計算機アーキテクチャ 34－3
（1979. 5. 23）

待ち行列システム・シミュレーションにおける並列処理

中川 徹　長谷川高志　近江谷康人
隈田一郎　相磯秀夫　（慶應義塾大学工学部）

1. はじめに

待ち行列論の目的は処理設備（Server）に被処理体（Transaction）が到着した時の待ち行列（Queue）の長さや待ち時間等を被処理体の到着時間間隔やServerの処理時間間隔から求めることである。[6]

これらの解を求めるには以下に示す2つの方法が考えられる。

(1) 待ち行列システムを状態確率に関する非線形連立微分方程式で表し、その解を解析的にあるいは数値計算で求める。

(2) 待ち行列システムをディジタル計算機上のソフトウェアである離散系シミュレータ上に記述し、シミュレーションにより解を得る。

最初の方法はトランザクションの到着時間間隔の分布、サービス時間間隔の分布及び窓口数に制約があり、一般的な場合について解析的な解を得ることは大変むずかしい。この解を高速な連続系シミュレータ，KCSS[2)3)]を用いて数値計算により求めることも可能であるが、モデルが大きくなるにつれて必要となるハードウェア量が増大し現実的でなくなる。

後者の方法はモデル上のQueueやServerに相当する機能ブロック間でトランザクションをやり取りすることにより離散系シミュレーションを行うものである。この方法の利点は、実測したヒストグラムを入力として与えられることや，サービス休止時間の設定が容易であること，及びトランザクションの属性（優先順位，種別等）に対する処理を行えることなどである。

このような利点を持った離散系シミュレーションは現在，GPSS,[7),8)] SIMSCRIPT[8)]等のシミュレーション言語で記述された汎用大型計算機上の離散系シミュレータで逐次に実行されている。しかしながら大規模なモデルを解く場合、トランザクションの発生により生じる事象の時刻管理や統計処理演算等に多大のCPU時間を費さなければならない。このためシミュレーション費用は高くつき、又解の得られる速度は遅くなる傾向があり、その価格性能比は良好とは言えない。

一方、連続系シミュレーションの分野において安価かつ高性能なLSIマイクロプロセッサを多数用い、MIMD方式[1)]の並列処理を実現する研究が当研究室で押し進められている。[2)3)]この研究成果をふまえて我々は『離散系モデルを並列処理によって高速かつ安価に解く専用シミュレータを開発し、その価格性能比の改善を図る。』ことを究極の研究目標として掲げた。

研究の第1段階に離散系モデルの1つである待ち行列モデルにおける並列処理方式について検討を行ったので以下報告をする。

2. 離散系シミュレーションにおける処理の並列性

並列処理により離散系シミュレータの処理の高速化を図るために、待ち行列システム・シミュレーションの解法手続きにおける処理の並列性を考える。

行うべき処理は大別して以下の4種類である。

i) イベント管理

Queue や Server などの機能ブロック各々へのトランザクションの到着による各機能ブロックの処理の起動及びそれらの間の時刻管理。

ii) Queue 操作

到着したトランザクションを Queue に加えたり、あるいは Server が Queue からトランザクションを取り出したりする操作。

iii) 属性処理

機能ブロック間を移動してきた複数のトランザクションの属性（優先順位、数値、種別等）に対する処理。

iv) 統計処理

各 Queue 及び Server における待ち時間と処理時間の平均や度数分布などを求める処理。

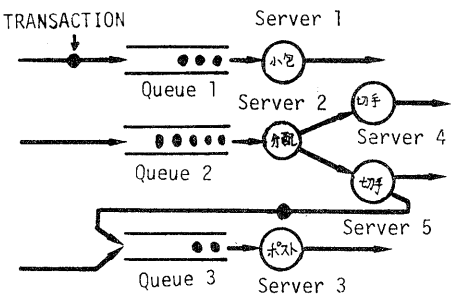

図1. 郵便局の窓口のモデル

イベント管理における時刻管理やイベントの検出及び処理の起動を、図1に示されるような Queue や Server といった機能ブロック毎に行うことができる。又、各機能ブロック単位で Queue 操作及び属性処理を行えるので i)～iii) の処理に関して並列性が存在する。

さらに統計処理の中で基本的な量である Queue の長さの度数分布とその平均値や、Queue の待ち時間の度数分布とその平均値といった量は各機能ブロック単位で独立に求まり、この処理にも並列性が存在する。

