社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－108 (7)
2006／10／6

グリッド環境において遊休GPUを活用するための資源選択手法

小谷裕基† 伊野文彦† 萩原兼一†

本稿では，GPUグリッドにおける遊休資源の定義を示し，資源選択手法を提案する．ここでGPUグリッドとは，GPUを装備するPCを計算資源とするデスクトップグリッドを指し，GPUを用いる大規模な汎用計算処理の実現を目的とする．定義は次の2つの観点に基づいて実験的に定める．(1) 資源所有者への外乱を最小化すること，および (2) グリッドユーザへ提供する演算性能を最大化すること．提案手法は低オーバヘッドで遊休GPUを検出することを目的として，スクリーンセーバを基とし，資源のVRAM使用量およびCPU使用率を調べる．また，ベンチマークおよびマッチメイキングを組み合わせることで資源選択を実現する．実験の結果，遊休資源の定義が妥当であることを示せた．また，高々262ミリ秒の低オーバヘッドで遊休資源を検出できた．

A Resource Selection Method for Using idle GPUs in Grid Environments

Yuki Kotani,† Fumihiko Ino† and Kenichi Hagihara †

This paper presents a resource selection method and shows a definition of idle resources for the GPU Grid. The GPU grid here consists of desktop computers at home and the office, utilizing idle GPUs and CPUs as computational engines for GPGPU applications. We experimentally define the idle state that minimizes interference to resource owners and maximizes application performance provided to grid users. Our method is based on a screensaver-based approach with low overhead sensors. The sensors detect idle GPUs by checking video random access memory (VRAM) usage and CPU usage on each computer. Detected resources are then selected according to a matchmaking framework and benchmark results. The experimental results show that the definition is reasonable. We also find that our method achieves a low overhead of at most 262 ms, minimizing interference to resource owners.

1. はじめに

GPU（Graphics Processing Unit）とは，グラフィクス処理を高速化するためのプロセッサである．最近のGPUは単精度ではあるもののCPUを超える浮動小数点演算性能を持ち，プログラマビリティも向上している．結果，GPUを汎用計算に用いる試みGPGPU[1]が注目を集めている．

我々はGPGPUアプリケーションを高速化するためのデスクトップグリッドの実現を目指している．GPUグリッドはデスクトップグリッドの一種であり，家庭やオフィスにある遊休状態のCPUに加え，GPUを汎用計算資源として用いる．GPUを用いることで，デスクトップグリッドはより魅力的な計算環境になり得る．

GPUはこれまで描画用プロセッサとして占有されることを前提にしてきたため，資源所有者およびグリッドユーザがGPUを共有する場合，多くの技術的な問題が生じる．ここで，資源所有者とは，グリッドに対して資源を提供する者である．また，グリッドユーザとは，提供された資源を用いてグリッドアプリケーションを実行する者である．典型的な問題として，資源所有者とグリッドユーザの間での資源の競合が挙げられる．特に，GPUグリッドを構築する上で以下の3つの問題を解決する必要がある．

P1. 遊休資源の定義が不明確であること．GPUはその所有者のディスプレイ画面を出力するために設計されているため，グリッドユーザの立場からの遊休資源の定義が明確でない．したがって，適切な資源を選択するために，遊休資源の定義を新たに定める必要がある．ここで，その定義は，資源所有者側およびグリッドユーザ側の両者の立場から，次の2つを満たすべきである．(1) 資源所有者への外乱を最小化することおよび (2) グリッドユーザへ提供する演算性能を最大化すること．

P2. GPUを外部から監視する手法が未確立であること．多くのOSはCPU使用率を提供できるが，

† 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

GPU に関して同様の性能情報をユーザに提供する機能を持たない．一方，最近の GPU は性能カウンタを内蔵するが，そのカウンタはベンダが提供する特定のドライバおよびソフトウェアでしか読み出せない．したがって，資源の環境やアプリケーションのコードを修正することなく GPU を監視する手法が必要である．

P3. GPU のマルチチタスク処理が非効率的であること．現在の GPU はハードウェアレベルでコンテキストスイッチに対応していない[2]．このため，ソフトウェアで協調型のマルチタスクを実現している．ゆえに，複数のアプリケーションを同時に GPU 上で実行する場合，性能は著しく低下する．

