計算機アーキテクチャ　105－1
ハイパフォーマンス
コンピューティング　50－1
（1994．3．10）

待ち行列並列シミュレーションのマッピング手法

高井 峰生　根本 貴由　成田 誠之助

早稲田大学理工学部

大規模な待ち行列シミュレーションの並列処理では、シミュレートする待ち行列モデルを複数のサブモデルに分割・マッピングすることが負荷分散に相当する。本稿では、シミュレーション時間に比べて十分短時間でマッピングできる繰り返し改良法を用いた。また、デッドロック回避問題に保守的手法の中で代表的なヌルメッセージ法を用いた。ヌルメッセージのオーバーヘッドは、モデル内のループと分岐で特に問題となる。より実行時に近い通信コストを考慮してマッピングするために、繰り返し改良法の通信コスト指標であるゲインの計算にヌルメッセージのオーバーヘッドを加えた。この手法によるマッピングの効果を AP1000 を用いて評価した。

A Mapping Method for Parallel Queuing Network Simulation

Mineo Takai　Takayoshi Nemoto　Seinosuke Narita

School of Science and Engineering, Waseda University

In parallel queuing network simulation, partitioning and mapping a simulated model corresponds to load balancing. This paper employs the iterative improvement method which can map the model in shorter time than the simulate time, and the null-message method for the deadlock avoidance. The null-message method degrades the simulation performance mostly at loops and branches in the model. For more exact estimation of the communication cost in running time, these null-message overhead is considered when the gain is computed on the basis of the iterative improvement method. The efficiency of this method was evaluated on an AP1000.

1 はじめに

従来から待ち行列ネットワークモデルで表されるシステム (以後、待ち行列システムと略す) は数多くある。しかし、近年になって少量多品種の生産ライン [13] や ATM(Asynchronous Transfer Mode) の発展 [11] など、以前よりも大規模な待ち行列システムを検討・評価する必要性は高まる傾向にある。

これらのシステムを評価する場合、待ち行列理論の適用などが考えられるが、モデル化する際の制約条件が多いことなどの理由から、実際にはシミュレーションを利用することが多い。また、他の解析手段による結果の検証にシミュレーションを使用することも多い。

待ち行列システムのシミュレーションは、大きく長期的なものと短期的なものとに分けることが出来る。短期的なシミュレーションに比べて時間のかかる長期的シミュレーションの場合、実際に対象システムをモデル化する際には、モデルを単純化、簡素化する意味でシステムの挙動を確率的に捉えることが多くある。モデル内に確率的要素がある場合、実際のシミュレートの際には (疑似) 乱数を用いることになるが、乱数を用いると統計的な面から一回の試行ではシミュレーション結果を評価することが出来ない。そこで、同じモデルであっても乱数系列を変化させてシミュレーションを複数回実行する必要がある [14] [1]。

このような繰り返し実行が要求される中で、一回の実行が長時間にわたる大規模なシステムになれば、全体として莫大な実行時間が要求される。そのため、大規模待ち行列シミュレーションの効率化は大変重要な課題である。

以上のような背景から、多くの待ち行列並列シミュレーションの研究 [4][5] がなされている。しかし、他の多くの並列アプリケーションと同様に負荷分散の問題がある。待ち行列並列シミュレーションの負荷分散は、シミュレートする待ち行列モデルを複数のサブモデルに分割することに相当し、一般のネットワークパーティショニング手法を適用することが出来る。

Nandy and Loucks [8] は、待ち行列シミュレーションと同様に離散事象型シミュレーションの一種である論理回路シミュレーションにおいて、Kirnighan and Lin [7] に代表される繰り返し改良法 (Iterative Improvement Algorithm) を用い、性能向上を得ている。しかし、期待通りの並列効果が出ないのは、離散事象型並列シミュレーションに特有なデッドロック回避と負荷分散を別々に扱っているためと思われる。

