計算機アーキテクチャ　128－13
ハイパフォーマンス
コンピューティング　70－13
（１９９８．３．６）

離散事象並列シミュレーションにおける動的負荷均等化

根本 貴由　西 昌吾　三橋 純　成田 誠之助
早稲田大学　理工学部

離散事象並列シミュレーションにおいて、モデルを分割し並列計算機にマッピングした状態の善し悪しが、並列処理の効率に大きく影響する。そこで、効率的なマッピング結果を得るため、できる限り多くの情報をモデルから読み出す努力がなされてきた。しかし、並列処理に伴う仮想時刻同期処理のオーバーヘッドなどをモデルの状態から読み出すことは非常に困難であるため、実際に並列処理を行うと、各プロセッサの負荷バランスが状況により不均衡になってしまうことが多かった。本稿において、シミュレーション中にマッピング状態を適宜変更するシミュレータを並列計算機 AP1000 上に実装し、評価を行った結果、並列処理による負荷バランスの低下を改善できることが確認できた。

Dynamic Load Balancing in Parallel Discrete Event Simulation

Takayoshi Nemoto　Syougo Nishi　Jun Mitsuhashi　Seinosuke Narita
School of Science and Engineering, WASEDA University

Parallel Discrete Event Simulation is greatly influenced the efficiency by the conditions of partitioning and mapping models onto parallel computers. Much effort has been directed toward getting for better mapping results. In spite of the effort, however it is very difficult to predict the overhead involved in virtual time synchronization. In many cases, the load of each processor is not balanced in the simulation. In this paper, a dynamic load-balancing simulator is implemented and evaluated on an AP1000. It is ensured that the proper approach can improve unbalanced processing load stemming from parallel simulation.

1　はじめに

近年のネットワーク網の発展は非常にめざましいものがあり、そのバリエーションも非常に多彩である。たとえば、インターネットの爆発的普及などがその最たるものであるが、それ以外にも新電電会社の台頭による電話網の相互利用であるとか、並列計算機開発のために必要な超高速通信機構などの研究など、その例は数え上げたらきりがない。

これらのネットワークシステムは、実験などによる試行錯誤の末に構築されるものであ

る。その試行錯誤の一つの手段としてシミュレーションによるシステムの検証が行われてきた。しかしながら、シミュレーションの対象となるシステムにおいて、構成要素などの接続状況が密になり、あるいはその中でシミュレートする内容を充実させる必要が出てくるなど、複雑化の傾向にある。また、同時にネットワーク網の内部的拡張や相互接続などによる大規模化も進行しており、シミュレーションシステムに求められる計算量も、それらのことがらに比例して増加の一途をたどっている。そこで、シミュレーションの高速処理を行うために、多くの研究者がその処理性能に注目し、多様なアプリケーションを効率よく導入している、あるいは導入する方法を模索している並列計算機を利用する研究がなされている。[1]-[5]

2 離散事象シミュレーション

ネットワーク網などをシミュレートする目的は、その経路の連続的な状態よりも、むしろ経路により結ばれている交換機であるとかルータなどでのパケット、あるいはセルの状況を確認することにある。逆にいうと、交換機やルータの状態により、その間を接続する経路の状態は決定してしまうともいえよう。そこで、システムをモデル化する際には、ノード(上の例では交換機やルータ)とリンク(接続経路)であらわし、トランザクション(パケットやセルなど)がシステムの中を移動することによる各ノードの状態変化を、事象という形で繰り返し処理することで、システムの振る舞いを模倣することができる。事象は実時間とは無関係でモデル内でのみ有効な仮想的な時刻値をもつが、その値は連続的でないとびとびの値をとる。このようなシステムのシミュレーションを行うことが、離散事象システムシミュレーション(DES: Discrete Event system Simulation)と呼ばれているのはそのためである。

3 離散事象並列シミュレーション

3.1 並列処理の実現

世に出ている多くの並列アプリケーションがそうであるのと同様に、シミュレータの並列化をいかにして実現するかという問題がある。DES では、前節の説明にもあるように、シミュレート対象モデルをノードとリンクで表現している。事象処理に伴うモデルの状態変化はノード上でのみ発生するものであるので、モデルを構成しているノードを適当にプロセッサに割り振ることで、離散事象システム並列シミュレーション(PDES: Parallel Discrete Event system Simulation)を実現することができる。

