

並 列 型 会 計 シ ス テ ム

米 川 清*

*熊 本 商 科 大 学

並列型会計システムについて、実証研究を行う。会計システムは、法定帳簿の連続性のために、逐次的プロセスを遵守しなければならない。その一方で、並列処理も要請されている。

以上から、システム理論とソフトウェア工学の立場から上記のモデル化を試みる。

Parallel Process for Accounting System

Kiyoshi Yonekawa*

* Kumamoto University of Commerce

On my thesis, I introduce a new paradigm of accounting system, which use parallel system approach.

The paradigm is composed of two model, that is, Automata Theory and Software Engineering.

1. はじめに

並列処理のためのシステムソフトウェアについては、かなり広範にわたり報告がなされている。その内容はOS、コンパイラ、デバッカの研究が主体であり、いずれも技術計算の並列処理を前提としている。他方、事務経営管理分野の経理財務システムでも並列処理の要求が提起され、本稿では当該アプリケーションシステムの並列処理の仕組みについて実証研究を行う。

経理財務システムの成果物は法定帳簿の作成である。その責務は帳簿組織の連続性にあり、法体系でも義務づけられている。この連続性を遵守するためには、システム自体の非決定性は認められない。上記の要件を満たすためには、プログラムの挙動を厳密に制御する逐次型のシステムであらねばならない。そして、並列処理を実施せねばならない内部矛盾を持つ。この事務処理システムを並列処理としてとりあげるには、些かおこがましさを禁じえない。

しかし、ビジネスアプリケーションの現場での並列処理への具体的な対応について述べられた参考文献を、筆者は未だ目にすることがない。上記の認識から、かつて現場に携わった一人の技術者の視点から開発モデルの一例を述べる。

2. システムの背景

モデルとしてとりあげる経理財務システムは、全国に設置した多数のWSから構成された古典的なLANと商用大規模コンピュータを結合したインフラ環境にある。LAN（スター型）では財務のサブシステムがリアルタイムで稼働し、大規模コンピュータにはリアル処理された財務データと経理データをバッチ処理する基幹システムが配置される。基幹システムでは日々に50万件の大量トランザクションが処理され、システム規模はリアルとバッチを合算すると200万ステップ（COBOL 換算含む）である。WSの入力画面数は120である。

次になぜ並列処理が必要であるかを述べる。経理財務システムの最大の使命は確定決算にある。エンドユーザは3月末日までに、取引の全てを入力する。その結果、過誤の修正、エラーデータの再入力、入力漏れ、決算整理のための特殊入力等が発生する。数字の最終確定までに20日程の猶予が必要である。

上記の猶予期間を追加日と呼び、この追加日は t_1 、 t_2 、 t_3 、 $\dots$ 、 t_m で示す。一方この追加日期間にも4月1日以降の取引が併行して発生する。従って、処理の正当手順にならない逐次的に処理を行えば、図1のとおりである。

3月末日 $\longrightarrow$ 3/ t_1 $\longrightarrow$ 3/ t_2 $\longrightarrow$ 3/ t_3 $\cdots \longrightarrow$ 3/ t_m $\longrightarrow$ 4/1 $\longrightarrow$ 4/2

図1： 逐 次 的 処 理

しかし4月1日のデータ入力が可能となるのが4月20日頃では、滞った4月分の事務量は膨大である。

以上から、追加日処理と期初処理の並列処理は必然となる。上記の流れを図2に示す。

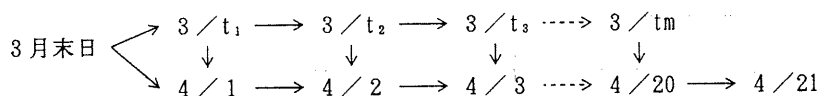


図2： 追加日 期 間 の 並 列 処 理

上図からユーザは3月 t_1 日のデータと4月1日のデータの同時入力が可能となる。従って4月1日の帳簿のポジションは3月末日+3月 t_1 日トランザクション+4月1日トランザクションとなる。

ここで問題点として記述しなければならないのは、4月1日の真実のポジションは3月 t_m 日の確定値に4月1日のトランザクションを反映して成立するのであり、この並列処理では非決定の伝搬遅延データが存在する事実である。

しかし時系列で見れば、4月1日のポジションでは多少の攪乱が起こりえるが、段階的に4月の帳簿は是正され3月 t_m 日のトランザクションが接続された時点、すなわち4月20日に数値の歪みは全て改修される。