本稿では、待ち行列モデルを PE 間結合が平等でない並列計算機へマッピングする問題において、繰り返し改良法でのグラフ分割の指標となるゲインの計算にデッドロック回避法のオーバーヘッドを考慮することによって、より効率的なシミュレーションを行なうことを目的とする。また、この手法の効果を実並列計算機 AP1000 を用いて評価する。

[1] 例えば、統計結果の精度を一桁上げるためには M/M/1 モデルでも 100 倍以上の試行を行なわなければならない。

2 待ち行列並列シミュレーション

2.1 待ち行列シミュレーションについて

待ち行列シミュレーションは、要素の到着、サービス開始、終了など、対象システム内に起こる事象によって進められるので、離散事象型シミュレーションの一種である。離散事象型シミュレーションは、

- 現在シミュレートしている時刻を表す仮想時刻
- 将来起こるであろう事象を登録する事象カレンダ
- 仮想時刻におけるシステムの状態を表す状態変数
- 統計処理をするために必要な統計変数

を用いて図 1のように進める。

```
/* 待ち行列シミュレーションの進行方法 */
仮想時刻 = 開始時刻;
事象カレンダに初期事象を登録する;
while (仮想時刻 < 終了時刻)
        事象カレンダから先頭事象を取り出す;
        仮想時刻 = 先頭事象の生起時刻;
        状態変数、統計変数を更新する;
        新たな事象を事象カレンダに登録する;
        事象カレンダを時刻順に並べ換える;
endwhile
```

図 1: 待ち行列シミュレーションの進行方法

新たに生じた事象は、元の事象より生起時刻が小さくなることはない。そこで、事象カレンダに登録されている事象を生起時刻の小さい順に処理することによって、仮想時刻の非単調減少を保証している。

2.2 待ち行列シミュレーションの並列処理

待ち行列シミュレーションの並列処理は、逐次処理の場合に用いた、仮想時刻、事象カレンダ、状態変数、統計変数に加えて、事象処理の正当性の判断基準となる保証時刻を用い、各ブロック[2]が図2のように処理を進める。

```
/* 待ち行列並列シミュレーションの進行方法 */
保証時刻 = 開始時刻;
仮想時刻 = 開始時刻;
事象カレンダに初期事象を登録する;
while (仮想時刻<終了時刻 or 保証時刻<終了時刻)
    while (事象カレンダの先頭事象時刻<保証時刻)
        事象カレンダから先頭事象を取り出す;
        仮想時刻 = 先頭事象の生起時刻;
        状態変数、統計変数を更新する;
        if (新たな事象が他のブロックで処理される事象)
            事象をメッセージにして送る;
        else
            事象を事象カレンダに登録する;
        事象カレンダを時刻順に並べ換える;
    endwhile
    送られてきた事象を事象カレンダに登録する;
    保証時刻を更新する;
endwhile
```

図 2: 待ち行列並列シミュレーションの進行方法

このように、各ブロックが事象をメッセージとして他のブロックに送り、割り当てられたサブモデルのシミュレートを非同期で進めることにより、高い並列効果を得ることが出来る。

2.3 デッドロック回避問題

2.3.1 保守的手法と楽観的手法について

2.2の図2をそのまま実行すると、シミュレーションモデルの形、マッピングの仕方によっては、各ブロックがお互いに入力メッセージを待って保証時刻が更新できず、処理が止まってしまう (デッドロック状態) 可能性がある。

デッドロック状態の例を図3に示す。図中、"gt: " は

[2]本稿ではグラフパーティショニング問題と用語を統一するために、PEをブロック、待ち行列をノード、待ち行列間の要素の流れをエッジと呼ぶことにする。

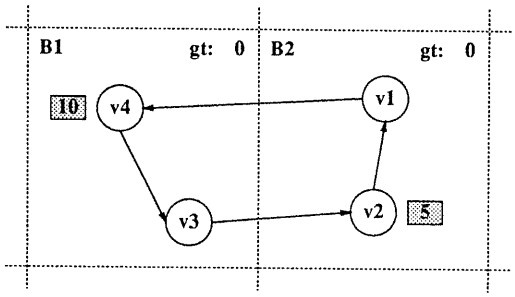


図 3: デッドロック状態の例

当該ブロックの保証時刻、網かけの四角は事象 (数字は生起時刻) を表す。ここでは、二つのブロックがお互いのブロックへ送る事象がなく、保証時刻が更新できないため、ブロック内の事象が処理できない状態にある。

