2010-16-02

GPGPU の実時間システムへの援用手法に関する研究
A Study on GPGPU Acceleration Method for Real-Time Systems

小山圭太 (Keita Koyama)　　Supervisor: 矢向高弘 (Takahiro Yakoh)

1　序論

近年，実時間制御システムは，プラントやマニュピレータ等の産業応用だけに留まらず，ヒューマノイドロボットや触覚制御等のユーザに近い分野へも応用されている．応用分野が広がると同時に，その制御システムはより複雑化し，処理時間が指数的に増加してしまう．この問題に対し，現在ではマルチプロセッサ及びマルチコア技術を用いた並列処理や，DSP のような副処理装置が用いられているが，分野によっては要求される性能が得られていない．一方，映像処理に特化した GPU(Graphics Processing Units) を汎用処理にも適用する GPGPU(General-Purpose computing on GPU) 技術が，超並列を用いた演算の高速化という点から様々な分野で注目を集めている．そこで本論文では，GPU による実時間制御システムの援用手法を提案し，実時間性を評価することで実時間制御システムにおける GPGPU の有用性を示す．

2　GPU による演算の高速化

GPU を実時間制御システムに応用する動機となった，GPU の演算性能を調査した．制御システムで用いられる既存の演算処理装置では十分な性能が得られていない例として，二足歩行ロボットにおける歩行パターン生成の計算を挙げる．このシステムでは，行，列それぞれが数千にもなる行列計算を扱う必要があり，計算時間の大部分を逆行列計算が占めていた．そこで，GPU の並列処理を用いて，この演算の高速化を試みた．図 1 にその結果を示す．グラフから，行列サイズが大きい場合に，GPU による演算のほうが CPU に比べて高速であることが分かる．このことは，GPU が大きな計算データサイズを扱う実時間制御システムの副処理装置として応用できる可能性を示している．

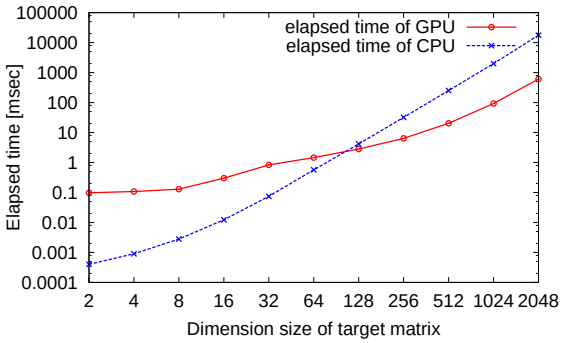

図 1: GPU による逆行列計算の高速化

しかしながら，実時間制御システムに求められる実時間性は，単に演算の高速性を意味するのではなく，予め決められた時刻にタスクを開始・完了できる能力である．つまり，GPGPU 技術を実時間制御システムに応用するためには，GPU で動作するプログラムにおける，実行時間の予想可能性が保証されなくてはならない．

3　実時間性を保証する GPU 開発手法

一般的に実時間制御システムではリアルタイム OS を用いることで，内部で実行されるタスクやプロセスの実時間性を保証している．本研究でも，リアルタイム OS である RTAI を用い，リアルタイムユーザ空間プログラムで開発を行った．リアルタイム OS には次の制約がある．実時間性を保証したいタスク内において，ソフトウェア割り込みによる Linux システムコールを行うと，実時間性を保証することができなくなる．一方，GPU で動作するプログラムやデータは，メインメモリから GPU のデバイスメモリへとロードされた後，GPU の演算器で計算される．このように GPU の利用には，NVIDIA®社が提供する CUDA の API を用いる必要がある．ところが，これら API の内部での処理の詳細は開示されていない．そのため，提供された API を無考慮に用いると，内部でソフトウェア割り込みによって Linux システムコールが呼び出され，実時間性を保証することが困難になる．そこで本研究では，まず様々な CUDA ドライバ API が，内部でソフトウェア割り込みを行うか否かを調査した．調査方法としては以下の方法を用いた．

