Vol.2009-HPC-121 No.15
2009/8/5

GPUを用いたアウトオブコアなコーンビーム再構成の高速化

興津佑輔[†1] 伊野文彦[†1] 岸武人[†2]
大西修平[†2] 萩原兼一[†1]

本稿では 1024^3 ボクセルを超えるような大規模なボリュームに対するコーンビーム再構成を GPU（Graphics Processing Unit）を用いて高速化する．1024^3 ボクセルを超えるような大規模なボリュームは GPU が有するビデオメモリに格納できない．したがって，提案手法ではボリュームをサブボリュームに分割し，さらに複数の GPU を用いて入力である投影像，または出力であるボリュームを分担して再構成する．投影像を分割する手法では，各 GPU が担当する投影像からボリューム全体を再構成する．一方，ボリュームを分割する手法では，各 GPU がすべての投影像から担当するサブボリュームを再構成する．評価実験では，提案手法が 1024^3 ボクセルからなるボリュームを再構成できることを確認した．また，Tesla S1070-400 を用いた実験では，1 つの GPU を用いた場合と比較して 2.6 倍の高速化を達成できた．

Accelerating Out-of-core Cone Beam Reconstruction Using GPU

Yusuke Okitsu,[†1] Fumihiko Ino,[†1] Taketo Kishi,[†2]
Syuhei Ohnishi[†2] and Kenichi Hagihara[†1]

This paper presents an acceleration method of cone beam reconstruction for large-scale volume using graphics processing unit (GPU). The proposed method reconstructs subvolumes using multi-GPU environments because such a large-scale volume exceeds the amount of video memory. With respect to the parallelization scheme for multi-GPU environments, there can be two parallelization approaches, projection parallelism or voxel parallelism. Projection parallelism scheme reconstructs the entire volume from a subset of projections on each GPU. On the other hand, voxel parallelism scheme reconstructs subvolume from all projections on each GPU. Experimental results demonstrate that our method can perform out-of-core reconstruction for 1024^3-voxel volume. In addition, the performance of NVIDIA Tesla S1070-400 is 2.6 times faster than that of single GPU.

1. はじめに

コーンビーム再構成はコーンビーム CT（Computed Tomography）で撮影した数百枚の投影像から撮影対象をボリュームとして生成する技術である．コーンビーム CT とは CT 撮影法の一種であり，X 線源と平面状の検出器が撮影対象の周囲を回転しながら撮影する．この技術は工業製品の非破壊検査に用いられており，工場の製造ラインへ検査工程を組み込むために実時間での再構成が要求される．しかし，生成するボリュームが大きく（1024^3 ボクセル以上），CPU を用いた実装では再構成に数時間を要する．ゆえに，アクセラレータによる高速化が研究されている．

アクセラレータとして，近年 GPU（Graphics Processing Unit）が注目されている．GPU はグラフィクス処理の高速化を目的に設計された専用ハードウェアであるが，近年では汎用計算に GPU を用いる GPGPU[1)]（General Purpose Computation on GPUs）が盛んである．例えば，nVIDIA 社が提供する総合開発環境 CUDA[2)]（Compute Unified Device Architecture）は C 言語を拡張した言語で汎用計算を実装できる．

GPU を用いてコーンビーム再構成を高速化した研究として文献 3)–5) があげられる．これらの文献では，コーンビーム再構成の実装として一般的な FDK（Feldkamp，Davis，Kress）法[6)] を実装している．興津ら[3)] は FDK 法の性能ボトルネックである逆投影処理に対し，GPU が有するビデオメモリ（VRAM）への参照回数を削減し高速化している．また，Sherl ら[4)] はプログラムの命令数およびレジスタの使用量を削減し，より多くのスレッドブロック[2)]（TB）をマルチプロセッサ[2)] へ割り当てることで逆投影処理を高速化している．これら 2 つの研究では，逆投影処理で必要な線形補間を，ハードウェアにより演算できるテクスチャメモリ[2)] を用いて実現している．一方，Noel ら[5)] は共有メモリ[2)] を用いて線形補間を実現している．

†1 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
†2 島津製作所分析計測事業部 NDI ビジネスユニット
Non-Destructive Inspection Business Unit, Analytical and Measuring Instruments Division, Simadzu corporation