このように各機能ブロック毎に処理が分散することを処理の分散性と呼び時刻に対し不確定な処理を開始することを処理の非同期性と呼ぶことにする。

定性的にはモデルが大きくなり機能ブロックの数が増加すればするほど、又、発生し移動するトランザクションの数が多くなればなるほどイベント管理、Queue 操作、属性処理、統計処理に存在する並列性が高くなる。

3. 並列処理方式

離散系シミュレーションの処理の2つの性質と記述言語を考慮して我々はトランザクション駆動の概念に基づいた並列処理方式を採用した。

これは機能ブロック間に分散している各種の処理を機能ブロック単位で1かたまりの Task と考え、その Task 1つ1つを1台1台の論理的な QSV プロセッサに割りあて、各 QSV プロセッサ毎に独立して入力トランザクションの到着を検出し、各々非同期に処理を開始するという処理方式である。このような処理方式は他の分野でもすでに提案、開発[2),4),5)]されており、分散性。非同期性を持つ処理に有効であることが実証されている[3)]。

トランザクション駆動型の並列処理を離散系シミュレーションに適用した時の利点は以下の通りである。

- モデル上の機能ブロックとその処理が QSV プロセッサと1対1に対応しているため、複雑なコンパイルをすることなく実行可能である。
- QSV プロセッサが各々独立して処理の開始を実行時に検出できるので、実行前にはスケジュール不可能で、かつ非同期的に発生する処理に迅速に対処でき、モデルのもつ並列性に従った並列処理が自動的に達成される。

- ユーザは並列性の高い大規模な待ち行列モデルをQSVプロセッサに対するQueueやServerの機能指定とQSVプロセッサ間の結線指定の2つをするだけで，並列性を意識することなく簡単に記述できる。

反面，欠点として

- QSVプロセッサがモデル内の機能ブロックの数だけ必要であり、しかも大半のQSVプロセッサは入力トランザクションの到着待ちの状態（サスペンド状態と言い、図3-a，図3-b上で処理を行っていない空白の部分）で遊んでいる。
- QSVプロセッサが1度処理を開始するとその処理が複雑かつ多量であり、処理時間を多く必要とする。この間、シミュレーション上の時刻更新は特別な工夫をしない限り一時停止し、他のQSVプロセッサはサスペンド状態になったままである。
- QSVプロセッサで実行できる機能は予めシミュレータ製作者が用意した機能に限られてしまい，より細かな機能の指定ができない。

しかしながらこれらの欠点は後述するいくつかの工夫で克服できると考えている。

4. トランザクション・フロー・ダイヤグラム

トランザクション・フロー・ダイヤグラムはユーザがモデルを表現・記述する時に並列性を意識してモデルを記述しなければならないなどという不合理な負荷を一切取り払った最も原始的かつ合理的な記号言語である。

その記述は図2に示すようにトランザクションを発生するORGの他に，Queue（Qn），Server（SVn）といった機能ブロックやその間の結線を示すパス，及びこれらの上を移動していくトランザクションの3要素で表現される。実はこの表現は図1に示すモデルそのものである。

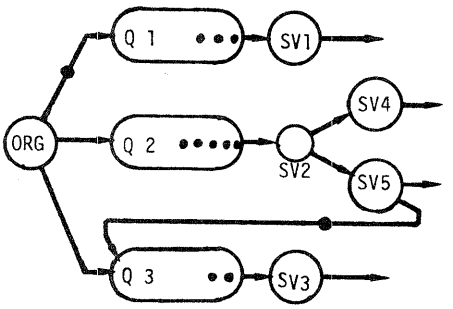

図2. TRANSACTION FLOW DIAGRAM

ユーザはまず機能ブロックを表す箱とその間の接続を示すパスを記述する必要がある。機能ブロックを表現するためにモデル上のServerに相当するブロックに，SV1，SV2，・・・SVn，・・・SVMというように名前をつける。次に各々のServerの中でその前に単一のQueueができるものはそのQueueに相当するブロックにSVnと同一の番号をつけた名前Qnを与える。それ以外のQueueブロックにはQM，QM+1，・・・という一連の番号付きの名前をつける。

このように記述する理由は、ServerとQueueの対応関係をはっきりさせることと，QSVプロセッサの一般的入出力構成を多入力1出力Queue・1入力多出力Serverの形にしたいからである。もちろんQSVプロセッサの一般的構成をこのように決めても何ら不都合が生ずるものではない。例えば多入力多出力Queueの表現は，サービス時間がゼロの即時サービス特性を持つServerがQueueの後にあると考えれば良く，又そのほうが自然である。