P3 は重大な問題であるが，GPU 非ベンダが直接的に解決することは容易でない．したがって，GPU グリッドを協調型マルチタスク環境のシステムと想定した上で，P1 および P2 を解決し，資源の集合から遊休状態の GPU を適切に選択することを考える．

本稿では，P1 を解決するために，GPU の遊休状態を実験的に定義する．一方，P2 を解決するために，低オーバヘッドな資源監視を実現するスクリーンセーバ型の資源選択手法を開発する．この提案手法は資源の VRAM 使用量および CPU 使用率を調べることで遊休状態の GPU を検出する．また，ベンチマークおよびマッチメイキング[3] を組み合わせることにより資源を選択する．提案手法は，最も広く使われていて最新の GPU に対応している Windows を対象とする．

以降では，まず 2 章で GPU グリッドの構成および遊休状態の定義について説明し，3 章で提案手法について述べる．次に 4 章で実験結果を示す．最後に 5 章で本稿をまとめる．

2. GPU グリッド

GPU グリッドは GPU を明示的に汎用計算資源として扱う点を除いて既存のデスクトップグリッドと同じ構成である．このように，既存システムとの差異を小さくすることで提案手法の既存システムへの導入を容易にできる．

2.1 システムの概要

図 1 に GPU グリッドの構成を示す．主に以下の 3 つの構成要素からなる．

- グリッド資源．グリッド資源はインターネットに接続するデスクトップ PC であり，家庭やオフィスにある．普段は資源所有者が使用するが，遊休状態のときにジョブの実行に用いられる．
- 資源管理者．資源管理者は資源を監視および選択する．また，グリッドユーザのジョブを管理し，それぞれのジョブに対して利用可能な資源をリストとして返す．このリストは，マッチメイキングに用いられる（3.2 節）．
- クライアント．クライアントはジョブを実行したいグリッドユーザのためのフロントエンドである．また，グリッド資源にもなり得る．

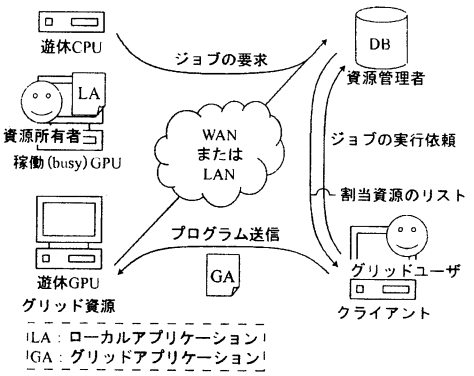


図 1 GPU グリッドの構成

以降では，グリッドユーザが資源を用いて実行するプログラムを GA（Grid Application）と呼ぶ．また，資源所有者が自身の資源によって実行するプログラムを LA（Local Application）と呼ぶ．

2.2 遊休状態の定義

遊休資源の定義は，1 章で述べたように，次の 2 点を満たすべきである．

R1. 資源所有者への外乱を最小化すること

R2. グリッドユーザへ提供する演算性能を最大化すること

これらを満たすために，次の 3 つの条件を全て満たす資源を遊休資源と定める．

*D*1. 資源所有者が対話的に資源を操作していない

*D*2. GPU が LA を実行していない

*D*3. CPU が GPU の性能を最大限に引き出すために十分な遊休状態にある

*D*2 は R1 を満たすために最も重要な条件である．また，R2 を満たすためにも必要である．この理由は，GPU がプリエンプティブなマルチタスクに対応していないからである．*D*2 を満たさない資源でアプリケーションを実行するとその性能が大きく低下する．結果，資源所有者が GA に妨げられる．また，グリッドユーザへ提供する演算性能も低下する．

資源所有者がディスプレイ画面を通して対話的に自らの PC を操作している場合も，D2 のときと同様の理由で R1 を満たさない．また，資源所有者が自らの

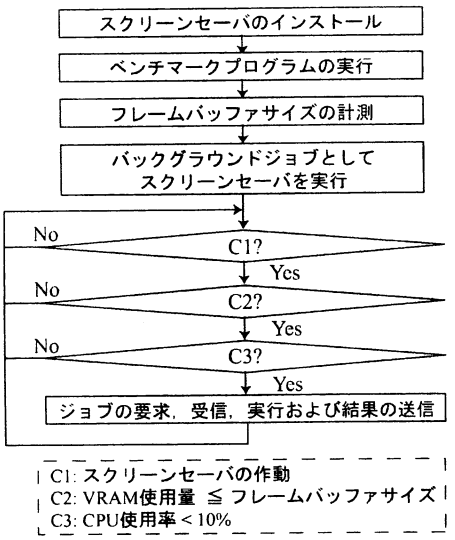


図 2 資源監視の流れ

操作するウィンドウにフォーカスを合わせている場合，GA のスループットが低下する（4.1 節）．したがって R2 も満たさない．以上から，$D1$ は遊休状態の必要な条件である．