- *strace* を用いる方法
  Linux ではシステムコールをトレースする，strace コマンドが用意されている．Linux システムコールはシステムコール・テーブルに登録されているが，リアルタイム OS のシステムコールは登録されていないので，コマンド実行時のログにコール名が明記されない．これを利用することで，各 CUDA ドライバ API がカーネル空間へのコンテキスト切り替えの有無を調べることができ，ソフトウェア割り込みを検出できる．
- システムコール・テーブルを用いる方法
  RTAI スケジューラでは，Linux のシステムコール・テーブルを拡張したものを利用されている．Linux システムコール・テーブルの最大長を超さない，システムコールベクトルならば，Linux システムコールとなる．RTAI スケジューラのソースを改変することで，調べることができる．

これらの方法を利用して，各 CUDA ドライバ API を用いた際のソフトウェア割り込みの有無を調査した．調査対象となる API は，GPU プログラム開発に最も重要と考えられる 39 個とし，その結果，13 個の CUDA ドライバ API がソフトウェア割り込みを伴うことがわかった．

この調査結果をもとに，CUDA ドライバ API の利用指針を提案する．実時間性が要求される手続きにおいて，ソフトウェア割り込みを伴う 13 個の API 使用を禁止する．一方，ソフトリアルタイム性で構わない手続きに対しては，その限りではない．本研究ではこの利用指針に従い，GPU プログラム内の実時間性が要求される手続きにおいて，ソフトウェア割り込みを除外することで，GPU プログラムの実時間性保証を図った．提案手法の動作モデルは図 2 のようになる．

4　評価方法

本研究では，GPU として NVIDIA GeForce®GTX 295，リアルタイム OS として RTAI を用いた．GPU 上で動作し実時間性を保証するプログラムは，3 章で述べたソフトウェア割り込みを伴わない CUDA ドライバ API と RTAI モジュールを用いて作成した．一方，本研究の手法と比較するために，ソフトウェア割り込みを伴う CUDA ドライバ API を用いた GPU プログラムを作成した．これらのプログラムでは，CPU のメインメモリから GPU のデバイスメモリへ計算用データの転送，転送されたデータを GPU で計算，デバイスメモリからメインメモリへ計算結果データを転送する．ある一定の周期で，これら

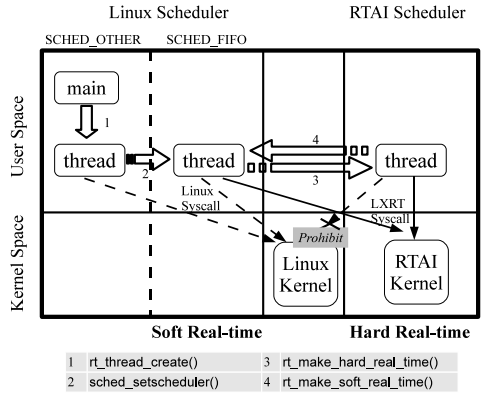

図 2: GPU プログラムの動作モデル

のタスクを周期的に実行し，実際に各タスクにかかった時間を計測する．また，バックグラウンドプロセスとして，外部装置に次の負荷をかけながら，提案手法のタスク完了時間も計測した．

- I/O Load
  スペシャルファイル/dev/zero を複写し続ける
- NIC Load
  パケットサイズ 64Byte の ICMP データをフラッディング PING コマンドで転送し続ける
- GPU Load
  実時間タスクが動作する GPU とは別の GPU において，データ転送を伴わない計算を行わせる

外部装置に負荷をかけるとハードウェア割り込みハンドラが起動する．このハンドラは RTAI カーネルで処理され，適切なタイミングで Linux カーネルに割り込みを通知する．よって外部装置に負荷をかけることでハードウェア割り込みが頻繁に起動され，実時間性が要求される GPU プログラムの実行時間に影響を及ぼす．従って，上記の負荷をかけた際のタスク完了時間を，無負荷時の完了時間と比較することで，外部装置からの割り込み処理が本研究の提案手法に与える影響を評価する．