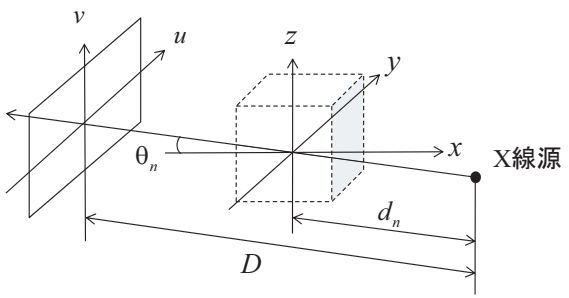


図 1 FDK 法における座標系

従来研究の中では興津らの手法が最も高速である．興津らの手法では，VRAM に格納できる 512^3 ボクセルまでのボリュームを実時間で再構成できる．一方で，非破壊検査で用いられる，1024^3 ボクセルを超えるような大規模なボリュームは VRAM の容量を超えるため再構成できない．なお，本稿では VRAM の容量を超えるような大規模なボリュームに対するコーンビーム再構成をアウトオブコアな再構成と呼ぶ．

提案手法ではアウトオブコアな再構成を実現するために，ボリュームをサブボリュームに分割する．その上で，複数の GPU を用いてすべてのサブボリュームを再構成する．ここで，複数の GPU を用いて並列化する手法はデータを分割する方法により 2 つに分類できる．具体的には入力である投影像を分割する投影像分割方式，および出力であるボリュームを分割するボリューム分割方式である．投影像分割方式は，各 GPU が担当する投影像からすべてのサブボリュームを再構成する．一方，ボリューム分割方式は，各 GPU がすべての投影像から担当するサブボリュームを再構成する．

以降では，まず 2 章で FDK 法について述べる．その後，3 章で 1 つの GPU を用いた基本実装について説明する．4 章では提案手法の設計および理論性能について述べ，5 章では評価実験の結果より実効性能が理論性能に従うことを示す．更に，提案手法の拡張性を考察する．最後に 6 章で本稿をまとめる．

2. Feldkamp, Davis, Kress 法

FDK 法[6] は，コーンビーム CT から得た $U \times V$ 画素からなる K 枚の投影像 $P_1, \ldots, P_K$ を入力として，N^3 ボクセルからなるボリューム F を出力する．このアルゴリズムは，フィルタリング処理および逆投影処理により構成される．**図 1** に FDK 法における座標系を示す．図 1 で，uv 座標系にある四角形は投影像 P_n（$1 \leq n \leq K$）を表し，xyz 座標系にある立方体はボリューム F を表す．D および d_n はそれぞれ X 線源から投影像またはボリュームの中心までの距離であり，θ_n は投影像の中心と X 線源を結んだ直線に対し xyz 座標系の x 軸がなす角度である．

まずフィルタリング処理について説明する．フィルタリング処理では，すべての投影像 $P_1, P_2, \ldots, P_K$ に Shepp-Logan フィルタ[7] を適用し，フィルタ済み投影像 $Q_1, Q_2, \ldots, Q_K$ を得る．このフィルタにより，逆投影処理で発生するノイズを低減できる．あるフィルタ済み投影像 Q_n の画素値 $Q_n(u,v)$ （$0 \leq u \leq U-1, 0 \leq v \leq V-1$）は，投影像 P_n の画素値 $P_n(u,v)$，フィルタサイズ R，および重み関数 $W_1(r,v)$ により式 (1) で与えられる．また，重み関数 $W_1(r,v)$ は式 (2) で与えられる．

$$Q_n(u,v) = \sum_{r=-R}^{R} \frac{2}{\pi^2(1-4r^2)} W_1(r,v) P_n(r,v) \tag{1}$$

$$W_1(r,v) = \frac{D}{\sqrt{D^2+r^2+v^2}} \tag{2}$$

次に逆投影処理について説明する．ボリューム F の各ボクセル $F(x,y,z)$ （$0 \leq x,y,z \leq N-1$）はフィルタ済み投影像 $Q_1, Q_2, \ldots, Q_K$ により式 (3) で与えられる．

$$F(x,y,z) = \frac{1}{2\pi K} \sum_{n=1}^{K} W_2(x,y,n) Q_n(u(x,y,n), v(x,y,z,n)) \tag{3}$$