デッドロックを回避する手法はさまざまなものが提案されているが、事象処理順序の矛盾を容認するか否かで大きく保守的手法と楽観的手法の二つに分類することが出来る [4]。

保守的手法は、何らかの方法で他のブロックの仮想時刻を知り、それによって事象処理に矛盾が起きないようにする。これに対し、楽観的手法は、事象処理の正当性が保証されなくても事象処理を行ない、矛盾が生じた場合は正しく処理されていた時刻から処理をやり直す (ロールバック)。

保守的手法の問題点としては、保証時刻を更新するための通信、同期が大きなオーバーヘッドになることが挙げられる。また、事象処理の正当性が保証されるまで実行を待たなければならないことも、処理効率を悪化させる原因となる。

楽観的手法の問題点は、保証時刻よりも後の生起時刻を持つ事象の処理ではロールバックが起こる可能性があるため、状態変化の履歴を保存しなければならないことである。さらに、ロールバックが生じた時に他のブロックに対して不当なメッセージを送出していた場合、メッセージの不当を知らせるアンチメッセージを送出しなければならず、ロールバックの連続発生が起こる可能性がある。

楽観的手法は一事象による状態の変化が少なく、状態変化の履歴を保存するオーバーヘッドが小さければ有効である。しかし、待ち行列シミュレーションでは、一事象ごとに状態変数の変化、統計変数の変化、発生させた乱数系列などを保存する必要があるため、楽観的手法は適さないと考えられる。

そこで、本稿では保守的手法を適用し、保守的手法の中でももっとも一般的なヌルメッセージ法 [1] を用いた。

2.3.2 ヌルメッセージ法について

ヌルメッセージ法は、モデル内に分岐などがある場合、本来のメッセージ送出先以外にも内容が空のメッセージ (ヌルメッセージ) を送出することによって保証時刻を更新し、デッドロックを防ぐものである。

ヌルメッセージは次の二つの場合に送出される。

1. 保証時刻が更新できず、事象処理が他のブロックのメッセージ待ちになった場合、全てのメッセージ送出先ブロックに対してヌルメッセージを送出
2. 複数のブロックに対してメッセージ送出可能性のあるブロックが、一つのブロックに対して通常メッセージを送出する場合、他のブロックに対してヌルメッセージを送出

以下に、それぞれのヌルメッセージ送出の例を示す。

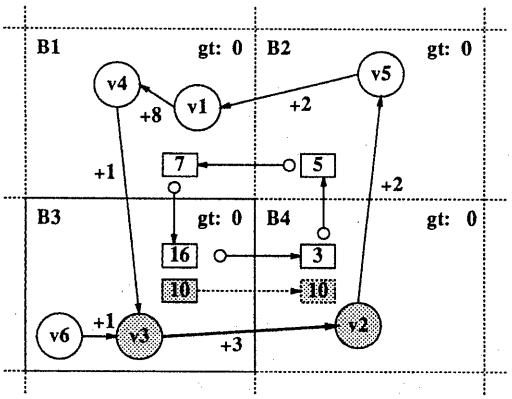


図 4: ヌルメッセージ送出 1の例

図 4中、エッジの側の数字はノード間の最小時刻遅延をあらわす。このモデルで、ノード v_3の事象によって v_2 に生起時刻 10 の事象が生じたとする。しかし、ブロック B_3は B_1から生起時刻が 10 以下のメッセージが送出される可能性があり、処理順序の正当性が確認できないため、B_4にメッセージとして事象を送出することができない。このとき、各ブロックがブロック内の最小時刻遅延を加算したヌルメッセージを送ると、B_3は B_1から 10 以下のメッセージが送出されないことが分かり、B_4 に対するメッセージを送出することが出来る。