3.2 マッピング

並列処理の効率を考慮すると、各プロセッサでの処理量が同程度になるようにノードを割り振る必要がある。また、異なるプロセッサで処理されているノード間リンクで発生するトランザクションの移動は、プロセッサ間通信により表現されている。しかし、通常の並列計算機において、プロセッサ間通信性能は各プロセッサの計算能力と比較して非常に低速である。そこで、並列シミュレーションにおいて、通信量をできる限り少なくなるようにノードをマッピングすることが有効であることも容易に想像できるであろう。

3.3 仮想時刻同期処理

PDES では、各プロセッサが割り当てられたノードで構成されるサブモデルの処理を行うため、それぞれのプロセッサが仮想時刻をもつことになる。このローカルな仮想時刻(LVT: Local Virtual Time)は各プロセッサにより異なるため、それによる事象処理順序の逆転が起こるのを防ぐために仮想時刻同期と呼ばれている処理を行う必要がある。この処

理はシミュレーションの並列化に伴うオーバーヘッドであり、その善し悪しが並列処理の効率を大きく左右するものである。そのため、これまでに多くの手法が提案され評価が行われたきた。その代表的なものに、処理順序の逆転が発生しないように処理を制限する保守的手法[3]として、Null Message 法[2]、問い合せ型 Null Message 法、同期法、また、処理順序の逆転が発生した場合に処理をロールバックする楽観的手法[1][3]として、Time Warp 法があげられる。本研究では、後に説明する動的負荷均等化において楽観的手法が処理をロールバックできる点で都合がよいと考えたため、まず Time Warp 法をベースに検証することにした。

4 Time Warp 法

4.1 基本的処理手順

Time Warp 法は、各プロセッサが時刻の制限なしに処理を進め、事象処理順序の逆転が発生した場合(つまり、他のプロセッサからある時刻の事象を受け取った時点で、それよりも後の時刻値をもつ事象を処理してしまっていた場合)に処理をロールバックすることで、最終的に正常な処理順序になることを保証する手法である。したがって、後にロールバックする可能性があるため、事象処理によるサブモデルの状態の変更状況を保持しておかなければならない。このロールバックのための情報はシミュレーションが進行するにつれて増大していく。計算機に搭載されている実メモリ量は有限であるので、実際には完全に時刻の制限なしに処理を進行することは不可能である。今回導入したシミュレータでは、ある一定間隔(メモリが不足する状況におちいることのない適当な間隔)で全プロセッサが同期し、ロールバックすることがない時刻として全体的仮想時刻(GVT: Global Virtual Time)を更新、その時刻までの情報を破棄することでこの問題に対処している。図 1に Time Warp 法のアルゴリズムを示す。

```
while (GVT < Endtime) {
  Process Event;
  Handle Message;
  if (contradiction is detected) {
    rollback;
  }
  if (GVT is to be updated) {
    barrier syncronization;
    GVT = Min(LVTs of All PEs);
  }
}
```

図 1 Time Warp 法のアルゴリズム

4.2 仮想時刻の同調による効果

前節において、Time Warp 法の基本的アルゴリズムを説明したが、これまでの研究において、各プロセッサの処理をまったく制限せず、機械的に GVT を更新するだけの状態で処理を進行させると、それぞれのプロセッサの仮想時刻が大きく異なってしまうことがわかっている。これはマッピングの善し悪しによる場合 (PE 間通信量の誤算、仮想時刻同期によるオーバーヘッドの影響など) もあるのだが、いずれにせよ Time Warp 法のオーバーヘッドの一つであるロールバック処理の発生する確率が増大してしまう。そこで、各プロセッサの仮想時刻が大きく異なることのないように同期間隔に行われる事象処理数に緩い制限を設けることで、ロールバック処理が発生する確率を大幅に減らすことができることがわかった。[5]

5 動的負荷均等化

5.1 待機プロセッサの発生

仮想時刻の同調を行うと、各プロセッサにおいて同期を行うまでの事象処理数が異なることになるが、この場合、処理待ちをしているプロセッサが出てくる。このことは、各プロセッサでの事象処理量と通信量に加えて、

仮想時刻同期処理をも考慮した上での処理バランスがよい状態ではないことを示している。それに対処するためには、仮想時刻同期処理によるオーバーヘッドを考慮した上でシミュレーション前のマッピングを行うことが理想である。しかし、事象処理量と通信量はシミュレーション以前にある程度予測がつくが、仮想時刻同期処理に伴って発生するオーバーヘッドの予測を事前に行うのは非常に困難である。よって、シミュレーション前には、これまで通り、予測される事象処理量と通信量をもとに静的なマッピングを行うことにする。その上でシミュレーションを開始し、その処理中に並列計算機での処理状況を監視することで、仮想時刻同期処理に起因する各プロセッサでの処理バランスの悪化を補正するべく、動的に負荷を均等化する方法を考えることにした。