Queue や Server に名前を付け終った後でそれらの間の結線を Q1－SV1，Q2－SV2，SV2－SV4，SV2－SV5，SV5－Q3，Q3－SV3，ORG－Q1，ORG－Q2，ORG－Q3という具合に記述する。アンダー

ラインをつけた結線は同一番号が付けてあるので省略することができる。

最後にユーザは、各機能ブロックの入力特性、サービス特性、出力特性を番号付きの名前に対し与え、その名前を関数名とする。その際、全く同一の特性を持つ関数は名前を＝で結んで定義しても良い。

- 入力特性には入力端子間の優先順位と到着トランザクションの属性に対する条件を記入する。
- サービス特性にはQueueの最大長，機能ブロック内の初期トランザクションの数，属性処理の内容，及び任意の分布関数に従ったサービス速度を記入する。
- 出力特性には各出力端子に対する優先度の分布を記入する。

以上のように記述されたトランザクション・フロー・ダイヤグラムは、ユーザが気付くことが困難な高い並列性の存在を無理なく表現できる一種のプログラムである。このプログラムを直接実行できる計算機はいわばトランザクション・フロー・プロセッサとも言うべきものである。この実現にあたって特に問題となる事項は以下の２つであろう。

i) 多数のQSVプロセッサ間のシミュレーション上の時刻同期や、トランザクションなどのデータ転送時の同期機構，及びアルゴリズムの開発。

ii) トランザクション・フロー・プロセッサ上の各QSVプロセッサ間の結線に相当する交信領域のハードウェア構成。

5. QSVプロセッサ間の時刻管理

多数のQSVプロセッサが各々独立してイベントを検出し非同期に実行を開始しても、トランザクション・フロー・プロセッサ全体の動き，及びそのシミュレーション結果が正しく保証されなければならない。そのためにトランザクションの発生，転送，受領と言ったQSVプロセッサ間の入出力操作においてシミュレーション上の時刻同期を取る必要がある。このような時刻同期の管理を行う方法として次の２つが考えられる。

A. 全てのQSVプロセッサを１つの主時計に同期させる集中管理による時刻同期方式（集中管理方式）。

B. 互いに結線されたQSVプロセッサ間で時刻情報をやり取りし、部分的な同期で全体同期を取る分散管理による時刻管理方式（分散管理方式）。

さらに各々の方式に対し、以下に示す２つの時刻更新方法が適用できる。

a. 時刻を常に１定値だけ進める固定刻み幅による時刻更新方式。

b. ある事象の発生時刻から次の事象の発生時刻までの時間幅だけ時刻を進める可変刻み幅による時刻更新方式。

以上の時刻管理方式と時刻更新方式を組み合わせて合計４つの方式が得られる。待ち行列システム・シミュレーションにおける処理の開始はトランザクションの到着という事象の発生によるので、時刻の更新は可変刻み幅による時刻更新方式を採用する方が良い。

トランザクション・フロー・プロセッサでは時刻管理に集中管理と分散管理のどちらの方式を採用すべきか、図２のトランザクション・フロー・ダイヤグラムを例にとって検討する。

簡単のためにServer (SV) のサービス間隔とOriginate (ORG) からのトランザクションの発生間隔を共にΔT_sとする。又，SVやORGからのトランザクションの行先はラウンド・ロビン方式に従うものとし、初期状態$T_s=0$におけるトランザクションの数をORG，SV_5が各々１ケでQ_3が２ケ，Q_1が３

チ，Q_2が4ケと仮定する。一方，このモデルのシミュレーションを行うQSVプロセッサがQueueへの入力操作(I)とServerからの出力操作(O)に要する処理時間を1ユニット・タイムと仮定する。

図3-a.と図3-b.に示すプロセッシング・タイム・フロー[2]は，横軸にシミュレータの処理時刻を実時間で取った時の各QSVプロセッサの処理の流れを表わしたものである。

図3-a.は集中管理方式であり、図からも明らかなようにシミュレーション時刻$n\cdot\Delta Ts$上のQSVプロセッサの処理がすべて終了してから同期的に全てのQSVプロセッサがΔTsだけ時刻を更新している。一方，分散管理方式