式 (3) での $Q_n(u,v)$ は，投影像 Q_n の座標 (u,v) における画素値を，$W_2(x,y,n)$ は重みを表す．重み $W_2(x,y,n)$ および座標 $(u(x,y,n), v(x,y,z,n))$ は式 (4)〜(6) で与えられる．

$$W_2(x,y,n) = \left(\frac{d_n}{d_n - x\cos\theta_n + y\sin\theta_n} \right)^2 \tag{4}$$

$$u(x,y,n) = \frac{D(x\sin\theta_n + y\cos\theta_n)}{d_n - x\cos\theta_n + y\sin\theta_n} \tag{5}$$

$$v(x,y,z,n) = \frac{Dz}{d_n - x\cos\theta_n + y\sin\theta_n} \tag{6}$$

また，$u(x,y,n)$ および $v(x,y,z,n)$ は実数である．したがって，画素値を線形補間する．

3. 基本実装

本章では，1 つの GPU を用いて $N \geq 1024$ のボリューム F を再構成する基本実装を説明する．$N \geq 1024$ の場合，ボリューム F のデータサイズは 4GB 以上である．現時点の VRAM 容量は最大で 4GB であること，および VRAM には入力である投影像も格納することを踏まえると，ボリューム全体を VRAM に格納できない．ゆえに，アウトオブコアな再構成になる．ここで式 (3) に着目すると，各ボクセル $F(x, y, z)$ は独立に計算できる．また，式 (1) および式 (3) より，フィルタリング処理は逆投影処理に対して独立である．ゆえに，ボリューム F を N^3/B（$B > 1$）ボクセルからなる B 個のサブボリューム F_b（$1 \leq b \leq B$）に分割できる．各サブボリュームは，b の値により x，y および z 座標の範囲がそれぞれ異なるため，逆投影処理へ b を引数として与え各座標の範囲を指定する．この範囲拡大により，各サブボリュームは N^3/B ボクセルからなるボリュームに対する再構成と同様に処理できる．そこで，VRAM にサブボリューム F_b を格納できるように B を選択し，各サブボリュームを従来手法[3] を用いて再構成する．

次に，ボリュームをサブボリュームへ分割する場合の形状について説明する．サブボリュームの処理には従来手法を適用するため，従来手法の特徴から分割方法を決定する．従来手法では，式 (6) を式 (7) および式 (8) で与えられる式に変形する．その理由は，ボクセルの z 座標のみが異なる N 個のボクセルに対する計算で，式 (8) の計算結果を再利用できるためである．

$$v(x, y, z, n) = v'(x, y, n)z \tag{7}$$

$$v'(x, y, n) = \frac{D}{d_n - x\cos\theta_n + y\sin\theta_n} \tag{8}$$

したがって，z 軸方向にボリュームを分割すると式 (8) を再利用できる回数が減り演算回数が増加する．一方で，x 軸方向または y 軸方向にボリュームを分割しても演算回数および VRAM の参照回数は変化しない．ゆえに，x 軸方向または y 軸方向にボリュームを分割する．更に，本実装では x 軸方向に連続するボクセルを VRAM 上で連続する領域に格納する．x 軸方向にボリュームを分割する場合，B に依存して Coalesced 参照[2] の条件を満たせない可能性がある．以上の理由により，y 軸方向にボリュームを分割する．

最後にアウトオブコアな再構成の疑似コードを**図 2** に示す．以下では，従来手法と異なる点を説明する．提案手法では，サブボリューム F_1 を処理する過程で生成したフィルタ済み投影像 Q_n をメインメモリへ転送する（9 行目）．なぜならば，フィルタ済み投影像 Q_n はサブボリュームに対して独立であり，残りのサブボリュームに対する処理で繰り返し使用できるためである．さらに，実装時には CUDA が提供するストリーム[2] を用いて CPU・GPU 間のデータ転送時間を隠蔽する．