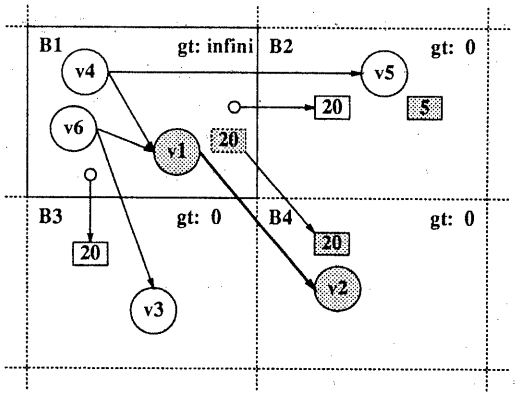


図 5: ヌルメッセージ送出 2の例

また、図 5のモデルでは、ブロック B_2内のノード v_5 に生起時刻 5 の事象が登録されていても、B_1から生起時刻が 5 以下の事象を含むメッセージが送出される可能性があるため、事象処理をすることが出来ない。この時、B_1が B_4へ事象を含むメッセージを送ると同時に B_2, B_3へもヌルメッセージを送出することにより、B_2 は事象処理を行なうことができる。

2.4 負荷分散問題

負荷分散問題は、待ち行列並列シミュレーションに限らず、ほとんどの並列アプリケーションで問題となる。

待ち行列並列シミュレーションは、各ブロックがメッセージを送受信しながら割り当てられたサブモデルをシミュレートすることによって進められる。よって、シミュレートするモデルを分割し、マッピングすることが負荷を分散することに相当する。

N個のノードで構成されるモデルを K個のブロックに分割・マッピングすることを考える。一定時間内に i番目のノード v_iを通過する要素の数を $S(v_i)$、同様にノード v_iから v_jへのエッジ e_{ij}を通過する要素数を $W(e_{ij})$ で表す。$S(v_i), W(e_{ij})$ をそれぞれノード v_iの大きさ、エッジ e_{ij}の重みと呼ぶ。また、p番目のブロック B_pに属するノードの大きさの和を $S(B_p)$、B_pと B_qとの距離を d_{pq}で表す。

ノードの大きさは当該ノードをシミュレートするのに必要な処理コストに比例し、エッジの重みはエッジの両端ノードが別々のブロックに割り当てられた時に発生するメッセージの数である。

この時負荷分散は、各ブロックのシミュレートに要す

る処理コストを均等化し、かつメッセージの発生量を最小化することである。そこで、以下のように定式化できる。

制約条件:

$$|S(B_p) - S/K| < \epsilon \tag{1}$$

目的関数:

$$\min \sum_{v_i \in B_p} \sum_{v_j \in B_q} (W(e_{ij}) \times d_{pq}) \tag{2}$$

ただし、$p, q = 1, \ldots, K$、Sはノードの大きさの総和、ϵ は Sに比べて充分小さい値とする。

負荷分散はシミュレート時間を減少する目的で行なわれるものであり、負荷分散をすること自体に大きな時間がかかっては意味がない。目的関数 (2) の最適解を求めることは NP 型問題であるため、近似解を求める手法を用いる必要がある。また、近似解を求める手法の中でも Simulated Annealing [6] のような時間のかかる手法は目的に合わない。シミュレーションの実行時間に対して充分小さい実行時間で近似解が求められ、モデルの諸特性を考慮できる手法として、Kernighan and Lin [7]、Fiduccia and Mattheyses [3] などの繰り返し改良法がある。本稿では繰り返し改良法を用いて負荷分散を行なう。

