TSUBAME 共同利用 平成 27 年度 学術利用 成果報告書

GPU 上での離散イベントシミュレーション実行に向けた基礎的検討
A Fundamental Study on Parallel Discrete Event Simulations on GPUs

大原 衛
Mamoru Ohara

地方独立行政法人東京都立産業技術研究センター
Tokyo Metropolitan Industrial Technology Research Institute
http://www.iri-tokyo.jp/

本研究は、GPU を用いた並列離散イベントシミュレーション（PDES）の高性能化、高信頼化を目的とする。本課題では、PDES の高信頼化のための基礎的検討として、ソフトウェア若化の PDES への適用を試みた。ソフトウェア若化は、ソフトウェアの除去困難なバグに対する高信頼化手法である。本研究では、楽観的な PDES 手法であるタイムワープ手法にソフトウェア若化を適用する手法を提案する。著者らは以前に提案手法の信頼性解析モデルを構築した。本課題では、実際に MPI（Message Passing Interface）を用いたタイムワープシミュレータを実装し、解析モデルで用いる諸パラメータの計測を行った。

In this study, we introduced software rejuvenation into parallel discrete event simulations (PDES) as the basis for the preparation of using GPUs in PDES. Software rejuvenation is a promising technique that avoids failures caused by hard-to-fix bugs in software. We previously proposed a number of software rejuvenation schemes for time warp technique, which is one of optimistic synchronization techniques used in PDES, and developed an analytical model for evaluating system reliability. Moreover, we implemented an experimental simulator using MPI (Message Passing Interface) on TSUBAME and measured parameters of the analytical model. In this article we report the measurement results.

Keywords: parallel discrete event simulation (PDES), time warp simulation, software rejuvenation, analytical model, message passing interface (MPI)

背景と目的

近年，GPGPU（General Purpose processing on GPUs）を用いた連続系シミュレーションの高速化は広く取り組まれているが，離散イベントシミュレーション（Discrete Event Simulation: DES）の並列化を試みた事例は報告が少ない．DES の応用は待ち行列やネットワーク，論理回路シミュレーションなど幅広く，これらの分野でも並列化による性能の向上が強く求められている．本研究は，GPGPU を用いた DES の性能向上を目的とする．

多数の計算ノードを用いた大規模シミュレーションでは，これらのノード間で同期を行いながら処理を進める．このため，一部のノードの故障や通信遅延によって，シミュレーションの進行が停滞する可能性を考慮しなければならない．本課題では，このような耐故障技術を確立するための基礎的な検討を行う．

ハードウェアの故障に対してはこれまでに多くの研究が行われている．実用的にも，いくつかの MPI（Message Passing Interface）実装では，ハードウェアの故障を検出し，他のノードで代替するなどの方策が採られている．一方，ソフトウェアの問題，いわゆるバグの発現に対しては，既存の MPI 実装では対応できない．近年，除去が困難なバグへの対応法として，予防的にソフトウェアを再起動するソフトウェア若化法（software rejuvenation）が用いられ始めている．これは従来，主にエンタープライズ用途で用いられてきたが，HPC（High Performance Computing）に対しても適用する試みが為されている．申請者らも，楽観的手法に対してソフトウェア若化を適用するための基礎的な検討を行っている．本利用課題では，著者らが先行研究で構築した解析モデルの検証を主な目的として，TSUBAME 上で DES の実稼働データを取得する．TSUBAME を利用することで現実的な規模の数値例を得ることが本年度における利用の目的である．

概要

申請者らは以前に，並列 DES（Parallel DES: PDES）の楽観的同期手法であるタイムワープ手法にソフトウェア若化を導入する単純な手法を提案し，その信頼性解析モデルを構築した．本課題では，具体的な PDES プログラムを実装し，これを用いて解析モデルのいくつかのパラメータを実測した．以下ではまず，申請者らが提案するソフトウェア若化法と信頼性解析モデルについて概説する．

DES では，一連のイベントを定められた順番で処理する必要がある．タイムワープ手法では，各プロセスは明示的な同期を行わず，独自にイベントの実行を進める．このため，システム全体の視点では，イベントが正しい順序で実行されない場合がある．各プロセスは，誤った順序でイベントが実行されたことを検知すると，そのイベントを実行する以前の状態にロールバックする．

プロセス間の情報の共有は，メッセージ交換によって行う．メッセージには，イベントを実行すべき論理時刻がタイムスタンプとして付加される．また，各プロセスは，自身が実行した最後のイベントの論理時刻をローカルクロックとして保持する．プロセスは，他のプロセスからメッセージを受信すると，そのタイムスタンプとローカルクロックの値を比較する．ローカルクロック以前のメッセージが到着した場合には，イベントが誤った順番で処理された可能性があるため，ロールバックを行う．このため，効率的なシミュレーション実行のためには，各プロセスの処理の進行が大きく異ならないようにすることが重要である．