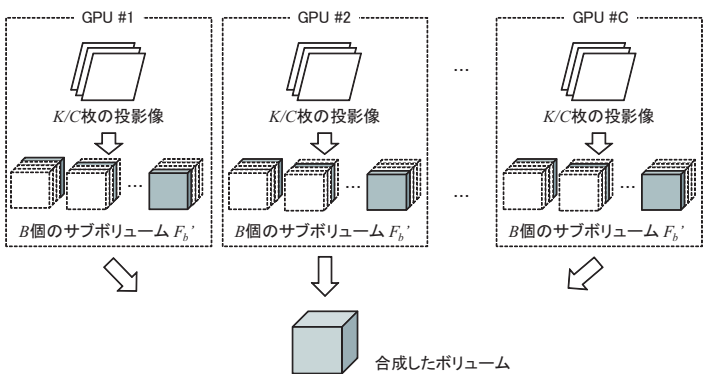
```
Input: Projection P_1 ... P_K, filtering size R, number of subvolume B, and
       parameters D, d_1 ... d_K, θ_1 ... θ_K
Output: Volume F
Algorithm Reconstruction()
 1: Divide volume F to subvolume F_1, F_2, ..., F_B;
 2: for b = 1 to B do
 3:    Initialize subvolume F_b;
 4:    for k = 1 to K do
 5:       if b == 1 do
 6:          Transfer projection P_k to global memory;
 7:          Q_k ← Filtering(P_k, R);
 8:          Transfer filtered projection Q_k to texture memory;
 9:          Transfer filtered projection Q_k to main memory;
10:       else
11:          Transfer filtered projection Q_k to texture memory;
12:       end if
13:       Bind filtered projection Q_k as a texture;
14:       F_b ← Backprojection(Q_k, D, d_k, θ_k, k, b);
15:    end for
16:    Transfer subvolume F_b to main memory;
17: end for
```

図 2 アウトオブコアな再構成を実現する疑似コード

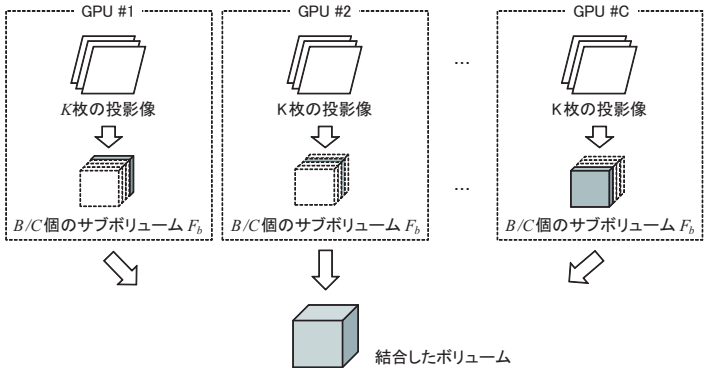
4. 提案手法

3 章で説明した基本実装を拡張し，C 個の GPU を用いたアウトオブコアな再構成について説明する．まず，C 個の GPU へ再構成処理を分割する方法について説明する．1 章で述べたように，並列化する手法は投影像分割方式およびボリューム分割方式の 2 つに分類できる．投影像分割方式では，式 (1)〜(2) に着目する．これらの式より，フィルタリング処理は異なる投影像間で依存がない．その結果，図 2 の疑似コードで投影像に関するループを並列化できる（4 行目）．一方，ボリューム分割方式は 3 章で述べたように異なるサブボリューム間では依存がない点に着目する．サブボリューム間で依存がないため，図 2 の疑似コードでサブボリュームに関するループを並列化できる（2 行目）．

次に，提案手法の概要を**図 3** に示す．図 3(a) で示すように，投影像分割方式は各 GPU



(a) 投影像分割方式



(b) ボリューム分割方式

図 3 複数の GPU を用いて再構成を並列化する手法の概要.

が K/C 枚の投影像から B 個のサブボリューム F_b' を再構成する．ここで式 (3) より，各ボクセル $F(x,y,z)$ はすべての投影像から算出する必要がある．ゆえに，各 GPU で再構成したサブボリューム F_b' を合成し，サブボリューム F_b を生成する必要がある．その後，サブボリュームを結合する．一方，図 3(b) で示すように，ボリューム分割方式は各 GPU が K 枚の投影像から B/C 個のサブボリューム F_b を再構成する．この方式では，すべての投影像から各ボクセル $F(x,y,z)$ を算出するため式 (3) を満たす．したがって，再構成したサブボリューム F_b を結合するだけで良い．ここで，$B < C$ の場合は $C - B$ 個の GPU が遊休状態になる．ゆえに，$B \geq C$ を満たす必要がある．なお，図 3(b) は $B = C$ の場合を表す．