3 マッピング手法

3.1 繰り返し改良法について

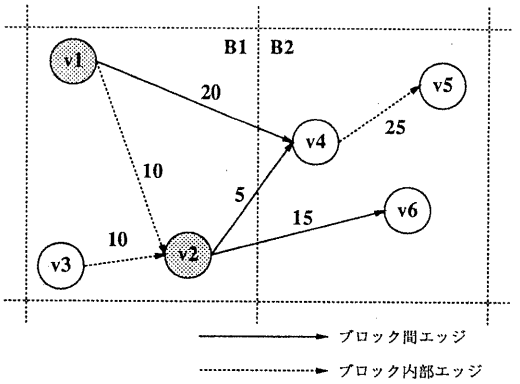
繰り返し改良法では、当該ノードが他のブロックへ移動した場合に変化する通信量をゲイン $G(v)$ と呼ぶ。$v_x \in B_1$が B_2へ移動する時のゲイン $G^{12}(v_x)$ は式 (3) で表される。

$$\begin{aligned} G^{12}(v_x) &= -\sum_{e_{|xi|}} (W(e_{|xi|}) \times d_{12}) \\ &\quad + \sum_{e_{|xj|}} (W(e_{|xj|}) \times d_{12}) \\ &\quad + \sum_{e_{|xk|}} (W(e_{|xk|}) \times (d_{1n} - d_{2n})) \end{aligned} \tag{3}$$

$$v_i \in B_1, x \neq i, v_j \in B_2, v_k \in B_n, B_n = \overline{B_1 \cup B_2}$$

ただし、エッジ e_{ij}, e_{ji}をまとめて $e_{|ij|}$で表す。

式 (3) の右辺第一項はブロック内部のエッジがブロック間のエッジになるために増加するメッセージの量、第二項は逆にブロック間エッジが内部エッジになるために減少するメッセージ量、第三項はブロック間距離の変化によって増減するメッセージ量である。



$$\begin{aligned} G^{12}(v_1) &= -W(e_{12}) \times 1 + W(e_{14}) \times 1 \\ &= +10 \\ G^{12}(v_2) &= -W(e_{12}) \times 1 - W(e_{32}) \times 1 \\ &\quad + W(e_{24}) \times 1 + W(e_{26}) \times 1 \\ &= 0 \\ G^{12}(v_3) &= -10, \; G^{21}(v_4) = 0 \\ G^{21}(v_5) &= -25, \; G^{21}(v_6) = +15 \end{aligned}$$

図 6: ゲインの計算例

ゲインの計算例を図6に示す。図6中、エッジの側の数字はエッジの重みを表す。

繰り返し改良法にもさまざまなものがあるが、ここではノードの大きさが不均一の場合も考慮した [3] の二分割法を基本とする。ブロック B_pに属するノードが登録されているリストを L_pとし、$S(L_p) = S(B_p)$ とすると基本アルゴリズムは図7のようになる。図7中、移動するノードのゲインが負になっても計算を終了しないのは、局所的な解 (local minima) に陥らないためである。

図 7のアルゴリズムを用いてモデルを 2^m個のブロックに分割・マッピングする場合、大きく分けて次の二つが考えられる。

1. 一度に 2^m個のブロックに分割する方法 [8] [9]
2. 図 7の分割を再帰的に m 回繰り返す方法 [2]

方法 2は直前の分割指標 (ブロック内部エッジの重み和最大、ブロック間エッジの重み和最小) との競合が生じるため、一般に方法 1の方が分割結果は良い。しかし、方法 1は現在属しているブロック以外の $2^m - 1$ 個のブ