各プロセッサが他のプロセッサの処理状況を常に監視することも不可能ではないが、そのための通信を多数行わなければならないため、そのオーバーヘッドを考慮するとあまり望ましい方法とはいえない。そこで、今回はGVT を更新するタイミングに全プロセッサの状態を交換し、その情報を元に、処理を他のプロセッサに依頼すべきかどうか、そうであればどのノードをどのプロセッサに処理依頼するかなどを決定することにした。

5.2 動的負荷均等化の利点

動的負荷均等化によるメリットとして、以下の点があげられる。

1) 仮想時刻同期処理を考慮した負荷分散
2) 時間によって各ノードでの処理負荷が変動するようなモデルへの対応

まず、1)によって、各プロセッサでの処理進行度が一様になることから、仮想時刻の同調によってなかば強制的に押さえるかたちであったロールバックの発生確率を自然に低減させることが可能となる。また、ロールバック処理の発生確率が低減するということは、ロールバックの深さが浅くなるということと同等であるため、過去のロールバック情報をより多く破棄できるということにつながる。そのため、実際には不必要であった(つまり、そのロールバック情報でロールバック処理が行われることはなかった)ロールバック情報数が減少し、その結果、メモリの利用効率が向上することも予想される。

次に、2)の状況はたとえば、あるノードにおいてある時間帯では非常に多くの事象処理が行われるが、それ以外の時間帯では処理量が極端に低下するような構成のモデルである場合、あるノードの能力がある時間帯は非常に低くなるような構成のモデルである場合などに起こり得るものである。このような場合、はじめに各プロセッサに割り当てられたマッピング状態が、仮にシミュレーション初期にバランスしているものであるとしても、処理負荷が変動してしまった後では、各プロセッサでの処理量が初期のものと異なってしまうため、処理バランスがくずれ、効率のよい並列処理を行うことができなくなってしまう。しかし、動的にマッピング状況を変更することが可能であるならば、処理負荷が変動した後でも各プロセッサの処理量がバランスした最適な状態に回復させることができる。

5.3 動的負荷均等化の欠点

前節では、動的負荷均等化によってもたらされるメリットについて述べてきたが、逆に処理効率が低下するような状況も想定することができる。5.1 でも少々触れられたことであるが、各プロセッサの負荷状況を知るため、他のプロセッサと通信を行う必要がある。この情報交換のための通信に要する時間は、動的負荷分散を行う必要がないような状況下ではまったく無駄な時間ということになる。このような場合、並列シミュレータの処理速度

を低下させるだけということになる。

6 評価

6.1 評価環境

これまで述べてきた動的負荷均等化に関する評価を行うため、実際にそれを行う場合と行わない場合との比較実験を行った。この評価を行う実計算機として、富士通研究所のAP1000 を 4×4=16 プロセッサ構成で用いた。

6.2 評価モデル

評価対象モデルとして、図 2 のような 1024 個のノードがランダム結合したモデルを 100 種類用意した。

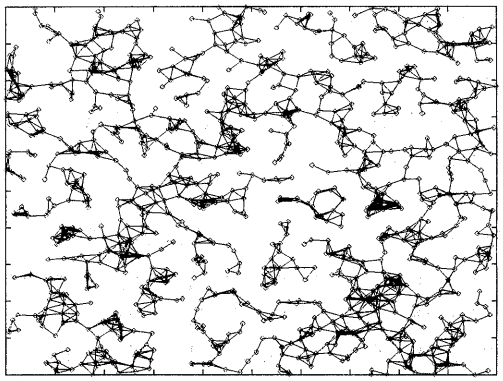

図 2 評価モデルの一例

6.3 静的マッピング

初期マッピング状態は、モデルを、図 2 のように平面に配置した状態をもとに行ったものを用いた。このマッピング方法は比較的高速に結果を得ることが可能であるにもかかわらず、プロセッサ間の通信量を大幅に押さえることができるなど、もともと並列処理の効率が高いマッピング結果を得られることが、これまでの研究により示されている。

6.4 評価結果

仮想時刻と処理時間の関係を表 1 に、そのグラフを図 3に示す。これらは多数のモデルを評価したものの一つの例である。

また、実際に並列処理を行った際の、処理時間、事象処理数、ロールバック回数、待ち時間などの例を表 2にまとめる。

表 2 並列処理の状況

	均等化	均等化なし
処理時間 [s]	110.204	116.386
送信事象数 [回]	1626263	1707572
アンチメッセージ数 [回]	895570	973404
事象処理数 [回]	1585003	1577510
ロールバック回数 [回]	1846521	1984037
同期回数 [回]	2160	2241
最大待ち時間の和 [s]	26.962	33.637
平均待ち時間の和 [s]	13.506	16.604
ノードの処理依頼数 [ノード]	99	0