$D3$ は R2 を満たすための条件である．GPU アプリケーションの実行には通常，CPU の介入が必要である．ゆえに，R2 を満たすためには，CPU は遊休状態でなければならない（4.1 節）．

3. 資源選択手法

本章では，遊休資源を検出する手法および検出した資源の集合から資源を選択する手法について述べる．

3.1 遊休資源の検出

図 2 に資源監視の流れを示す．$D1$〜$D3$ を満たす資源を検出するために，提案手法は次の 3 ステップで資源を調べる．

C1. スクリーンセーバの作動

C2. VRAM 使用量 ≦ フレームバッファサイズ

C3. CPU 使用率 < 10%.

C1，C2 および C3 はそれぞれ $D1$，$D2$ および $D3$ を資源が満たすか否かを調べることを目的としている．

まず，資源が $D1$ を満たすか否かは，スクリーンセーバの作動により調べる．低オーバヘッドで資源所有者が資源を操作していないことを検出できる．また，ポーリングによる手法[4] に比べ，資源所有者が作業を早く再開できる．

提案手法のスクリーンセーバは，資源が GA を実行している間，ディスプレイ画面を書き換えない．これにより，スクリーンセーバの作動による CPU および GPU の負荷が増大することを回避し，GPU における最大限の性能をグリッドユーザに提供できる．

資源が $D2$ を満たすか否かは，VRAM 使用量を調べることで評価する．GPU は常に画面の出力（フレームバッファ）に必要な容量を VRAM に確保する．また，GPU プログラムを実行する際にはその実行に必要な容量を VRAM に確保する．したがって，監視時の使用量と初期の使用量（フレームバッファサイズ）を比較することで資源が $D2$ を満たすか否かを評価できる．VRAM を用いる提案手法は，次の利点がある．

- 修正不要
  VRAM 使用量は DirectDraw の GetCaps 関数を用いて簡単に調べられる．Windows PC であれば利用でき，特別なライブラリやハードウェアは必要ない．
- 低オーバヘッド
  GetCaps 関数はデバイスドライバから VRAM の情報を取得する．GPU の介入が無く，低オーバヘッドで調べられる．

初期の使用量はスクリーンセーバをインストールするときに計測する．この理由は，初期の使用量が，ディスプレイ画面の解像度および色深度だけでなく，デバイスドライバのバージョンなどのハードウェアおよびソフトウェア環境にも依存するためである．

資源が $D3$ を満たすか否かは，OS の提供する性能情報を用いて評価する．予備実験（4.1 節）より，CPU 使用率が 10%未満の資源を遊休とする．この性能情報は，VRAM 使用量と同様，GPU の介入無しに低オーバヘッドで取得できる．

3.2 遊休資源の選択

検出した遊休資源の集合から資源を選択するために，提案手法では，ベンチマークによる手法[5] およびマッチメイキングによる手法[3] を組み合わせる．

ベンチマークによる手法[5] は GPGPU アプリケーションを対象として資源の性能を計測することを目的とする．実測する理由は，GPU の仕様が必ずしも資源の実効性能を表さないからである．実際に，異なるデバイスドライバを用いることにより，ミドルレンジの GPU がハイエンドの GPU より高速に動作する場合がある．したがって，提案手法は，スクリーンセーバのインストール時にベンチマークプログラムを実行し，遊休状態にある資源の実効性能を計測する．

一方，マッチメイキングによる手法[3] は柔軟かつ一般的な資源選択のフレームワークを提供することを目的とする．GPU は計算資源として発展途上である．例

表 1 実験環境

	PC1	PC2	PC3
CPU	Pentium 4 3.4 GHz	Pentium 4 3.0 GHz	Pentium 4 2.8 GHz
GPU	nVIDIA GeForce 7800 GTX	nVIDIA GeForce 6800 GTO	nVIDIA Quadro FX 3400
フィルレート	6.88 Gpix/s	4.2 Gpix/s	5.6 Gpix/s
バス	PCI Express		
ドライバ	Ver 79.70	Ver 78.01	Ver 66.93