表 1 複数の GPU を用いた場合における時間計算量およびメモリ使用量

		投影像分割方式	ボリューム分割方式
時間計算量	ダウンロード	$O(BKUV/C)$	$O(BKUV/C)$
	フィルタリング処理	$O(KUV/C)$	$O(KUV)$
	逆投影処理	$O(KN^3/C)$	$O(KN^3/C)$
	リードバック	$O(N^3)$	$O(N^3/C)$
	後処理（CPU で処理）	$O(N^3C)$	$O(C)$
メモリ使用量	VRAM	$O(N^3/B)$	$O(N^3/B)$
	メインメモリ	$O(N^3C)$	$O(N^3)$

次いで，時間計算量およびメモリ使用量について説明する．**表 1** に投影像分割方式およびボリューム分割方式に対する，各処理の時間計算量およびメモリ使用量を示す．なお，表 1 でのダウンロードとは投影像をメインメモリから VRAM へ転送する処理を意味しており，リードバックは再構成したサブボリュームを VRAM からメインメモリへ転送する処理を意味する．また，後処理は CPU で実行する処理である．

まず，ダウンロードおよびリードバックの時間計算量について説明する．ダウンロードおよびリードバックは，時間計算量が CPU・GPU 間のデータ転送量に比例する．ダウンロードでは図 2 の疑似コードで示したように，1 つのサブボリューム F_b を処理する毎にすべての投影像を転送する．投影像分割方式では 1 つの GPU が B 個のサブボリュームを処理するため，K/C 枚の投影像を B 回転送する．また，ボリューム分割方式では 1 つの GPU が B/C 個のサブボリュームを処理するため，K 枚の投影像を B/C 回転送する．また，投影像のデータサイズは $O(UV)$ である．ゆえに，時間計算量は共に $O(BKUV/C)$ である．一方，リードバックでは各サブボリューム F_b が 1 回ずつ転送される．投影像分割方式は 1 つの GPU から B 個のサブボリュームを転送する．また，ボリューム分割方式は 1 つの GPU から B/C 個のサブボリュームを転送する．サブボリューム F_b のデータサイズは $O(N^3/B)$ であるため，時間計算量はそれぞれ $O(N^3)$，$O(N^3/C)$ になる．

次いで，フィルタリング処理および逆投影処理の時間計算量について述べる．フィルタリング処理では，1 つの GPU が処理する投影像の画素数に時間計算量が比例する．投影像分割方式では 1 つの GPU が K/C 枚の投影像を処理し，ボリューム分割方式では 1 つの GPU が K 枚の投影像を処理する．1 枚の投影像は UV 画素であるため，時間計算量はそれ

ぞれ $O(KUV/C)$，$O(KUV)$ になる．また，逆投影処理の時間計算量は，1 つの GPU が処理する投影像の枚数およびボクセル数に比例する．投影像分割方式では 1 つの GPU が K/C 枚の投影像から B 個のサブボリュームを再構成する．一方，ボリューム分割方式では 1 つの GPU が K 枚の投影像から B/C 個のサブボリュームを再構成する．1 つのサブボリュームは N^3/B ボクセルからなるので，時間計算量は共に $O(KN^3/C)$ である．

また，後処理として投影像分割方式では C 個のサブボリューム F'_b を合成する．また，この処理の時間計算量は合成するボクセル数に比例する．1 つのサブボリュームは N^3/B ボクセルからなり，サブボリュームは B 個あるため，時間計算量は $O(N^3C)$ である．一方，ボリューム分割方式は後処理として B 個のサブボリュームを結合する．したがって，時間計算量は $O(B)$ である．

最後にメモリの使用量について説明する．VRAM の使用量は，どちらの手法も 1 つのサブボリュームを格納するだけである．ゆえに，VRAM の使用量は $O(N^3/B)$ になる．一方，メインメモリの使用量は大きく異なる．投影像分割方式は C 個の GPU から B 個ずつサブボリュームをリードバックする．それゆえ，メインメモリの使用量は $O(CN^3)$ である．また，ボリューム分割方式は C 個の GPU から B/C 個ずつサブボリュームをリードバックする．したがって，メインメモリの使用量は $O(N^3)$ と求まる．

5. 評価実験

本章では，提案手法の性能および拡張性について述べる．また，ストリームによる性能向上を評価する．データセットには $K = 1200, U = 1024, V = 1012, N = 1024$ である実データを用いた．また，実行時のパラメータには $R = 512, B = 8$ を採用した．実行環境は CPU として Xeon E5450（3.0GHz）を持ち，16GB のメインメモリを搭載する PC を用いた．さらに，GPU として 4 基の GPU を搭載する nVIDIA Tesla S1070-400 を接続した．OS は CentOS 5.3 であり，ドライバのバージョンは 185.18.08 である．また，CUDA のバージョンは CUDA 2.2 を用いた．

