデオメモリに格納しておく必要がある. この際, 各々のプロセッサに対して個々にプログラムサイズの上限が定められている. NV35 の場合, Vertex Program の命令数の上限は 256, Fragment Program に関しては 1024 である. さらに, Vertex Processor ではプログラム内でループを構成できるが, ループの反復回数は 256 回が上限である. このように, プログラムのコードサイズ, ならびに, 動的実行命令数に上限が存在するため, 大きなプログラムを実行するためには, 複数回のレンダリング処理に分割して実行する必要がある. 一回のレンダリングが終了する毎に, CPU にパラメータ渡し等が必要になると上記(B)の影響もうけてしまう.

3 GPU 上での数値計算

Fragment Processor が浮動小数点演算をサポートする以前には, Vertex Processor をつかった汎用計算が試みられていたが[2], Fragment Processor が浮動小数点演算をサポートすると GPGPU の対象は Fragment Processor でのテクスチャマッピングベースの汎用計算に移ってきた.

図 2 はテクスチャマッピングを用いた基本演算の例である. 図 2 (a)は N×N の 2 次元配列 X[N][N] と Y[N][N] に対して, 各要素毎の演算を行なって Z[N][N] を計算する例で, Fragment Processor のパイプライン性能を最も発揮できる演算パターンである. 前述の通り, Fragment Processor では１ピクセルあたり RGBA の 4 つの要素を同時に計算することができるため, N×(N/4) サイズのテクスチャに N×N の 2 次元配列を埋め込むことで, 最大 4 倍の性能を得ることができる.

図 2 (b)もほぼ同様であるが, 配列 Y の添字が, X,Z の添字とアフィン関係にある場合の例である. この例では, 配列 Y をアフィン関係の式に基づき座標変換したのち, 演算に必要なデータのみがサンプリングされ, X との演算に使用さ

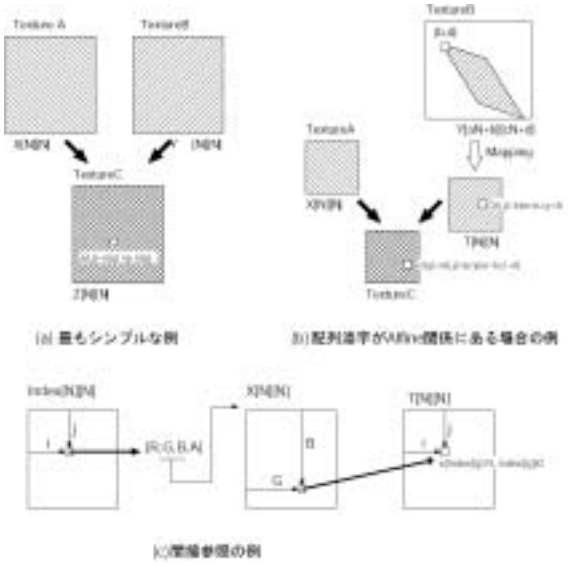

図２ Texture Mapping を用いた基本演算

れる. 図では説明の都合上, T[N][N]というテクスチャを導入しているが, 実際には T[N][N]は作成せずに, サンプリングと同時に対応する X の要素との間で直接演算が行なわれる.

図 2 (c)は配列添字にインデックス配列が使われた場合の例である. この場合は, Dependent Texture を使った間接参照が行なわれる. この例でも便宜上 T[N][N]を導入しているが実際に作成されるわけではない.

図 3 は, 配列要素の総和等に見られる Reduction 系演算の実装例である. この例では, 配列 X を N/2 だけ並行移動した(仮想的な)配列 T を作り, 配列 X と配列 T の左半分に対して, Texture Mapping を適用し新たな配列 U と V を作る. 次に U と V を Texture Mapping することで, 元の配列の 1/4 のサイズに縮退した配列 W をつくる. 以下 W に対して同様のステップを繰り返すことで Reduction 演算を実現する. 一回 Texture Mapping を行なう毎に, 座標系の再設定が必要となるため, 若干のオーバヘッドが生じる.