えば，ある環境で実行できるアプリケーションが，ドライバやアーキテクチャの違いにより，他の環境で実行できないことがある．したがって，GPU グリッドでは，ユーザが適切な資源を選択できるように，柔軟なフレームワークが必要である．

4. 実　　験

表 1 に実験環境を示す．実験には，異なる CPU および GPU を持つ 3 つの計算機を用いた．

GPGPU アプリケーションとしては次の 3 つを用いた．2048 行列の LU 分解 (LU)[6]，係数行列 64 の共役勾配法 (CG)[7] および 2 次元／3 次元剛体位置合わせ (RR)[8]．それぞれの特徴について以下に簡単に説明する．

- LU : テクスチャに格納した行列データを操作することで LU 分解を実現する．CPU は GPU が処理すべきテクスチャの位置を計算する．
- CG : LU 分解と同様，テクスチャを用いる．GPU が処理すべきテクスチャの位置を GPU が計算するため，CPU の負荷は LU 分解に比べて小さい．
- RR : 3 つのアプリケーションの中で最も CPU の負荷が小さい．一方，2 次元画像および 3 次元ボリュームデータを処理するため，GPU 側の処理は最も重い．

4.1 遊休資源の定義に対する評価

遊休資源の定義が妥当か否かを検証するために，遊休状態の資源および稼働状態 (busy) の資源における GA および LA のスループットを比較する．ここで，スループットとは，1 秒間あたりにおけるアプリケーションの実行回数である．

稼働状態とは，以下に示す $D1$，$D2$ および $D3$ の否定である (2.2 節)．

$\overline{D1}$: 資源所有者が資源を対話的に操作している状態．実験では，ウェブページを閲覧中の所有者に対しグリッドジョブが与える外乱を測定することを目的とし，ウェブページ描画中に GA を実行し，それぞれのスループットを計測した．ウェブページの描画にはベンチマークソフトである PCMark05 を用いた．

$\overline{D2}$: GPU が LA を実行している状態．実験では，GPU アプリケーションへの外乱を測定することを目的とし，LA 実行中における GA のスループットを測定した．各アプリケーションには LU，CG または RR を用いた．

$\overline{D3}$: CPU が GPU の性能を最大限に引き出せない状態．実験では，CPU の負荷を変化させたときの GPU における実効性能の振る舞いを測定することを目的とし，0%から 100%までの異なる CPU 使用率で GA のスループットを計測した (**図 3**)．

図 3 に $\overline{D3}$ の結果を示す．全てのアプリケーションにおいて CPU 使用率の増加とともにスループットが低下している．これらの結果から，CPU 使用率が高々10%の資源を遊休とするのが妥当である．ケーススタディにおいても，検出できる遊休時間の観点から 10%の閾値が適切である (4.3 節)．

図 4 に $\overline{D1}$ および $\overline{D2}$ における結果を示す．まず $\overline{D1}$ について検証する．ウェブページ描画時，LU (図 4(a)) および CG (図 4(b)) は，スループットが大きく低下している．これは，LU および CG は RR に比べて CPU の介入が多いこと，およびウィンドウフォーカスが資源所有者の操作するウィンドウにあることが原因であると考えられる．フォーカスのない GA は OS により低い優先度となる．結果，CPU が介入するまでの時間が増加し，GPU が遊休状態である時間が増加する．一方，ウェブページの描画回数は秒間約 2 ページから約 0.5 ページに低下した．このように，$\overline{D1}$ にある資源は，ジョブの実行に用いるべきでない．

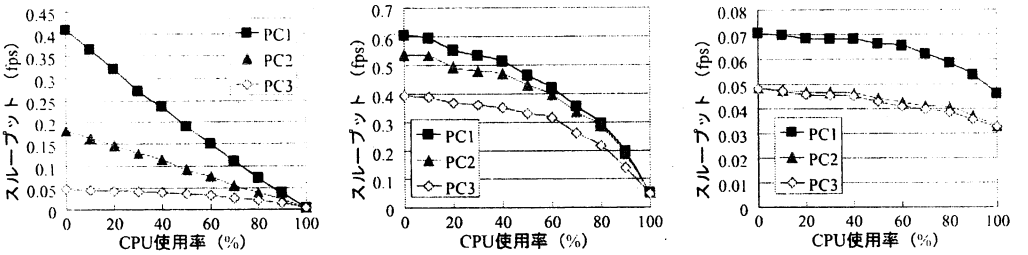
最後に $\overline{D2}$ について検証する．図 4 において，資源所有者が GPU アプリケーションを実行した場合，GA のスループットが大きく低下している．このように，複数の GPU アプリケーションを同時に実行した場合，それらのスループットは大きく低下することから，$D2$ は遊休状態の条件に必要であると考える．