5.1 性能評価

表 2 に $C = 1, 2, 4$ における結果を示す．実験では TB のサイズを 64×8 スレッドとした．なお，メインメモリの容量不足により，$C = 4$ の場合は投影像分割方式を実行できない．

表 2 で示すように，$C = 1$ の場合との比較では $C = 2$ の場合は約 1.7 倍，$C = 4$ の場合は約 2.6 倍高速である．C に比例して性能が向上しない原因は，各手法に並列化できない処理が存在するためである．また後処理を含めると，$C = 2$ ではボリューム分割方式および投影像分割方式は同等の性能だといえる．各処理を個別に比較すると，フィルタリング処理，逆投影処理，およびボリュームのリードバックに要する時間について，C との関係は表 1 で示した時間計算量通りである．しかし，投影像のダウンロードに要する時間および C の関係は表 1 で示した通りではない．この原因は，CPU・GPU 間を接続する PCI-Express バスのオーバヘッドだと考えられる．PCI-Express バスは同時に複数の GPU へデータを転送できず，転送命令が競合すると待ち時間が発生する．

表 2 複数の GPU を用いた場合における実験結果（秒）

	$C = 1$	$C = 2$		$C = 4$
	基本実装	投影像分割方式	ボリューム分割方式	ボリューム分割方式
初期化	0.52	0.52	0.26	0.13
ダウンロード（投影像）	14.88	9.73	9.78	6.32
フィルタリング処理	9.24	4.69	9.24	9.24
逆投影処理	63.76	31.94	32.01	16.60
リードバック（投影像）	1.59	1.10	1.63	2.27
リードバック（ボリューム）	1.27	1.41	0.67	0.35
後処理	—	3.76	0.00	0.00
合計	91.26	53.15	53.59	34.91

また，提案手法ではフィルタ済み投影像をメインメモリへ転送し，繰り返し使用する．表 2 より，フィルタ済み投影像のリードバックに要する時間は，フィルタリング処理に要する時間と比較すると十分に小さい．さらに，フィルタ済み投影像を繰り返し使用しない場合はフィルタリング処理に B 倍の時間を要する．したがって，フィルタ済み投影像をメインメモリへ転送し，繰り返し使用する工夫は有効だといえる．

次に，**図 4** にフィルタリング処理，逆投影処理，CPU・GPU 間のデータ転送，および後処理に要する時間の比率を示す．なお，CPU・GPU 間のデータ転送時間はダウンロードおよびリードバックに要する時間をまとめた値である．図 4 より投影像分割方式は，$C = 2$ では後処理に要する時間の比率が約 8%と増加し，他の処理は時間の比率が減少する．なぜならば，後処理を除く処理は C に比例して高速化できるが，後処理に要する時間は C に比例して増加するためである．ゆえに，C が増加すると後処理が性能ボトルネックになると考えられる．一方，ボリューム分割方式ではフィルタリング処理の比率が $C = 1$ の約 10%から $C = 4$ では約 25%と増加する．この方式では，フィルタリングを除く処理は C に比例して高速化できるため，C が増加するとフィルタリング処理が性能ボトルネックになると考えられる．