/* 二分割法の基本アルゴリズム */
$S(L_1) \simeq S(L_2)$ になるようにノードを分配する;
各ノードのゲインを計算する;
L_1, L_2をゲインの大きい順に並べ換える;
$L_{from} = max\{S(L_1), S(L_2)\}$;
$L_{to} = min\{S(L_1), S(L_2)\}$;
$n = N; i = 1; G(v_0) = 0$;
while ($n \neq 0$)
  while (L_{from}に動けるノードがある)
    L_{from}の先頭ノード v_x^iを取り出す;
    v_x^iを L_{to}に登録し固定する;
    L_{from}, L_{to}内ノードのゲインを再計算する;
    L_{from}, L_{to}をゲインの大きい順に並べ換える;
    $L_{from} = max\{S(L_1), S(L_2)\}$;
    $L_{to} = min\{S(L_1), S(L_2)\}$;
    $i = i + 1$;
  endwhile
  $\sum_{i=0}^{n} G(v_x^i)$ が最大となるような n を求める;
  v_x^{n+1}以降のノードを他方のリストに登録し直す;
endwhile

図 7: 二分割法の基本アルゴリズム

ロックに対するリストを作る必要があり、ブロック数の二乗に比例して計算量が増加する。また、ブロック間結合が平等でない、すなわち分割の際にブロック間の距離を考慮しなければならない場合、方法1では距離の遠いブロック間でのノード移動が頻繁に起こり[3]、何らかの制限を加えなければ計算が収束しない可能性がある。そこで、本稿では方法2の再帰的方法を適用した。

3.2 ヌルメッセージを考慮したゲイン計算

3.2.1 ヌルメッセージの通信コスト

実際にシミュレーションを行なう場合、一事象に含まれる情報は少ないので、通常メッセージの転送時間そのものは非常に小さく、メッセージ送受信の手続きが通信の主なオーバーヘッドになる。通常メッセージとヌルメッセージの送受信に必要なコストは同じであるので、メッセージ発生量の増減を表すゲインの計算にもヌルメッセージを考慮する必要がある。

[3]距離を考慮した場合、ゲインの大きさが考慮しない場合の距離倍になるため、遠いブロック間での移動はゲインが大きくなる

3.2.2 ループに対する考慮

2.3.2の図 1のような場合、各ブロックの保証時刻はメッセージがループを一周してはじめて増加する。また、その時の保証時刻の時刻増加は、少なくともループを構成するエッジの最小時刻遅延の和となる。よって、ループ Oに属するエッジ e_{ij}を含むブロックの仮想時刻が T増加するためには、最高で

$$W'(e_{ij}) = \frac{T}{\sum_{e_{xy} \in O} L(e_{xy})} \tag{4}$$

のメッセージが必要になる。ただし、$L(e_{xy})$ は e_{xy}の最小時刻遅延、Tはシミュレート時間を表す。このとき保証時刻の増加は、メッセージが通常メッセージであるか、ヌルメッセージであるかには依存しない。また、e_{ij}が複数のループに属していた場合、それぞれのループについて式 (4) を計算し、その最大値が必要なメッセージ数になる。

以上から、式 (4) を使うことにより、ループに属するエッジの重みにヌルメッセージを含めることができ、より実行時に近いメッセージ発生量を考慮することができる。

3.2.3 分岐に対する考慮

ブロック B_pに属するノード v_xについて、v_xが関わるブロック間エッジの重みの総和を $C^p(v_x)$ で表すと、

$$C^p(v_x) = \sum_{e_{|xy|}} (W(e_{|xy|}) \times d_{pq}) \tag{5}$$

$$v_y \in B_q, B_q = \overline{B_p}$$

この時、式 (3) のゲイン $G^{12}(v_x)$ は式 (6) で表される。

$$G^{12}(v_x) = C^1(v_x) - C^2(v_x) \tag{6}$$

しかし、(3) や (6) の計算方法ではヌルメッセージの通信コストを考慮していない。

2.3.2の 2のように、あるブロックから複数のエッジが出力されている時を考える。あるブロックに対してメッセージを出力する際、他のブロックに対して保証時刻を更新できればヌルメッセージを送出することになる。他の全てのブロックにヌルメッセージを送出すると仮定すると、ノードの通信コストは、