以上のことから，本稿で定義した遊休資源の定義は，R1 および R2 の観点から，妥当な定義であると考える．

4.2 オーバヘッドの評価

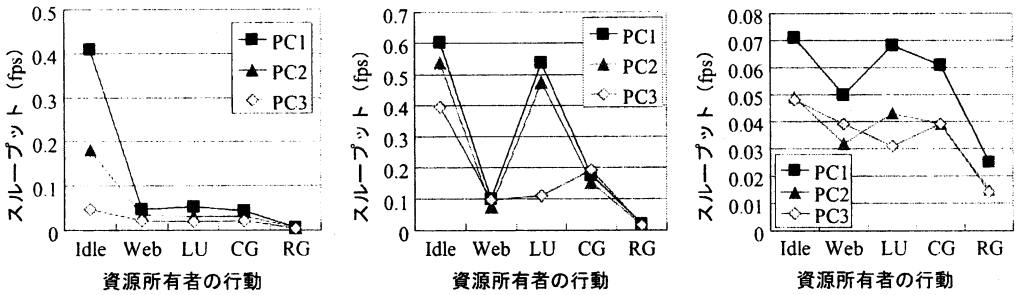
ここでは提案手法における監視のオーバヘッドを評価する．また，提案手法の監視が LA にどの程度の外乱を与えるかを調べる．実験では，LU，CG および RR を LA として用いた．

表 2 に，PC1 について，提案手法が LA にどの程度の外乱を与えたかを調べた結果を示す．PC1 が，提案手法が LA に最も大きい外乱を与えた資源である．



(a) LU (b) CG (c) RR

図 3 異なる CPU 使用率における GPGPU アプリケーションのスループット



(a) LU (b) CG (c) RR

図 4 所有者の行動時における GPGPU アプリケーションのスループット

表 2 提案手法の監視が 3 つの LA に与える外乱（秒）

LA	PC1		
	監視なし	監視あり	監視による外乱
LU	2.5	2.6	0.1
CG	1.7	1.8	0.1
RR	14.2	14.5	0.3

最も大きい外乱は RR に与えた 300 ミリ秒である．この時間は RR の実行間に比べて十分短いと考える．

300 ミリ秒の外乱は，262 ミリ秒ある監視のオーバヘッドによるものである．その内訳は C1 が 190 ミリ秒，C2 が 2 ミリ秒，および C3 が 170 ミリ秒である．スクリーンセーバの作動は描画を伴うため GPU を使用するが，このとき所有者は作業をしていないことおよび 190 ミリ秒が短時間であることから，このオーバヘッドは問題ないと考える．VRAM および CPU 使用率を調べる時間は CPU 側で実装していることおよび性能情報を参照するのみであることから短い．したがって，提案手法は所有者への外乱を最小化する低オーバヘッドな手法であるといえる．

4.3 ケーススタディ

最後に，実環境で提案手法が検知できた遊休時間を示す．スクリーンセーバを我々の研究室にある 3 つの PC にインストールし，30 日間監視した．スクリーンセーバが作動するまでの待ち時間は 5 分である．

表 3 に，提案手法の各ステップ C1〜C3 における資源状態の分類を示す．提案手法が遊休資源をどの程度的確に検出できているかを示すことを目的としていることから，短時間（1 分）間隔で資源の状態を監視し，分類している．

表 3 より，提案手法は PC1 において延べ 2796 分のうち 1859 分の遊休時間を検出できている．このように，提案手法は約 61〜72%の遊休時間を検出できている．残りの 28〜39%は主にスクリーンセーバが作動するまでの 5 分間に資源が遊休状態であった時間である．

PC1 では，C1 により稼働時間が 9059 分から 185 分に減っている．これは提案手法が稼働時間の 97%を最初のステップで効果的に排除できることを示してい