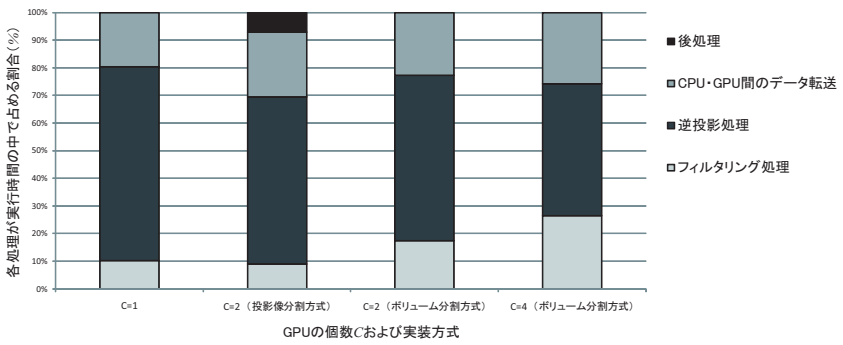


図 4 各処理に要する時間の比率（%）

5.2 転送時間の隠蔽に対する評価

図 4 で示すように，実行時間のうち CPU・GPU 間のデータ転送に要する時間は約 20〜25%を占める．したがって，ストリームにより CPU・GPU 間のデータ転送時間を隠蔽する効果は大きいと考えられる．しかし，逆投影処理では CPU・GPU 間のデータ転送時間を隠蔽できない．なぜならば，逆投影処理ではテクスチャメモリによる線形補間が必要であり，仕様[2] により CPU・GPU 間のデータ転送時間を隠蔽できない cudaArray 型[2] を用いるためである．それゆえ，フィルタリング処理でのみ CPU・GPU 間のデータ転送時間を隠蔽できる．**表 3** は各 C に対して，ストリームにより隠蔽できた時間，および隠蔽できた時間が CPU・GPU 間のデータ転送時間の中で，どの程度の割合を占めるかを示している．表 3 で示すように，投影像分割方式のみ隠蔽できた時間が短い．その理由は，フィルタリング処理でのみ CPU・GPU 間のデータ転送時間を隠蔽できるのに対し，投影像分割方式ではフィルタリング処理に要する時間が短いためである．また表 2 より，C が大きくなると CPU・GPU 間のデータ転送時間は短くなる．その結果，表 3 で示すように C が大きくなると隠蔽できる時間の割合は上昇する．

6. ま と め

本稿では，VRAM の容量を超えるような大規模なボリュームをサブボリュームに分割し，複数の GPU を用いて再構成する 2 つの手法を提案した．提案手法は各 GPU で投影像を分担する投影像分割方式，および各 GPU でサブボリュームを分担するボリューム分割方式からなる．評価実験の結果，1 つの GPU では VRAM に格納できないボリュームを再構成できた．また，4 つの GPU を用いた場合は 1 つの GPU を用いた場合と比較して 2.6 倍高速であった．さらに拡張性を評価した結果，投影像分割方式では生成したボリュームを合成する処理が性能ボトルネックであると分かった．一方，ボリューム分割方式はフィルタリング処理が性能ボトルネックである．

今後の課題には，投影像分割方式およびボリューム分割方式を改善し，さらに拡張性に優れた手法を開発することがあげられる．

表 3 ストリームにより隠蔽できた時間（秒）および割合（%）

		隠蔽できた時間	隠蔽できた割合
$C = 1$	基本実装	2.81	15.84
$C = 2$	投影像分割方式	2.25	18.40
	ボリューム分割方式	2.79	23.13
$C = 4$	ボリューム分割方式	2.98	33.26

謝辞 本研究の一部は，科学研究費補助金基盤研究（A）（2024002）および大阪大学グローバル COE プログラム「予測医学基盤」の補助による．

参 考 文 献

1) GPGPU: General-Purpose Computation Using Graphics Hardware (2007). `http://www.gpgpu.org/`.
2) nVIDIA Corporation: CUDA Programming Guide Version 1.1 (2007).
3) Okitsu, Y., Ino, F. and Hagihara, K.: Fast Cone Beam Reconstruction Using the CUDA-enabled GPU, *Proc. 15th Int'l Conf. High Performance Computing (HiPC'08)*, pp.108–119 (2008).
4) Scherl, H., Keck, B., Kowarschik, M. and Hornegger, J.: Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA), *Proc. Nuclear Science Symp. and Medical Imaging Conf. (NSS/MIC'07)*, pp.4464–4466 (2007).
5) Noël, P.B., Walczak, A.M., Hoffmann, K.R., Xu, J., Corso, J.J. and Schafer, S.: Clinical Evaluation of GPU-Based Cone Beam Computed Tomography, *Proc. High-Performance Medical Image Computing and Computer Aided Intervention (HP-MICCAI'08)* (2008).
6) Feldkamp, L.A., Davis, L.C. and Kress, J.W.: Practical cone-beam algorithm, *J. Optical Society of America*, Vol.1, No.6, pp.612–619 (1984).
7) Shepp, L.A. and Logan, B.F.: The Fourier Reconstruction of a Head Section, *IEEE Trans. Nuclear Science*, Vol.21, No.3, pp.21–43 (1974).