Vol.2012-SLDM-155 No.23
Vol.2012-EMB-24 No.23
2012/3/3

組み込みシステムの GPGPU 適用のための アーキテクチャ提案

山木戸伸行[†a)]　　長谷川修[†b)]

[†] 東京工業大学大学院総合理工学研究科知能システム科学専攻，横浜市

Email: a) yamakido.n.aa@m.titech.ac.jp　b) oh@haselab.info

あらまし　近年，ビックデータの処理を目的として，高い演算能力へのニーズが高まっている．しかし，組み込みシステムでは，コスト，消費電力などの問題により搭載できる CPU の数には限界がある．我々は，この問題の解決策の一つとして GPGPU の利用を検討している．本研究では，GPGPU を組み込み系に組み入れてスケラビリィティを持った処理を行うための計算アーキテクチャ MTG(Multi-Task for GPGPU)を提案する．そして，MTG にプラットフォーム化を実施し MTG Platform として実装した．

キーワード　GPGPU，組み込みシステム，マルチタスク

The architecture propose for using GPGPU in the embedded system

Nobuyuki Yamakido[†a)]　and Osamu Hasegawa[†b)]

[†]Department of Computational Intelligence and System Science, Tokyo Institute of technology, Yokohama-shi 226-8503 Japan

Email: a) yamakido.n.aa@m.titech.ac.jp　b) oh@haselab.info

Abstract In recent year, there is a growing need for high performance computing for processing of image and big-data. However there is a limit to have the number of CPU in an embedded system, because of cost and electric requirements. We consider one of solutions is the use of GPGPU. In this research, we propose MTG (Multi-Task for GPGPU) architecture for using GPGPU efficiently in an embedded system to process intelligent information processing. And we develop a MTG technology platform.

Key Words GPGPU, embedded system, Multi-Task

1.　はじめに

近年，動画像やビッグデータなど高い処理性能が要求される情報処理に対するニーズが高まっている．また Honda の ASIMO や TOYOTA のパートナーロボット等など知能情報処理を取り入れたロボット等の組み込みシステムの研究開発も進んでいる．これらを実現していくために問題になっているのが演算処理の問題である．例えば ASIMO に搭載れている CPU は 30 個にもおよび，全体の消費電力 600～900W の内，約 400W をも占めている[1]．組み込みシステムでは，処理内容が非常に大きくなるアルゴリズム，FFT 等（高速フーリエ変換）に関しては LSI と呼ばれる専用回路化で実現している．しかし，ロボットなどの製品化を考えた場合に複雑な知能情報処理などのアルゴリズムを LSI 化していく事は，機能の多様性の問題やアルゴリズムのアップデート，オンライン学習等といった処理内容の変化等の問題などにより困難である．そこで，その解決のための一手段として高い計算処理能力を持つ GPGPU の利用に着目する．GPU は汎用系の CPU に比べて，数倍程度，消費電力当たりの処理性能が優れている[2]．また，GPU メーカーである NVIDIA 社の長年の開発により GPGPU を C 言語ベースで扱うことが出来るようになっている．消費電力，開発の難易度の点から見ても，組み込みシステムと GPGPU は非常に相性が良いと考えられる．そこで，本研究では将来的に，組み込みシステムに GPGPU を取り入れる際や知能情報処理を実行する際に，生じると考えられる問題点の解決法を示していき，GPGPU をマルチタスクで利用

ⓒ 2012 Information Processing Society of Japan

する事により，演算効率の向上を実現する GPGPU のアーキテクチャを提案する．また，それらの提案アーキテクチャの容易な利用を可能とするミドルウェア MTG (Multi Task for GPGPU)Platform を実装した．