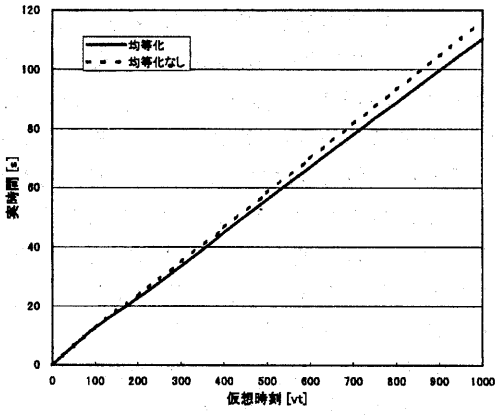

図 3 処理時間の比較

表 1 仮想時刻と処理時間の関係

仮想時刻 [vt]	0	100	200	300	400	500	600	700	800	900	1000
均等化 [s]	0.000	12.744	22.752	33.655	44.928	55.997	66.833	77.888	88.772	99.666	110.204
均等化なし [s]	0.000	12.777	23.610	35.093	46.579	58.182	69.806	81.501	93.129	104.533	116.386
差 [s]	0.000	0.033	0.858	1.439	1.652	2.185	2.973	3.613	4.357	4.867	6.182

7 考察

結果について、全てのモデルにおいて同様の結果が得られたので、例示した結果のみについて考察を加える。

表 1および図 3を見ると、約 100[vt]までは同程度の実処理時間であるが、それ以降は負荷均等化の効果により傾きが小さくなる結果が得られた。最終的な処理時間において5.3[%]ほど処理時間の短縮が見られる。

その理由を考えるために表 2に目を移す。送信事象数とは、プロセッサ間通信で送られた事象数のこと、アンチメッセージとは、上のように送信された事象を取り消すためのメッセージである。アンチメッセージはロールバックが行われた場合に発生する可能性がある。表を見ると、アンチメッセージが 8.0[%]ほど減少している。これは 5.2 の 1)で予想されていることに合致する結果となった。送信事象数が少ないのは、アンチされる送信事象の割合が減少していることと同値である。さらに、ロールバック回数を比較すると、約6.9[%]その回数が減少している。この結果も、上と同様の傾向を示す結果であるといえる。

次に、待ち時間の比較を行う。最大待ち時間とは、同期毎に最大待ちを行っていたプロセッサの待ち時間の総和である。その時間が6.7[s]ほど短縮されている。また、平均待ち時間とは、全プロセッサの総待ち時間の平均であるが、この時間も 3.1[s]短縮されていることがわかる。短縮された処理時間が 6.2[s]であるので、ロールバック処理のオーバーヘッドは時間にして 3.1[s]ほど低減できたことになる。また、動的負荷均等化による、ノード割り当ての変更数は 99 であった。これは、ノード総数 1024 の 9.7[%]に相当する。

今回のシミュレータは、時間帯によって各ノードの事象発生量や能力が増減するモデルを記述できるものではなかったが、今後、さまざまな状況をモデル化できるような汎用的なシミュレータを導入した場合、今回現れた結果よりも多くの性能向上が望めるであろう。

8 まとめ

本研究では、シミュレーション中の各プロセッサに加わる負荷を均等化することで並列処理の効率向上につながることを確認した。今後は Time Warp 法以外の保守的手法において、動的負荷均等化が有効に機能するかについて継続して研究を進めていく予定である。

謝辞

最後に、この研究を行うにあたり、すばらしい研究開発環境を提供していただいた富士通研究所、ならびに早稲田大学村岡研究室の皆様方に大変感謝いたします。

参考文献

[1] David R.Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems, Vol.7, No.3, pp.404-425, July 1985.

[2] J.Misra, "Distributed Discrete-Event Simulation", ACM Computing Surveys, Vol.18, No.1, pp.39-65, March 1986.

[3] Richard M.Fujimoto, "Parallel Discrete Event Simulation", Communication of the ACM, Vol.33, No.10, pp.30-53, October 1990.

[4] Richard M.Fujimoto, "Parallel and Distributed Discrete Event Simulation: Algorithms and Applications", Proceedings of the 1993 Winter Simulation Conference, pp.106-114, 1993.

[5] 根本, 高井, 成田, "タイムワープ法を用いた離散事象並列シミュレータにおける仮想時刻の同調", The Fifth Parallel Computing Workshop, P1-D, March 1996.