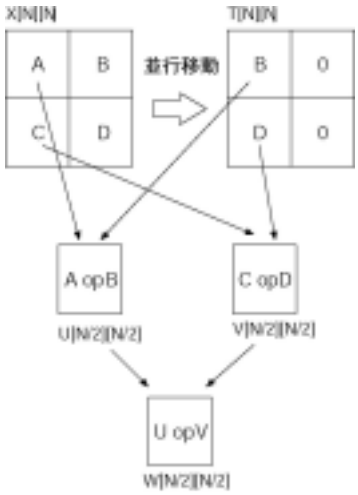

図 3 Reduction 系演算

4 GPU を用いた汎用計算の研究動向

4.1 プログラミング環境

従来の GPU プログラム開発は，ゲーム開発者等に見られるような GPU のアセンブラ言語を駆使したプログラミングか，あるいは OpenGL や Direct3D 等のグラフィクス向け高級言語をもちいたプログラミングの何れかであったが, NVIDIA の Cg(C for graphics)[3,4], さらには, Stanford 大学の Brook[5]や Waterloo 大学の Sh[6]等の高級言語が提案されたことで, GPU プログラミング環境が大幅に改善された.

Cg は，高いプログラマビリティと様々な機能をもち始めた GPU に対する，プログラム開発効率を向上する目的で GPU メーカ自らが開発したプログラミング言語で，OpenGL や Direct3D 等のグラフィクス向け高級言語では記述が困難／不可能な GPU の細部に渡る C-like なプログラミング環境を提供する.

Brook は, ANSI C をストリーム処理のための言語拡張を行なったデータ並列言語の一種である. Merrimac[7]等のストリームプロセッサ向けに開発されたものが汎用 GPU 向けに移植され公開されている.

Sh は, 条件分岐等の GPU のハードウェア制約を隠蔽した仮想マシンを想定し, その仮想マシンに対する API を C++ のクラスライブラリとして提供している.

Cg はプログラマが GPU のハードウェアを意識してプログラムすることを前提にしているのに対して，Brook や Sh はユーザに GPU のハードウェアを隠蔽するため，プログラムのどの部分が実際に GPU で実効されるのかが分りにくいという欠点がある.

4.2 基本演算

次に，GPU を用いた基本的な数値演算に関する研究について言及する. 初期のプログラマブル GPU では，Vertex Processor のみが座標計算のための浮動小数点演算機能を備えていたため，Thompson ら[2]は Vertex Processor を用いた数値計算法を紹介している. ここでは，2.2 節(2)で述べた手法を用いてスカラ値配列を 4 次元座標ベクトルにパッキングして並列度を 4 倍にするとともに，複数の Vertex Processor を SPMD 並列動作させることで，要素数が非常に大きく演算強度が高い演算に対しては CPU の 10 倍近い演算性能を出している.

最近の GPU では，Fragment Processor でも浮動小数点演算機能を備えており，かつ，通常，GPU 内では Fragment Processor 数 ＞ Vertex Processor 数であるため，GPU での数値演算は Fragment Processor を使ったものが主流である. 前述の Brook を用いて，GPU 性能を測定したデモでは GeForce5900Ultra を用いた浮動小数点乗算(A[i]=A[i]*A[i])で 20GFLOPS を出している.

Fragment Processor の Texture Mapping 機能を使って NxN の行列 A，B の行列積計算を求める例は文献[8]に紹介されている. この例では，一回の Texture Mapping で A[i,k]*B[k,j]を全ての i,j に対して実行し，この計算を異なる k に対して N 回実行した結果を足し合わせることで A＊B を求めている. さらに，この論文は GPU の固定小数点演算のみを使った演算で如何にして誤差を抑えるかを議論している. また，この論文では GPU が採用している Unified Memory Architecture が Texture Mapping を用いた数値計算のボトルネックになるであろうことを予測している.

行列積に関しては，ベクトルとベクトルの積に分解したうえでさらにそれを 2 次元の RGBA テクスチャにパッキングして計算を行なうアルゴリズム等の改良が提案されている[9,10].