2. 研究背景

GPGPU に関しては，非常に多くの研究が行われている．特に特定のアルゴリズムを GPGPU に適用する研究は多く報告されており，非常に高いパフォーマンスが得られている．GPGPU で高いパフォーマンスを実現する際，よく用いられる手法が，アルゴリズム全体，もしくは大部分を GPU 内で実行できるように設計する手法である．これは，実行パフォーマンスのみを考えると非常に有効ではあるが，複雑な設計になってしまうことが多い．また，プログラムが特定のデータに限定さえている場合も多く，修正することも非常に困難である．そのため，現実的なアプリケーション開発を考えると大部分の処理は通常通り CPU 側で実施して, 負荷部分になるロジックのみを CPU 側のコードとは独立になるように設計し，GPGPU 化を行うべきである．この場合，CPU と GPU の間でデータのやり取りが頻繁に発生する．CPU と GPU は独立したデバイスになっているため転送のオーバヘッドが非常に大きい．そのため，NVIDIA 社も言及しているがある一定以上の大きさで転送しなければボトルネックになってしまう[2]． 解決法の一つとして考えられるのが，データを集約しておき，大規模な容量での転送を行う手法である．中村ら[3]も，この手法を用いてコンパイル時に，転送命令を変更して集約させる提案を行っている. 本研究における, メモリ転送問題の解決提案も，根幹にあるのは同様の発想である．また，GPGPU はある程度の問題規模がないと GPU のリソースを使い切ることが出来ず，高いパフォーマンスを出すことが出来ない．この問題に対して，NVIDIA 社はコンカレントカーネルという GPU 内で同時に異なるタスクを実行できる機能を 2010 年に発表した Fermi アーキテクチャより実装している．これにより，小規模なタスクでも同時に実行する事で, GPU リソースを有効活用する事が可能となっている．これに関するパフォーマンス向上は，我々の発表[6]も含めて幾つか報告されている．本研究では，この機能を有効に活用して，独立したスレッドで GPGPU タスクをマルチタスク的に利用する事により，GPGPU の処理の共通化を行い，演算効率向上を計る MTG (Multi-Task for GPGPU)アーキテクチャを提案する．また，多くの研究で，効率の良い GPGPU のアーキテクチャが提案しているが，その提案を実際の開発に組み込むのは大きな課題となっている．本研究では，GPGPU の利用に際し，本論文の提案内容を意識する事無く，簡易に利用できるようにプラットフォーム化を行う．

3. 問題点と解決提案

GPGPU でパフォーマンスの障害になる問題点として下記の 4 つが挙げられる．これを満たさない限り，導入コストに対する効果が十分に得られない可能性がある．そのため，GPGPU アプリケーションの開発においては，アルゴリズムの動作原理や入力データの特性などまで把握しておき，適切な設計を行う必要がある．

問題点 1

粒度がある程度，大きな処理である必要がある．GPGPU は同時並列数が大きいため，問題規模が小さい場合，粒度が非常に小さくなってしまう．

問題点 2

CPU 側と GPU 側で通信を頻繁に行ってはならない. 一度の通信では 0.05msec 程度のオーバヘッドが発生する．

問題点 3

ベクトル型演算機としてふるまうため，連続したメモリアクセスを行う必要がある．

問題点 4

If 分岐によるデバイスメモリへのアクセスは控える．分岐予測を持っていないため，両分岐の処理を実行してしまう．

問題点③，④に関しては，設計次第で回避することが出来る．すでに，様々な手法がアルゴリズムに合わせて提案されている．しかし，問題点①，②に関しては対象となるアルゴリズム，または入力データに依存しており回避する事が難しい．この様な問題点から，GPGPU の利用は，現段階においては，一般的なプログラムに関してエンコーダなど非常に限られた用途にとどまっている．上記の問題点を解決しなくては，組み込み系での利用にも同様な弊害となるであろう．そこで, 問題①と問題②に対して. 問題点を明確化にし，その解決法を提案する．また，リアルタイム性についても言及する．なお，本研究で行う実験は，全て表 1 の計算環境で行う．