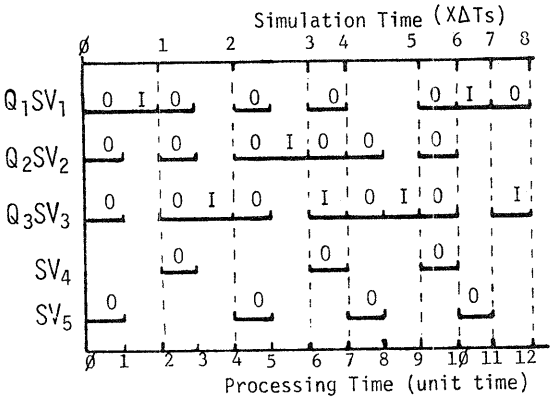

図3-a.CONCENTRATED TIME-MANAGEMENT

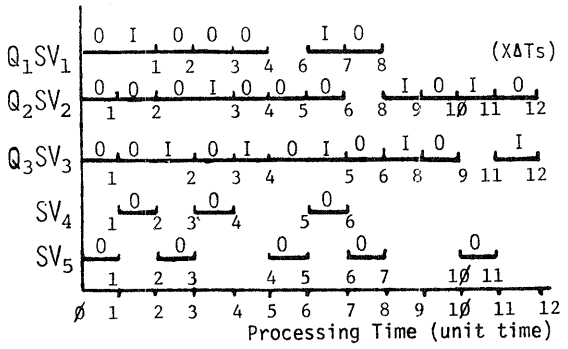

図3-b. DISTRIBUTED TIME-MANAGEMENT

においては図3-b.に見られるようにある時刻$n\cdot\Delta Ts$の処理を終えたQSVプロセッサが独立に自分の時刻を$(n+1)\cdot\Delta Ts$に更新する。そしてたまたまトランザクションが到着した時に、そのトランザクションに対する処理を独立に開始している。その結果処理が詰めて実行されるので集中管理方式よりも効率良く並列処理が実現されている。

例に示した図2のモデルは特に一般的な待ち行列モデルでもなく，又図3-a.と図3-b.を作図した時のいくつかの仮定はかなり大ざっぱである。しかしながら、より一般的な待ち行列モデルではトランザクションの発生・転送が多数の機能ブロックに分散しており，その間隔が分布を持った乱数でばらついて決まるという事実に基づいて、我々はシミュレーション上の時刻管理に『分散管理方式で可変刻み幅による時刻更新方式』を採用することとした。

6．QSVプロセッサ間の同期機構

言語の記述に従って互いに結線された同一構造のQSVプロセッサの間で前述の時刻管理方式を実現するためには，以下の2つの情報を送り主のQSVプロセッサから次段のQSVプロセッサへ転送する必要がある。

i) トランザクションを発生・転送したQSVプロセッサ各々が独自に持つシミュレーション上の時刻。

ii) トランザクション各々が持つ優先順位や数値といった属性。

6.1. パケットの概念の導入

トランザクションの発生した時刻（シミュレーション上の時刻）とそれが持つ属性に送り先を付けた1かたまりのデータを<u>パケット</u>と呼ぶことにする。パケットの送り先は1つ以上でも良いし動的に変化しても良い。

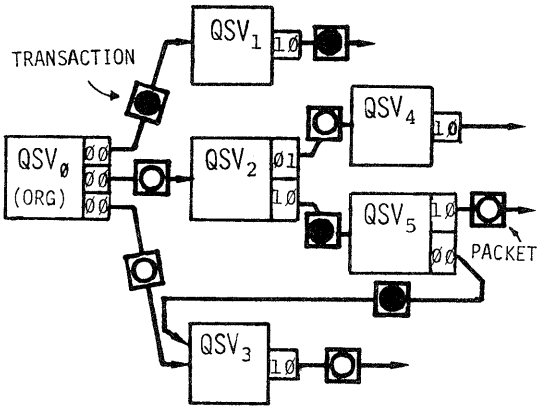

図4．QSVプロセッサ間の同期

図4.にパケットを用いた同期概念の例を示す。図中，QSVプロセッサ間の接続を示すリンク上に記された ● 印がトランザクションであり，[●] と [○] 印がパケットを示す。このような時刻情報と送り先情報を伴ったパケットはその送り主に接続されている全てのQSVプロセッサに転送される。

転送されたQSVプロセッサの中で送り先に指定されたものはそのパケットを自分宛てのトランザクションが含まれた実パケット（図4.中，[●]と表記）として受け取る。そして自己の時刻を，到着したパケットの時刻にまで更新し，次にそのトランザクションに関するQueue操作，属性処理，統計処理，出力パケット生成といった一連の処理を開始する。