表 3 監視ステップ C1〜C3 にあった資源の時間（分）

ステップ	PC1 の状態				PC2 の状態				PC3 の状態			
	遊休	稼働			遊休	稼働			遊休	稼働		
			NC2	NC3			NC2	NC3			NC2	NC3
初期	2796	9059	176	2025	2251	9374	668	270	2272	8778	496	1715
C1	1859	185	86	161	1629	196	196	25	1392	145	102	47
C2	1859	99	—	99	1629	0	—	0	1392	43	—	43
C3	1859	—	—	—	1629	—	—	—	1392	—	—	—

—：存在しない状態　　NC2：C2 の否定（VRAM 使用量 > フレームバッファサイズ）
NC3：C3 の否定（CPU 使用率 ≥ 10%）

る．残りの 3%も C2 および C3 により排除できている．

ところで，C3 における CPU 使用率の閾値を 10%から 20%とすると，検出できた遊休時間は 3%しか増加しなかった．したがって，10%という閾値は，R1 および R2 の観点から妥当であると考える．このように，提案手法は 3 ステップによって効果的に遊休資源を検出できる．

各資源における遊休状態の持続時間を分類すると，5 分以上 10 分未満であった割合は PC1 が 21%，PC2 が 12%，および PC3 が 16% であった．また，10 分以上であった割合は PC1 が 42%，PC2 が 57%および PC3 が 38%であった．この結果は，資源の遊休状態が 10 分以上続く可能性があることを示している．結果は資源所有者に依存するが，一つの資源において 10 分以下のタスクを実行させるグリッドを構築する上で有用な情報である．ここで，GPGPU アプリケーションは VRAM 容量の制限などから，5 分以下の実行時間のものが多い．したがってこの提案手法が検知できた時間は GPU グリッドを構築する上で十分な長さであると考える．

5. おわりに

本稿では GPGPU アプリケーションを高速処理することを目的として，GPU グリッドのための資源選択手法を提案した．また，GPU グリッドにおける遊休資源の定義を示した．提案手法は低オーバヘッドで遊休 GPU を検出することを目的として，スクリーンセーバを基に VRAM 使用量および CPU 使用率を監視する．

実験の結果，定義は資源所有者への外乱を最小化し，かつグリッドユーザへ提供する演算性能を最大化する上で妥当であった．また，提案手法は高々262 ミリ秒の低オーバヘッドな監視を実現でき，LA に与える外乱は高々300 ミリ秒であった．ケーススタディでは，提案手法により，10 分以上遊休状態の続く可能性のある資源を効果的に検出できた．

今後は，実用的なアプリケーションを用いて，GPU グリッドの有用性を示したい．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (B) (2) (18300009) および特定領域研究 (17032007) の補助による．

参考文献

1) GPGPU: General-Purpose Computation Using Graphics Hardware (2005). `http://www.gpgpu.org/`.
2) Pronovost, S., et al.: Windows Display Driver Model (WDDM) v2 And Beyond, *Windows Hardware Engineering Conf. (WinHEC'06)* (2006). `http://www.microsoft.com/whdc/winhec/trackdetail06.mspx?track=11`.
3) Raman, R., et al.: Resource Management through Multilateral Matchmaking, *Proc. 9th IEEE Int'l Symp. High Performance Distributed Computing (HPDC'00)*, pp. 290–291 (2000).
4) Litzkow, M.J., et al.: Condor - A Hunter of Idle Workstations, *Proc. 8th Int'l Conf. Distributed Computing Systems (ICDCS'88)*, pp. 104–111 (1988).
5) Buck, I., et al.: GPUBench: Evaluating GPU Performance for Numerical and Scientific Application, *Proc. 1st ACM Workshop General-Purpose Computing on Graphics Processors (GP^2'04)*, pp.C–20 (2004).
6) Ino, F., et al.: Performance Study of LU Decomposition on the Programmable GPU, *Proc. 12th IEEE Int'l Conf. High Performance Computing (HiPC'05)*, pp.83–94 (2005).
7) Corrigan, A.: Implementation of Conjugate Gradients (CG) on Programmable Graphics Hardware (GPU) (2005). `http://www.cs.stevens.edu/~quynh/student-work/acorrigan_gpu.htm`.
8) Ino, F., et al.: Fast 2-D/3-D Registration Using a Laptop PC with Commodity Graphics Hardware, *Proc. Computer Assisted Radiology and Surgery: 20th Int'l Congress and Exhibition (CARS'06)*, pp.53–54 (2006).