$$C'^p(v_x) = \sum_{e_{xy}} W(e_{xy}) \times \sum_{v_y \in B_q} d_{pq} + \sum_{e_{zx}} (\sum_{e_{zo}} W(e_{zo}) \times d_{qp}) \tag{7}$$

$$v_y, v_z \in B_q, B_q = \overline{B_p}, v_o \in B_r, B_r = \overline{B_q}$$

となる。

ここで、式 (7) の右辺第一項は当該ノードの出力エッジに関する項であり、第二項は入力エッジに関する項である。

式 (7) をゲインの計算式 (6) に用いることにより、分岐に対するヌルメッセージも考慮することができる。

4 手法の評価

4.1 シミュレート対象モデル

本稿では次の二つのモデル (ノード数 256 個) を評価に用いた。

1. 各ノードがループをなすように結合されたモデル
2. 各ノードが二分木に結合されたモデル

各モデルを図 8に示す。図中、網かけのノードはソースノードである。各待ち行列は FCFS(First Come First Service)、サーバのサービス時間は定数分布、要素はソースからポアソン分布で到着する。

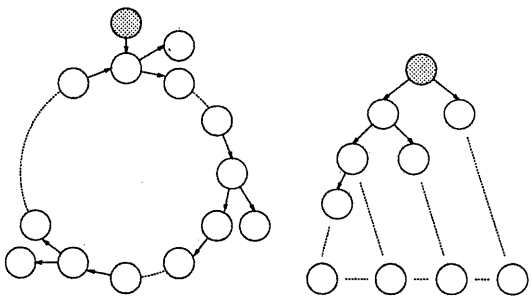


図 8: シミュレート対象モデル

モデル内の分岐は、全て乱数を用いた確率的分岐とした。これにより、要素の流量はシミュレーション実行前に予測することができるので、ノードの大きさ、エッジの重みには要素の流量からの予測値を用いた。

また、待ち行列シミュレータは、富士通研究所の高並列計算機 AP1000 上に待ち行列モデルとサブモデルのマップを入力とする専用シミュレータを開発した。

4.2 シミュレート結果

繰り返し改良法では、マッピングの初期化の際に乱数を用いるが、この乱数がマッピングの結果に大きな影響を与える。そこで、乱数の種を 3 回変化させて 16 のブロックにマッピングした際のシミュレート時間を測定した。これを表 1,2に示す。ただし、ループモデルについてはループに関して、二分木モデルについては分岐に関してヌルメッセージを考慮したものと、考慮しないものについてそれぞれ比較した。同じ乱数の種を用いた場合、ヌルメッセージを考慮したマッピングの方がしない場合に比べて 4%から 35%実行時間が短縮されていることが分かる。

乱数の種	1	2	3
null message 考慮なし	746.59	665.60	608.00
null message 考慮あり	485.68	596.06	495.67

表 1: ループモデルの実行時間 [秒](256node,16block)

乱数の種	1	2	3
null message 考慮なし	70.01	61.53	65.92
null message 考慮あり	68.14	59.12	60.66

表 2: 二分木モデルの実行時間 [秒](256node,16block)

4.3 マッピング時間

前節で使用した二つのモデルを 2 から 32 個のブロックにマッピングする際のマッピング時間を図 9に示す。ただし、マッピングには SS2(Sparc40MHz) を用いた。

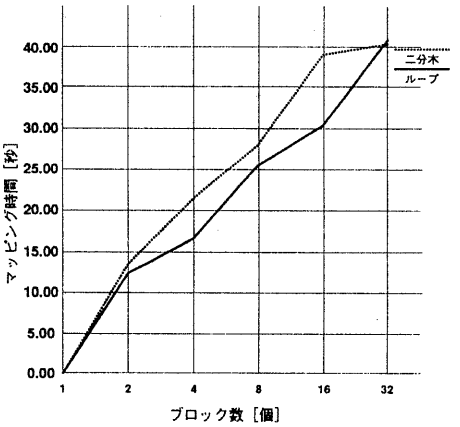


図 9: マッピング実行時間 (256 ノード)