4.3 応用演算

グラフィクスの世界でも，よりリアルな映像表現を得るために物理シミュレーション結果をつかった画像データが用いられている. 従来は物理シミュレーションは CPU で行ない結果のみを GPU で表示する手法が取られていたが，物理シミュレーションの一部も GPU で行なうことで対話性を向上する試みがなされている[10,11,12,13]. その代表が, 流体シミュレーションと粒子シミュレーションであり，雲の動きや水面の波の表現などで使われている.

Harris 等[15]はセルオートマトンを拡張した結合写像格子法(Coupled Map Lattice 法: CML 法)[16]を用いた物理シミュレーションの GPU への実装法を示しており, ナビエストーク方程式を解く流体シミュレーションやグレイスコットモデルを用いた反応拡散シミュレーションを行なっている. そこで使われている GPU への実装技術としては, 2.2 節(5)で述べたフィルタ機能を用いた陽解法による偏微分方程式の求解や, 事前に求めた関数表を Texture として保存しておき, 実行時には Dependent Texture として Lookup することで関数計算を省略する手法等がある. さらに, Goodnight 等は境界値が与えられた場合の熱拡散問題に対して MG 法を用いた GPU 実装[17]を示している. そこでは, Occlusion Query を用いた収束判定の高速化や, 多重解像度テクスチャ間の相互補完機能を用いた高速化が提案されている.

Kaufman 等のグループでは，格子ボルツマン法(以下, LBM 法と呼ぶ)[18]を用いた流体シミュレーションの実装

を提案している[19]. LBM 法は仮想的な粒子をある規則的な格子に沿って運動させることで流体の運動を表現する手法であり数値流体力学分野で近年注目されている手法である. さらに, Kafuman 等は複数の GPU を用いて LBM 法に基づく流体シミュレーションの並列化を行ない, GPU クラスタによる高性能計算の可能性を示している[20].

これらの他にも, Moreland 等による FFT の GPU 実装[21]や, GPU 上での疎行列の扱いを議論した[22,23]等, 多くの数値計算に関する実装の報告がなされている[9].

また, 少々変った応用としては結果の可視化を前提にしたデータベースの集合演算に GPU を応用する研究なども行なわれている[24].

5 まとめ

最新の GPU では, GPU 内の浮動小数点演算器がすべて稼働した場合の理論的な演算性能は 100GFLOPS にも達するといわれているが, 実用上は全ての Fragment Processor が同時に動作可能な状況が現実的な理論最大性能である. それでも 10〜20GFLOPS 程度と非常に高い演算性能をもっている. 一方で, メモリバンド幅は 30〜40GB/s と汎用 CPU のメモリバンド幅よりは高いものの, 10GFLOPS の演算器をフル稼働するのに十分なデータを供給できるとは言いがたい. さらに現在のビデオメモリはディスプレイ出力用のフレームバッファ領域やテクスチャデータ保存用のテクスチャ領域, その他種々の用途で共有する Unified Memory Architecture を採用しており, このアーキテクチャが GPU 上の汎用計算における演算性能の足枷になっているという報告もでている[2,24]. また, GPU であるが故のアーキテクチャ上の制約から, 任意のプログラムが効率的に実行可能であるとは言いがたい.

しかしながら, グラフィクス処理の演算パターンと類似の演算を多用するアプリケーションや計算結果の実時間可視化が必要なケースでは, GPU 上の汎用数値計算にも大きな可能性が残されていると考える.

参考文献

[1] General Purpose Computation on Graphics Processor ホームページ http://www.gpgpu.org/

[2] Thompson, C., Hahn, S. and Oskin, M. : Using Modern Graphics Architectures for General-Purpose Computing : A Framework and Analysis, Proc. of 35th International Symposium on Microarchitecture (MICRO-35) pp. 306--320 (2002).

[3] NVIDIA Corporation: Cg Toolkit User's Manual Release 1.2 (2004).

[4] NVIDIA Corporation: NVIDIA OpenGL Extension Specifications (2003).

[5]http://graphics.stanford.edu/projects/brookgpu/

[6]http://libsh.sourceforge.net/

[7]W.J. Dally, et al., ``Merimac:Supercomputing with Streams," SC2003, November 2003

[8] Larsen, E. and McAllister, K. : Fast Matrix Multiplies using Graphics Hardware, Proc. Supercomputing 2001(2001).