表 1: 実行環境

CPU	Core i7-980X
GPU	Tesla 2050
OS	Windows Server 2008R2
メモリ	16GB
言語	C++，CUDA

ⓒ 2012 Information Processing Society of Japan

Vol.2012-SLDM-155 No.23
Vol.2012-EMB-24 No.23
2012/3/3

3.1 粒度問題

GPGPU には大量のコアが搭載されている．Fermi アーキテクチャの GPU には 16 基のマルチプロセッサが存在しており，マルチプロセッサ毎に SM プロセッサが 32 コアとなっている．つまり，合計で 512 個のコアが存在する．また，2 命令同時実行を行う仕組みとなっているため，最低でも 1024 並列を行う必要がある．しかし，よほど大きな問題でない限り，このような大きな並列性で実行した場合，一度の処理で数命令程度の実行内容しかなく，粒度が非常に細かくなってしまう．そのため，オーバヘッドが顕著となり，十分に性能を得ることが出来ない．並列処理を行う上で重要な法則にアムダールの法則と呼ばれるものがある．

アムダールの法則

$$X = \frac{1}{F + \frac{1-F}{N}} \quad \cdots \text{式(1)}$$

X : 性能上昇率
F : 並列化不可の割合
N : 並列数

この式は，並列数を増やしても，オーバヘッドが大きい場合，性能が向上しない事を示している．これは，言い換えると並列数が上昇する度に，リソースの利用効率が低下する事を意味する．このため，我々はタスクを少ないコア数で実行すると高い利用効率が維持できる点に着目した．そこで，GPGPU のリソースを同時に複数アプリケーションが利用するアーキテクチャを提案する．また，その有効性の検証のため，以下の実験を行った．

実験

11 個のベクトルの総和を求めるが．ベクトル長は変化させる．なお，実行時間のみを測定しており，転送時間は含めていない．

手法１

全コアを一つのタスクに割り当てて計算する．処理を 11 ベクトル分 11 回繰り返す．これは，通常の GPGPU の使用法である．

手法２

ベクトルの総和計算を GPU で同時に異なる処理を実行させるコンカレントカーネル機能を用いて，コアを１プロセッサずつ，ベクトル総和計算タスクに割り当てて計算する．これは，11 個のタスクが同時で実行されており，マルチタスク処理と同等の負荷となっている．

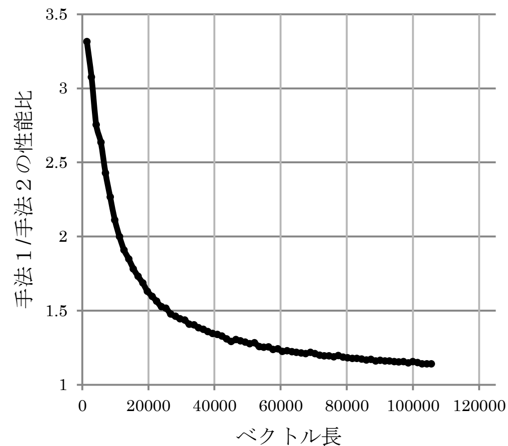

図１　問題規模毎の性能比

結果を図 1 に示す。これによると，問題規模が小さな処理においても約数倍の処理能力の改善が実現できている．想定道理，タスクに割り当てるコア数を分割する事により，並列度を減らして，粒度を太くできているのである．そのため，仮に実行時間が 11 倍かかっても，総実行時間はオーバヘッドの部分が，手法 1 では 11 回生じているのに対して，1 回で済んでいる．この特性を用いて複数の処理が GPGPU を共有し合いながら利用する．つまり，タスクがコアを共有する事を実現し問題の解決を目指す．

3.2 メモリ転送問題

提案を行う前に，メモリ転送に関して実測を行った．CPU と GPU は独立したデバイスとして存在しており，PCI Express 2.0x16 インターフェイスで接続されている．このインターフェイスの転送性能の理論値は 16GB/Sec であるが，NVIDIA 社が保証している速度は双方向 12GB/sec までである．図 2 は下記の実験環境において実際に測定した結果である．