ソフトウェア若化を行う際にも，一部のプロセスの進行が妨げられることは避けなければならない．この目的で，本研究の提案手法では，システム中のすべてのプロセスが同期的に若化を行う．タイムワープ手法では，大域的な時刻であるグローバルバーチャルタイム（Global Virtual Time: GVT）を定期的に求める．このための分散アルゴリズムとして，同期的なものと非同期的なものが提案されている．提案手法では，同期的 GVT 計算アルゴリズムを仮定し，ソフトウェア若化は GVT 計算の直後に行う．

システムは時間 T ごとに GVT 計算を行う．また，k 回の GVT 計算を行うごとに連携チェックポインティングを行う．チェックポインティングが完了すると，すべてのプロセスが再起動し，チェックポイントデータを読み込んで状態を再構築する．その後，すべてのプロセスの状態再構築を待ち合わせて，シミュレーションを再開する．

GVT 計算間隔 T やソフトウェア若化周期 k が大きく，ソフトウェア若化が稀にしか行われない場合には，バグが発現しプロセスに障害が発生する可能性がある．連続する 2 回のソフトウェア若化の間にいずれかのプロセスに障害が発生したとき，障害プロセスはまず再起動を行い，次いで直前のチェックポイントデータを読み込む．タイムワープ手法では，以前の状態に復帰した障害プロセスからの古いタイムスタンプを伴ったメッセー

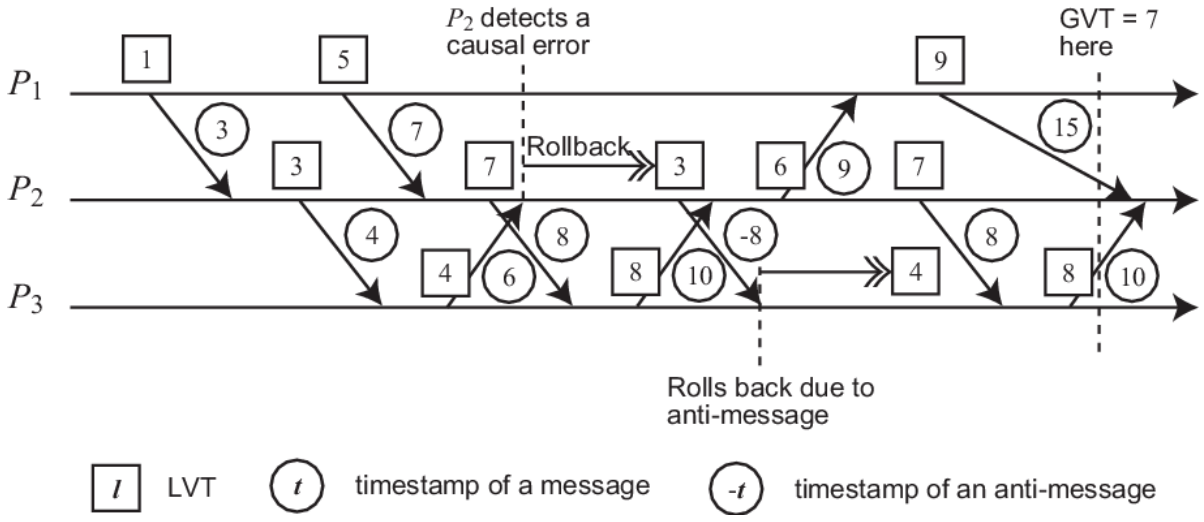

図 1 タイムワープ手法のローカルクロック（LVT），GVT の進行

ジによって，他の正常なプロセスでロールバックが発生することが想定される．これを防ぐために，障害プロセスはチェックポイントデータを読み込んだ後に，他のすべてのプロセスに特別なメッセージを送信する．このメッセージを受信したプロセスは，シミュレーションの実行を中断する．

他のプロセスから受信したすべてのメッセージを処理し状態の回復を完了すると，障害プロセスはこれを他の全プロセスに通知する．PDES の多くの応用では，すべてのプロセスが同じソフトウェアを実行する．このため，一つのプロセスに障害が発生した際には，他のプロセスにおいても障害が発生しやすい状態であることが予想される．本手法では，障害プロセスからの状態回復の通知を受信すると，ソフトウェア若化を行い，すべてのプロセスを再起動する．