一方，送り先に指定されなかったQSVプロセッサはそのパケットを時刻情報のみの空パケット（図4中，[○]で表記）と認識し、自己の時刻更新を行い，次段へのパケットを生成する。

空パケットの概念の導入によりトランザクションがたまたま行かなかった後段のQSVプロセッサ全てにも，トランザクションが到着したQSVプロセッサと同様に時刻の更新を知らせることが可能となった。その結果，時刻の完全な分散管理がトランザクション・フロー・プロセッサ上で実現可能となる。

6.2. 同期フラグ

QSVプロセッサ1台は多入力・多出力の構成を一般形としている。それゆえ入力に到着する複数の異なる時刻を持つパケットを順序良く取り込み、送り主のQSVプロセッサにパケット受領を知らせる必要がある。この要求を満たすために QSV プロセッサのリンク全てに2ビット幅の同期フラグを設けた。図4のQSVプロセッサの出力端子に示す [ØØ]，[Ø1]，[1Ø] がそれである。

フラグが2ビットになっている理由は，リンク上の次の3状態を表現したいからである。

i) 転送すべき新しいパケットがリンク上に存在する状態で、これを図4の QSV_0 の出力端子にあるフラグのように [ØØ] と表現する。

ii) リンク上に存在するパケットが既に次段のQSVプロセッサに受け取られてしまい，新しいパケットが要求されている状態で、図4の QSV_1 の出力端子に示すように [1Ø] と表す。

iii) 図4において QSV_2 と QSV_4 とを結ぶリンク上のフラグに [Ø1] と表された状態で、これは QSV_2 からの実パケット [●] を QSV_4 が拒否していることを示す。モデル上ではQueueの最大長を越えた数の入力トランザクションを追い返していることに相当する。この時 QSV_4 は時刻更新のために時刻情報のみ受け取る。

以上の3つの状態を表す同期フラグの上位1ビットを正常受領を示すACKフラグ，下位1ビットを異常受領を示すNAKフラグと呼ぶことにする。ACK，NAKフラグを"Ø"にリセットできるものは送り主のQSVプロセッサであり"1"にセットできるものはパケットを受け取るQSVプロセッサだけである。

6.3. QSVプロセッサ同期アルゴリズム

同期フラグを用いた多入力・多出力QSVプロセッサの時刻及びデータ転送に関する同期アルゴリズムを図5に示す。このフローチャートは大きく分けて以下の3つの部分から構成される。

(1) パケットの入力同期操作

到着した複数のパケット各々の持つ時刻を認識してパケットの発生した順序を乱さずにQSVプロセッサ内に取り込むための同期操作。

(2) パケットに対する処理

パケットの取り込みに伴う各QSVの時刻更新と出力パケットの生成。

(3) パケットの出力同期操作

次段のQSVプロセッサに確実に新しいパケットを転送するための同期操作。

ここでは特にパケットの入出力同期操作に関係する事項について述べる。

パケットの入出力同期操作は、ある多入力Queueに到着した複数のパケットの中から時刻的に古いものから順に取り込むことが、その操作の全てである。そこで多入力Queueはその入力端子全てに新しいパケットが揃ったことを、入力端子上の同期フラグが全て 00 であるかを検査することで認識する。次にその新しいパケットの中から一番古い時刻を持つパケットを選択しQSVプロセッサ内部に取り込む。そして図6のアルゴリズムに従ってそのプロセッサ内部の時刻更新とQueue操作を行い、送り主のQSVプロセッサにパケット受領を知らせる。

出力同期操作において多出力の端子を持つQSVプロセッサは、その多出力のリンク上にある全ての同期フラグが 10 で受領を示していることを確認してから次の新しいパケットを次段全てのQSVプロセッサに転送できる。もし 01 のフラグがあった時にはトランザクションが拒否されたのであるから、予めユーザによって指定された処理がある時はそれを実行する。もし処理の指定がない時は拒否された数を計数して次の新しいパケットを転送する。パケットの転送は新しいパケットをリン