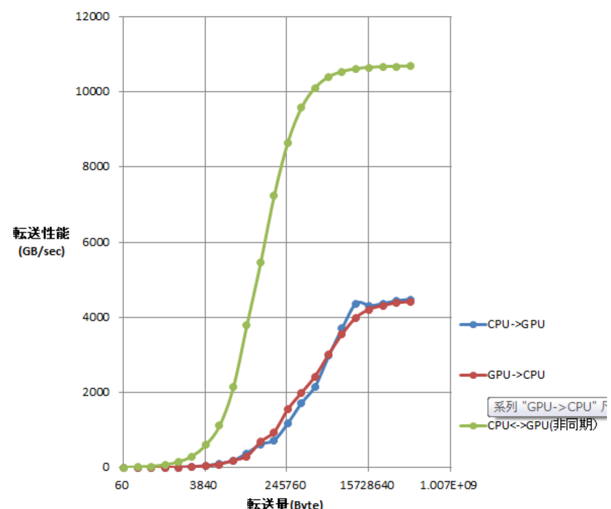

図２　メモリ転送性能

ⓒ 2012 Information Processing Society of Japan

NVIDIA 社が提供している GPGPU を実行するためのドライバ API で，転送に関するものについては，通常，利用する転送が完了するまで待機する転送と，非同期で実施していく転送の二種類がある．非同期の転送は，転送処理を事前に手続きすることで転送時間の一部を隠蔽して実行される．転送速度を測定した結果，理論値に近い値を出すためには，非同期転送を CPU 向きと GPU 向きの双方の転送を同時に行った場合のみであった．また，数 Kbyte の転送では数 MB/Sec 程度の転送性能しか出すことが出来ない事も判明した．さらに，一回の転送に最低でも 0.05msec 程度のオーバヘッドが生じてしまうことも分かった．

つまり，仮に複数の転送を一括で送ることが出来るなら，転送効率をあげることが出来る．しかし，いつ転送が実行されるかは，実行するまでは不確定である．そこで，複数の転送を非同期に管理して，自動的に集約，転送，再配置を実行する事により解決を図る．粒子問題の解決法としてマルチタスクでの利用を挙げた．マルチタスク化と転送の共通化は非常に相性が良い．マルチタスクで実施される転送命令を管理する事により，共通化する事が出来る．タスクが複数存在し，それぞれが，小規模な転送を多く行うなど実施した場合に非常に有効になる．

3.3 リアルタイム性

組み込みシステムへの適用を考えた際に避けることが出来なのが，リアルタイム性への要求である．GPGPU では，タスクの切り替えのたびに担当するコアが切り替わってしまうため，実行時間が毎回変わってしまう．これを解決するためにタスクには常にコアを一定数割り当てるようにしておくことにする．これに対して下記の簡易的な実験を行った．

実験

7 つの CPU スレッドで GPU に対して排他的に処理を行う．その際，7 つのタスクに振り分ける合計が常に GPU の全コア数になる場合と全コアをタスクに割り当てる場合で実行時間を測定する．なお，タスクはベクトルの加算計算としている．

結果は，初回実行時を除外すると，毎回 7.0msec 実行で誤差 0.3msec 程度になっている．パイプラインの構成上の初めの数回を除き，ほぼ同じ実行時間で処理される．これは，同一タスクを同一コアで引き続き実行しているからだと思われる．しかし，問題規模が大きくなっていくと，コア数の割り当てを全コア数にした方が，性能の向上が行える事も確認された．これは，GPGPU 自体が同時に多数のスレッドを実行すると共に，実行パイプライン構造を持っている為と考えられる．今回は，ある一定時間を超えるタスクに関しては全コアの割り当てとする事としている．タスクが最も高速に実行し，リソースの利用率の最適化を行うためのコア数の割り当てる問題に関しては，将来の課題としている．これを実現させるためには，タスクの実行時間予測を行う必要がある．