[9] GP2 : ACM Workshop on General Purpose Computing on Graphics Processor, (2004). http://www.cs.unc.edu/Events/Conferences/GP2/

[10] Mark Harris : "GPGPU: Beyond Graphics," Advanced OpenGL Tutorial at Game Developers Conference, (2004) http://developer.nvidia.com/object/gdc_2004_presentations.html

[11] 手山奈緒子, 安藤祥子, 村木 茂, 藤代一成 : プログラマブル GPU を用いた反応拡散系パターンダイナミクスの演算と表示の高速化, 画像電子学会第 210 回研究会講演予稿集, pp.17--20, (2004).

[12] 小野佳織, 藤代一成, 竹島由里子, 塚越誠一 : 拡張サーフェルを用いた砕波の 3 次元アニメーション : Visual Computing シンポジウム'04, pp.209--212, (2004).

[13] 金井 崇, 安井悠介 : GPU による細分割曲面の意匠形状評価,Visual Computing シンポジウム'04, pp.85--90, (2004).

[14] 近藤 亮, 金井 崇 : 簡易化四面体メッシュを用いた詳細メッシュのインタラクティブな物理法則アニメーション,Visual Computing シンポジウム'04, pp.7--12, (2004).

[15] M.J. Harris, et al.: "Physically-Based Visual Simulation on Graphics Hardware," EUROGRAPHICS Workshop on Graphics Hardware, pp.100--118, (2002).

[16] Kaneko, K.(Ed): "Theory and applications of coupled map latices," Wiley, (1993).

[17] Goodnight, N., Wooley, C., Lewin, G., Luebke, D. and Humphreys, G. : "A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware," EUROGRAPHICS Workshop on Graphics Hardware, pp.102--111, (2003).

[18] 蔦原道久, 高田尚樹, 片岡 武 : 格子気体法・格子ボルツマン法, コロナ社, (1999).

[19] Xiaoming Wei, et al., : "Simulating Fire with Texture Splats," Proc. Visualization, pp.227--234, 2002.

[20] Zhe Fan, et al., : "GPU Cluster for High Performance Computing." Supercomputing 2004(to appear).

[21] Moreland, K. and Angel, E. : "The FFT on a GPU," EUROGRAPHICS Workshop on Graphics Hardware, pp.112--119, (2003).

[22] J. Kruger and R. Westermann, :"Linear Algebra Operators for GPU Implementation of Numerical Algorithms," Proc. ACM SIGGRAPH 2003, pp.908--916,(2003).

[23] J. Bolz, et al., "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," Proc. ACM SIGGRAPH 2003, pp.917--924,(2003)

[24] SIGGRAPH2004 GPGPU Course Note.
http://www.gpgpu.org/s2004/

[25] Ertl, T., Weiskopf,D., Kraus, M., Engel,K., Weiler, M., Hopf, M., Rottger, S., and Rezk-Salama C. : "Programmable Graphics Hardware for Interactive Visualization," Eurographics2002 Tutorial Note(T4) (2002).

[26] NVIDIA Corporation: NVIDIA GEFORCE FX 5900 PRODUCT OVERVIEW (2003).

[27] NVIDIA Corporation: NVIDIA CineFX Technical Breif, TB-00626-001_v01, (2003).

[28] Woo, M. and Neider, J. and Davis, T. (株式会社アクロス訳) : OpenGL プログラミングガイド第 2 版, ピアソンエデュケーション (1997).

[29] J.A.エドワード (滝沢, 牧野訳) : OpenGL 入門,ピアソンエデュケーション (2002).

[30] 山内, 他, アクティブボリュームレンダリングに基づくシミュレーションステアリング, 信学技報 CPSY2001-35, pp.1-8, 2001 年 8 月.

[31] 篠本 雄基 : 汎用グラフィックスカードを用いた数値計算, 京都大学工学部卒業論文, (2004).

[32] 森眞一郎,他 : 汎用グラフィクスカード上での簡易シミュレーション, 電子情報通信学会信学技報 CPSY2004-24, pp.25—30(2004).