5 結果及び考察

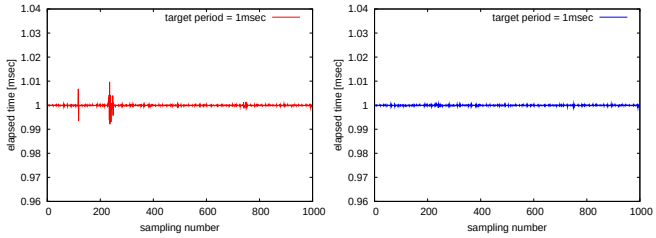

図 3: 周期 1 ミリ秒におけるタスク完了時間

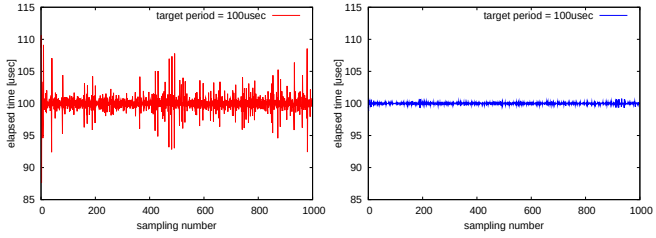

図 4: 周期 100 マイクロ秒におけるタスク完了時間

図 3 及び図 4 において，左図はソフトウェア割り込みを伴う非実時間 GPU プログラム，右図はソフトウェア割り込みを伴わない実時間性を保証する GPU プログラムにおける，タスク完了時間を表す．図 3 は設定周期が 1 ミリ秒，図 4 は設定周期が 100 マイクロ秒のときの，タスク完了時間の様子を図示している．右図における提案手法は，ソフトウェア割り込みを伴う左図と比較して，大幅にジッタ (ゆらぎ) を削減していることがわかる．つまり，右図の提案手法は，いずれもタスク完了時刻が予測可能であり設定周期内で完了できるが，一方左図における Linux システムコールを呼び出す GPU プログラムは，完了時刻の予測可能性が低く，実時間性を保証することができないことが示された．

表 1: タスク完了時間の標準偏差と Peak-to-Peak ジッタ

		SWI 有り	提案手法
設定周期	SD	1.86μsec	0.183μsec
$10^3\mu$sec	Jitter	13.9μsec	1.70μsec
設定周期	SD	1.56μsec	0.142μsec
$10^2\mu$sec	Jitter	23.0μsec	1.93μsec

表 2: 外部装置に様々な負荷をかけた際の提案手法の性能

設定周期		None Load	I/O Load	NIC Load	GPU Load
10^4	SD	1.71	11.5	60.6	1.05
[μsec]	Jitter	18.0	80.4	281	24.9
10^3	SD	0.183	3.61	13.5	0.501
[μsec]	Jitter	1.70	32.1	58.8	5.22
10^2	SD	0.143	1.49	6.62	0.540
[μsec]	Jitter	1.93	17.8	28.1	4.68

また，図 3,4 における，タスク完了時間の標準偏差 (SD) 及び Peak-to-Peak ジッタを表 1 に示す．Peak-to-Peak ジッタとは，タスク完了時間の最大値と最小値の差のことである．一般的に，システムの実時間性を評価する際は，この Peak-to-Peak ジッタが重要となる．標準偏差及びジッタの値が増加すると，タスク実行時間の予測が困難となり，実時間性を保証することができなくなる．表 1 の各設定周期において，ソフトウェア割り込みを伴う GPU プログラムと提案手法を比較すると，提案手法は標準偏差及びジッタともに約 1/10 に低減していた．従って，提案手法による標準偏差とジッタの低減は，提案手法を用いた GPU プログラムが実時間性を保証していることが確認された．

次に表 2 は，外部装置に負荷をかけた際，提案手法のタスク完了時間の標準偏差及び Peak-to-Peak ジッタをリストしている．表 2 より，I/O 及び NIC に負荷をかえると標準偏差とジッタは大幅に増加してしまった．特に NIC 負荷の増加幅は大きく，これはフラッディング・パケット生成における CPU 資源の圧迫が原因と考えられる．一方 GPU 負荷において，標準偏差及びジッタの増加はほとんど見られなかった．このことは，複数 GPU 間で適切にロードバランシングが行えていることを示している．以上の計測結果及び評価によって，本研究で提案した実時間性を保証する GPU プログラム手法の有用性が示された．

6 結論

本研究では超並列に演算処理を行うことができる GPU を，実時間システムの副処理装置として用いる手法を提案した．提案手法を用いて GPU 上で周期的にタスクを行わせ，そのタスク完了時間を計測した．計測結果より，実時間システムで要求される周期において，提案手法はタスク完了時間のジッタを低減させ，実行時間の予測可能性を示した．従って，提案手法は実時間性を保証し，実時間システムにおける本手法の有用性を確認した．