4. MTG (Multi Task for GPGPU) Platform

前節で列挙した提案法を意識しながらアプリケーションを構築するには，毎回，非常に複雑なシステム設計する必要があり，重要なアルゴリズム自体に注力することが出来ない．そこで，我々はプログラマが意識することなく提案内容による性能向上を取り入れることが出来るミドルウェア MTG (Multi-Task for GPGPU) Platform を開発した．

4.1 システム概要

図 3 に MTG Platform の全体システム図を示す．マルチタスク実現のためのコアの割り当てを行うタスクマネージャや，転送の共通化を非同期で実行するための 3 パイプライン 2 バッファ非同期転送を実装した．コアの最適配置に関しては，実行時間を計測して最も実行時間が大きいものに優先的にコアを割り当てている．なお，スケジューリング更新(コアの最適配置計画)に関しては 100msec ごとに実施している．

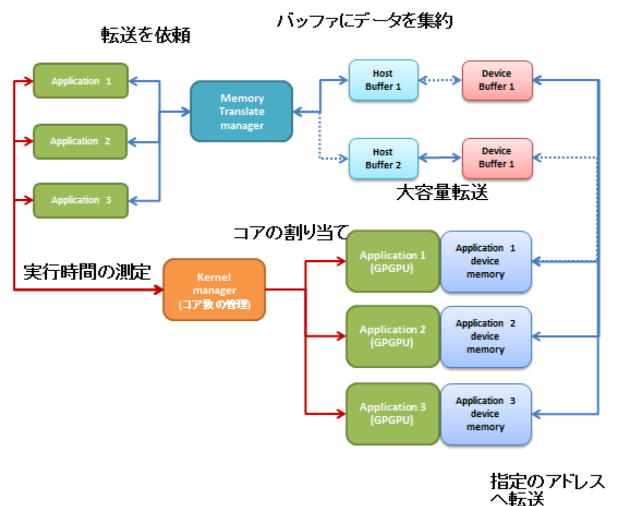

図 3　全体システム図

4.2　3 パイプライン 2 バッファ非同期転送

独立したタスクが，転送命令を発行する際，通常では図 4 のように順次転送が実施されていく．Application1 が DATA1 の転送を行っている最中には，Application2 が DATA2 の転送命令を発行しても待機状態となってしまう．一回の転送のオーバヘッドが 0.05msec 程度存在している事からも，これは大きなタ

ⓒ 2012 Information Processing Society of Japan

イムロスにつながってしまう.

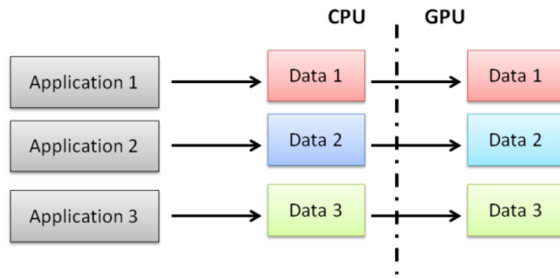

図 4　通常のメモリ転送

これでは，CPU と GPU 間の転送を何度も行う必要があり転送の効率が悪くなる．また，待機状態になってしまう Application が存在してしまい，最悪の場合は長時間，実行までに待機してしまう可能性が生じてしまう. そこで, 以下に提案する処理により解決を図る。

処理 1

同時点で通信を行おうとしている Application の Data を一回 CPU 側の別メモリ空間に転送しておき集約させる.

処理 2

集約したデータを GPU 側に転送を行う．その際，転送は非同期で行い，同時に GPU から CPU への転送も受け付ける.（CPU への転送も一連の処理と同様のことを実施する）

処理 3

GPU 側のデータは集約されたデータなので，本来転送するはずであった場所に GPU 内部でメモリ転送を行い再配置していく.