提案法を用いたシステムの非可用性は，以下のように表される．ただし，解析モデルで用いる記法を表 1 にまとめた．

$$\overline{A} = \frac{H}{S},$$

$$H = \int_0^{kT} \{t_c + t_r + R(x) + t_c + t_r + t_c\} dF_N + U,$$

$$S = \int_0^{kT} \{x + t_r + t_c + R(x) + t_c + t_r + t_c\} dF_N + U,$$

$$U = \int_{kT}^{\infty} \{t_c + t_r + t_c\} dF_N,$$

$$R(x) = \frac{x}{\alpha + (1-\alpha)\beta}.$$

表 1 記法

A	可用性
S	連続する 2 回の若化の時間間隔
H	S のうち若化と障害によるオーバヘッド
U	S のうち障害が発生しない場合のオーバヘッド
X	直前のソフトウェア若化から障害発生時点までの間隔
$R(X)$	障害で失われたイベントの再実行時間
F	X の確率分布関数
N	プロセス数
F_N	N 個のプロセスに最初に発生する故障の確率分布関数
T	GVT 計算間隔
k	ソフトウェア若化周期
t_c	チェックポイント生成・読込時間
t_r	プロセス再起動時間
α	1 イベント当たりのロールバック発生率
β	1 回のロールバックによって失われる平均イベント数

結果および考察

以下の手順でソフトウェア若化を行い，諸パラメータの計測を行った．

1. ドライバスクリプトが MPI プログラムを起動する．
2. システムは k 回の GVT 計算を行う．
3. 各プロセスは，チェックポイントを 2 次記憶に書き込む．
4. MPI プログラムが終了する．
5. ドライバスクリプトは，MPI プログラムが終了すると，MPI プログラムを即座に再起動する．
6. 各プロセスが 2 次記憶からチェックポイントデータを読み込む．

手順 3，6，および 3—6 の所要時間をそれぞれ計測し，3—6 の所要時間から 3，6 の所要時間を減ずることで t_r を求めた．

計測結果を以下に示す．シミュレーションモデルは，WARPED2 とともに公開されているものから，“airport”，“traffic” を用いた．airport は，飛行場における飛行機の発着を，traffic は交差点における自動車の往来を模擬する．両モデルで，オブジェクト（空港または交差点）は，X-Y 格子状に 100×100 個を配置し，10,000 論理時間のシミュレーションを実行した．

表 2 から，各シミュレーションモデルに対して，プロセス数 N の最適値があることが分かる．N が小さいときにはαが大きく，ロールバックが発生しづらい．一方，シミュレーション時間は長く，並列効果が十分に得られていないと言える．反対にプロセス数が多すぎると，ロー

ルバックの頻度が高くなり，シミュレーションの効率が低下する．

上述のソフトウェア若化手順は MPI プロセスの起動を含むため，プロセス数が多ければ再起動時間 t_r が長くなることが予想される．ソフトウェア若化周期 k が小さいときには，この傾向が見られる．また，k が大きくなると t_r が長くなる傾向も見られる．この要因については，明らかでない．

表 2 イベントコミット率とロールバックによって失われる平均イベント数

モデル	N	α	β	実行時間（秒）
airport	2	0.992	10.68	648.8
	4	0.955	13.62	364.5
	8	0.907	13.99	23.86
	16	0.815	14.19	125.44
	32	0.929	8.82	42.78
	64	0.359	15.26	215.95
traffic	2	0.998	5.15	100.2
	4	0.980	8.90	48.17
	8	0.981	7.11	23.86
	16	0.975	6.06	11.64
	32	0.940	7.04	7.18
	64	0.623	10.67	15.36

表 3 ソフトウェア若化コスト

モデル	N	k	C (MB)	t_c (秒)	t_r (秒)
airport	8	10	8.4	1.03	1.08
		20	11.2	1.39	1.11
		30	14.5	1.85	1.44
		40	11.8	1.48	1.63
	16	10	6.7	0.91	1.20
		20	10.5	1.39	1.22
		30	9.7	1.30	1.41
		40	10.6	1.40	1.60
traffic	8	10	10.5	1.08	1.02
		20	13.0	1.35	1.16
		30	12.0	1.24	1.29
		40	13.8	1.44	1.37
	16	10	8.5	0.93	1.13
		20	10.9	1.17	1.27
		30	11.1	1.21	1.21
		40	13.5	1.45	1.25

まとめ、今後の課題

本課題では，楽観的な PDES 手法であるタイムワープ手法を用いたシミュレータを試験的に実装し，それを用いてソフトウェア若化に関する信頼性モデルの諸パラメータの実測を行った．障害の発生しない状況下でのソフトウェア若化コストなどが計測された．

シミュレータに意図的に故障を挿入することによって，障害が発生した状況における実際の挙動を観察し，計測を行うことなどが今後の課題である．