図より、2^m個のブロックにマッピングする場合、m に比例したマッピング時間を要することが分かる。これに

より、非常に多くのPEを持つ並列計算機にも問題ない時間でマッピングをすることができる。

また、128から640個のノードで構成される二分木とループのモデルを16個のブロックにマッピングした場合のマッピング時間を図10に示す。シミュレーション

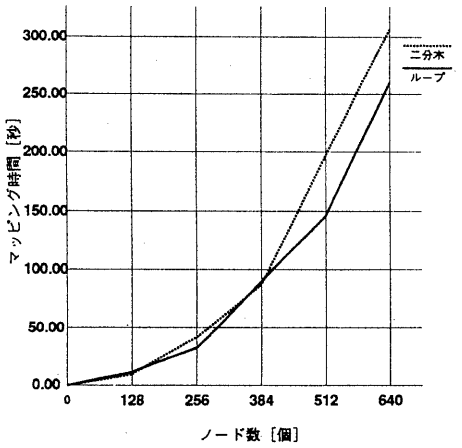


図 10: マッピング実行時間 (16 ブロック)

は複数の試行を行なわなければならないことを考慮すると、シミュレート時間に比べて十分短時間でマッピングされたことが分かる。

5 おわりに

以上の結果より、ノードの移動指標であるゲインの計算にヌルメッセージのオーバーヘッドを考慮することによって、実行時により近い通信コストが考慮でき、従来に比べて効率的なシミュレーションを行なうことができた。

今回は、ネットワークパーティショニング手法としては繰り返し改良法に絞った。繰り返し改良法とヌルメッセージを考慮したことによる効果を分離するためにも、他のパーティショニング手法についても検討していきたい。また、本稿で用いた2つのモデル以外の一般的なモデルでも有用性を評価していく予定である。

謝辞

最後に、AP1000を利用するに当たって多大なご協力をいただいた村岡研究室、富士通研究所の皆様方に大変感謝いたします。

参考文献

[1] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transaction on Software Engineering*, Vol. 5, No. 5, pp. 440–452, September 1979.

[2] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, Vol. 10, pp. 35–44, 1990.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pp. 175–181. IEEE, 1982.

[4] R. M. Fujimoto. Parallel discrete event simulation. *Communication of the ACM*, Vol. 33, No. 10, pp. 30–53, October 1990.

[5] R. M. Fujimoto and D. M. Nicol. State of the art in parallel simulation. In *1992 Winter Simulation Conference*, pp. 246–254. SCS, 1992.

[6] D. S. Jonson, C. R. Aragon, L. A. Mcgeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part 1, graph partitioning. *Operations Research*, Vol. 37, No. 6, pp. 865–892, Novenber-December 1989.

[7] B. W. Kirnighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291–307, September 1970.

[8] B. Nandy and W. M. Loucks. An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers. In *6th Workshop on Parallel and Distributed Simulation*, pp. 139–146. SCS, 1992.

[9] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, Vol. 38, No. 1, pp. 62–81, January 1989.

[10] 森雅夫, 宮沢政清, 生田誠三, 森戸晋, 山田善靖. オペレーションズリサーチ II. 朝倉書店, 1989.

[11] 川島幸之助. 通信網に関するシミュレーション. オペレーションズ・リサーチ, Vol. 38, No. 11, pp. 583–587, Novenber 1993.

[12] 森戸晋, 中野一夫, 相沢りえ子. SLAMII によるシステム・シミュレーション入門 改訂版. 共立出版, 1993.

[13] 米田清, 藤原睦. 半導体生産工程のシミュレーション. オペレーションズ・リサーチ, Vol. 38, No. 11, pp. 578–582, Novenber 1993.

[14] 若山邦紘. シミュレーションソフトと統計的側面. オペレーションズ・リサーチ, Vol. 38, No. 11, pp. 566–570, Novenber 1993.