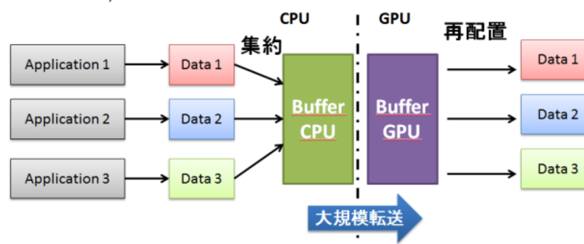

図 5　3 パイプライン 2 バッファ非同期転送

以上 3 つの処理をパイプライン化して動作させる. なお，GPU から CPU への処理に関しても，これとは独立に実施している．CPU 内部の転送と GPU 内部の転送を行い冗長のように思われるが，実行に必要な時間は CPU-GPU 間の転送時間の方が圧倒的に大きいため，隠蔽される．そのため，実質 Application 側から見ると通常の転送に見えるが，システム全体では，一回分の処理の待機と転送時間で転送を行う事が可能になる．これによる効果は，転送が頻繁に起こるプログラムであるほど有効に働く.

4.3 GPU タスクマネージャ

実行時間を予測してコア数を割り当てていき，実行時間の最適化を図る．実行時間が増大するタスクにはコア数を増やし，即座に完了するタスクにはコアの割り当てを少なくする．さらに，リアルタイムの指定をするタスクに関しては，コア数を一定のまま変えずにおく．また，提案時にも指摘しているが，問題規模の大規模化に伴うリソースの有効活用のために，実行時間がある一定を超えた場合には全コア割り当てを実施している．本研究では，実行時間予測に関しては前回実行に生じた時間を測定して, そのまま利用している.

4.4 性能評価

ベクトル演算を行い，性能を評価した. 時間の測定は CPU から転送して GPU で計算を実行, 演算結果を CPU に転送するまでを測定した．なお，実行時間は 10000 回実行した平均時間となっている. 結果を図 6 に示す.

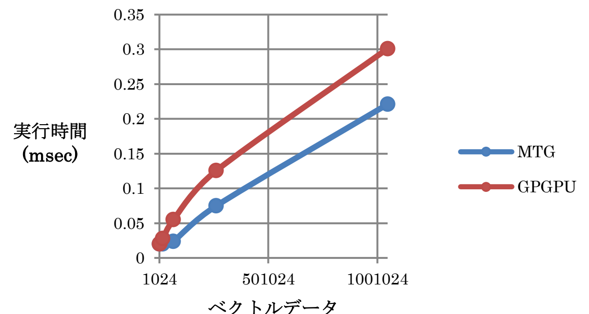

図 6　ベクトル計算による性能検証

GPGPU を通常利用する方法と比較して，MTG を利用してマルチタスクで処理する方法が，効率良い処理が実施出来ている事が確認できる．これは，データ数が少ない場合，粒度の小さな問題で特に顕著である．65536 ベクトルを処理する際，最大の効率化を果たしており MTG では実行時間が 0.024msec なのに対して GPGPU は 0.055msec 要した．これは，約 2 倍の性能向上である.

5.　応用問題

MTG Platform がベクトル演算のような単純な計算だけでなく，知能情報処理の様なアプリケーションでもパフォーマンスの向上が実現可能であるか検証する. 今回は，オンラインで学習可能である自己増殖型ニューラルネットワーク SOINN[5]を用いた．SOINN には，入力されたデータと作成されていく内部のノードと呼ばれるデータ群と距離計算を求めて近傍ノードを求める処理が存在する．処理負荷の大部分がここに関するものである．そのため，SOINN の処理時間はノード数

ⓒ 2012 Information Processing Society of Japan

n に対して O(n)を取る．また，SOINN 自体の計算は非常に軽量であるため，一度の処理では GPGPU の適用の効果が薄いため，１データ入力処理を行う処理を，64 データ同時処理を行えるように拡張したバッチ型 SOINN を開発した．これを GPGPU に適用することにより，256 次元のデータで，ノード数が 830 個程度の場合に約 9.2 倍の高速化した．これを用いて，SOINN を複数同時に形成しマルチスレッドで実行することで，MTG Platform の性能検証を行った．SOINN のパラメータは DeadAge = 500，lamda =1000 としている．精度検証は，欠損データを SOINN に与えて欠損地の推定を行っている．人工データを用いて 4 クラス，256 次元のデータを 3000 データ発生して入力する．以下が結果である．