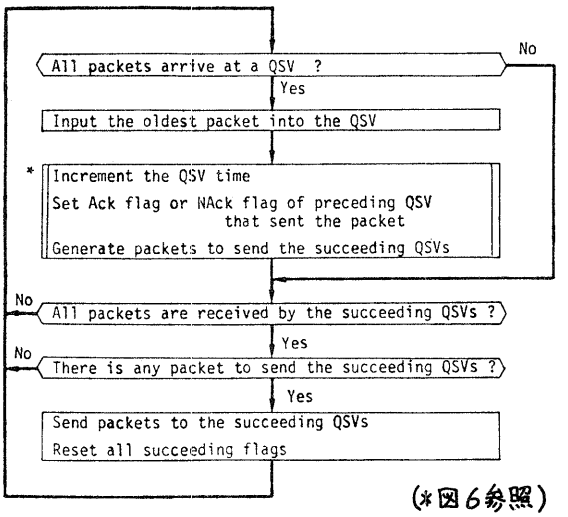

図 5. Synchronization algorithm

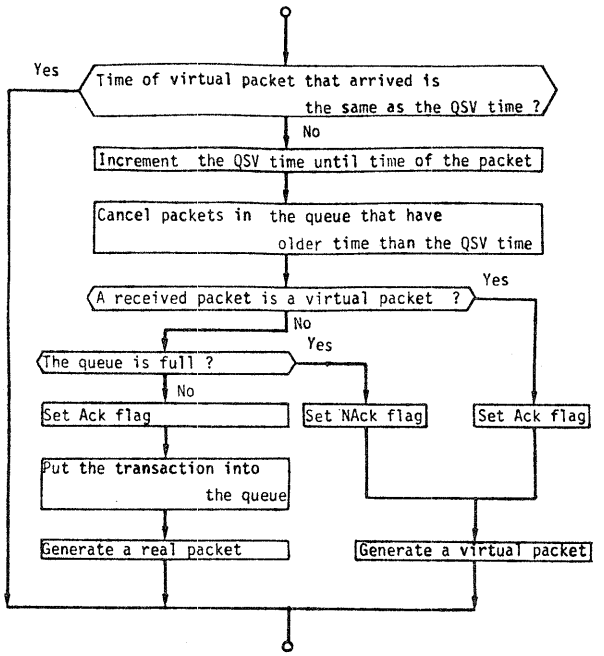

図 6. Algorithm in QSV processor

ク上に出力した後に出力側の同期フラグ全部を[00]にセットすることで行われる

以上のような同期フラグを用いた同期アルゴリズムを用いることによってQSVプロセッサ間の約束だけでデータと時刻の同期を分散して取れる。

7. おわりに

離散系シミュレーションにおける処理の高速化を,安価なマイクロプロセッサを用いて実現することを考え,そのための並列処理をトランザクション駆動型で行うことを提案した。今回は特にトランザクション・フロー・ダイヤグラム,QSVプロセッサ間の時刻管理,QSVプロセッサ間の同期機構を中心に述べた。完全に分散化された時刻管理とユニークな同期機構はQSVプロセッサの稼動率を自動的に極大値に向上させるものである。現在,QSVプロセッサの処理を行う物理プロセッサ(PU)のハードウェアとソフトウェアの設計,及びQSVプロセッサ間の交信領域に相当する,完全に同時読み出しが可能な共有メモリ(マルチ・リード・パケット・メモリ)の設計を行っている。

このプロトタイプ,KDSS-I (keio Discrete System Simulator model-I)では約50台のPUを用いて100～500のQueueとServerを有するモデルを解けると考えている。

参考文献

[1] M.J.Flynn,
" Very High-speed Computing Systems,"
Proc,IEEE,54,12,pp1901-1909,Dec,1966.

[2] E.Yura et al,
"An approach to Parallel Processing for Continuous Systems Simulation with Microprocessors," 2nd UJCC,pp172-177, 1975.

[3] R.Yoshikawa,et al,
"A Multi-microprocessor Approach to A High-speed and Low-cost Continuous-system Simulation,"NCC,pp.241-246,June, 1977.

[4] J.B.Dennis,D.P.Misunas,
"A Preliminary Architecture for a Basic Data-Flow Processor,"
Proceeding of the Second Annual Symposium on Computer Architecture,IEEE,New York, pp.126-132, january,1975.

[5] D.P.Misunas,
"A Computer Architecture for Data-Flow Computation,"
MS Thesis , Department of Electorical Engineering and Computer Science, M.I.T. Cambridge, Mass., June,1975.

[6] 森村,大前,
「応用待ち行列論」
日科技連

[7] GPSS 1100 マニュアル
日本ユニバック

[8] SIMSCRIPT マニュアル
日本ユニバック

[9] R.Efron, G.Gordon,
"A general purpose digital simulator and example of its application,"
IBM SYSTEM JOUNAL Vol.3,NO.1,1975.