表 2: 実行結果

	ノード数	予測値	正解値
CPU	829	2.057792	2.026700
GPU	830	2.057791	2.026700

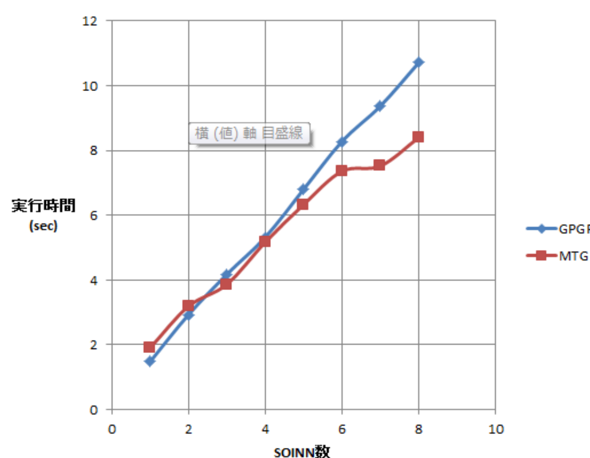

図 7 MTG 利用時の性能比較

図 7 より，MTG が有効に働いており，8 個の SOINN での検証では，約 20%性能が向上している事が分かった．なお，この効果はデータに依存しておらず，実データである気象庁のアメダスデータベースによる降水量予測の適用でも同様のパフォーマンスを確認している．この検証により MTG Platform が一般のアルゴリズムにも効果を発揮する事が確認できた．

6. おわりに

MTG アーキテクチャを提案することにより，一般的な処理の様な粒度の小さなタスクを行う組み込みシステムための計算アーキテクチャを実現した．しかし，未解決な課題も多く存在する．一つは，コア数の最適配置問題である．コア数の計画を効率よく行うためには，実行時間の予測を正確に行う事が不可欠である．これに関しては，本間ら [4]が様々な条件で，実行時間に関する考察を行っている．二つ目は，ロボット等の実際の機器への適用である．この場合，現時点での GPU 単体の消費電力，スペース，実行環境の問題からも，ネットワーク越し利用しに，ていくシンクライアント型の利用が最も適切であると思われ，複数の組み込みシステムが GPGPU を行うサーバを共有して実行していくシステムが理想である．ネットワーク越しでの利用を MTG Platform を拡張して実現していかなくてはならない．

参考資料

1) 垂井 康： “広瀬 真人：二足歩行ロボット「ASIMO」の生みの親”， アントレプレナー列伝３，武田計測先端知財団
2) NVIDIA CUDA Programming Guide, http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf
3) 中村 晃一，林崎 弘成，稲葉 真理，平木 敬: SIMD 型計算機向けループ自動並列化手法,情報処理学会研究報告． [ハイパフォーマンスコンピューティング] 2010-HPC-126(10), 1-8, 2010-07-2
4) 本間咲来 須田礼仁 ：GPGPU におけるデータ転送とカーネル実行のヒューリスティックスケジューリング，情報処理学会研究報告． [ハイパフォーマンスコンピューティング] 2011-HPC-129(22), 1-7, (2011)
5) Furao Shen and Osamu Hasegawa, "An Incremental Network for On-line Unsupervised Classification and Topology Learning" , Neural Networks, Vol.19, No. 1, pp.90-106, (2006)
6) 山木戸伸行, 長谷川修 ：自律型移動ロボットを実例とした組み込みシステムのための GPGPU を用いたマルチタスク実装，情報処理学会研究報告． [ハイパフォーマンスコンピューティング] 2011-HPC-129(22), 68-68, (2011)

ⓒ 2012 Information Processing Society of Japan