3. システム理論のアプローチ

初歩的な有限オートマトンにより、当該経理財務システムについて考察する。数理的システム理論の立場で記述すると、 A を入力アルファベットとし、 B を出力アルファベット、 C を状態アルファベットとする。状態と出力の動作を2つの関数で示すと、以下のとおりである。

$$\delta : C \times A \rightarrow C$$

$$\lambda : C \times A \rightarrow B$$

δ は状態遷移関数、 λ は出力関数である。

次にあるべき図1のような逐次的サブシステムについて、オートマトン定義の δ のみを考察し図3に示す。

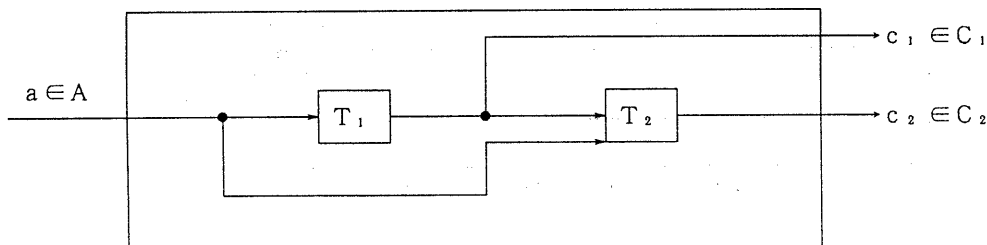


図3： 逐次システム

$T_1 = \langle A, C_1, \delta_1 \rangle$, $T_2 = \langle C_1 \times A, C_2, \delta_2 \rangle$ とする遷移システムを考えると、 T を次のように定める。 $T = \langle A, C_1 \times C_2, \delta \rangle$

ただし、 $\delta : (C_1 \times C_2) \times A \rightarrow (C_1 \times C_2)$ は

$$\delta((c_1, c_2), a) = (\delta_1(c_1, a), \delta_2(c_2, (c_1, a)))$$

このとき T は、 $T = T_1 \cdot T_2$ となる。

この入力-出力システムの結合を図2に基づき、単純な並列システムに分解すれば図4である。

正しくは $\delta : C \times A \rightarrow P(C)$ (非決定論)の形で表記すべきであるが、単純化のために決定論の場合を前提とする。

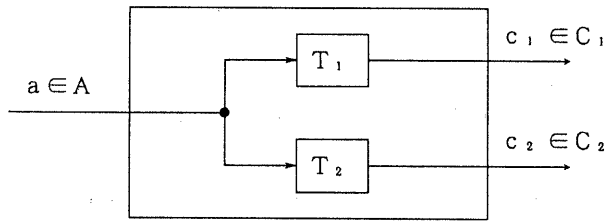


図4： 並列システム

図4の $T_1 = \langle A, C_1, \delta_1 \rangle$, $T_2 = \langle A, C_2, \delta_2 \rangle$ とすれば、 T を次のように定める：

$$T = \langle A, C_1 \times C_2, \delta \rangle$$

ただし、 $\delta : (C_1 \times C_2) \times A \rightarrow (C_1 \times C_2)$ は

$$\delta((c_1, c_2), a) = (\delta_1(c_1, a), \delta_2(c_2, a)) \text{ である。このとき } T \text{ は、}$$

$$T = T_1 + T_2 \text{ となる。}^{1)}$$

これは、図3のあるべき論からすれば、もちろん矛盾である。以上から、アプリケーションシステムの対応が必然となる。

4. アプリケーションモデルの分析

並列型経理財務システムのおおよそのアプリケーションモデルを図5に示す。

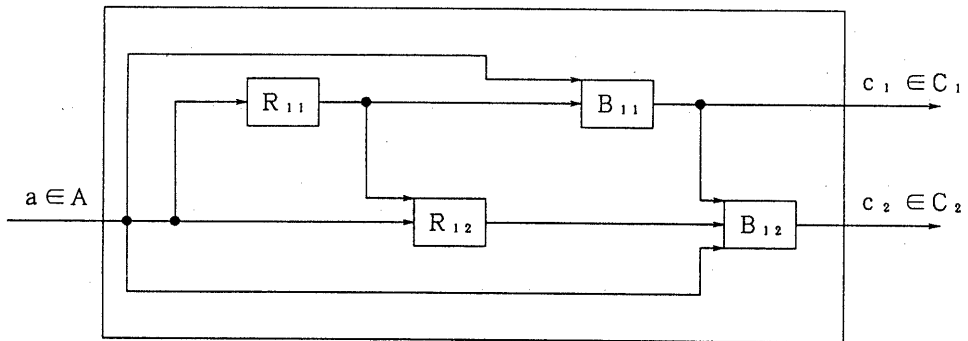


図5： 並列型アプリケーションモデル

R はリアル系サブシステムであり、 B はバッチ系サブシステムである。各サブシステムの意味は下記のとおりである。

R_{11} ：追加日手形システム

R_{12} ：期初手形システム

B_{11} ：追加日基幹システム

B_{12} ：期初基幹システム

R_{11} と R_{12} 、 B_{11} と B_{12} は全て同一のプログラムである。入力モデルをかりに手形システムに限定し、後述のパラメータに3月1日を指定すれば $R_{11} \rightarrow R_{12} \rightarrow B_{12}$ のデータの流れとなり、パラメータに4月1日を指定すると $R_{12} \rightarrow B_{12}$ の流れとなる。

そこでデータ依存関係とプロセス間の同期をどうとるかについて、手形システムを一例にその問題を図6に示す。

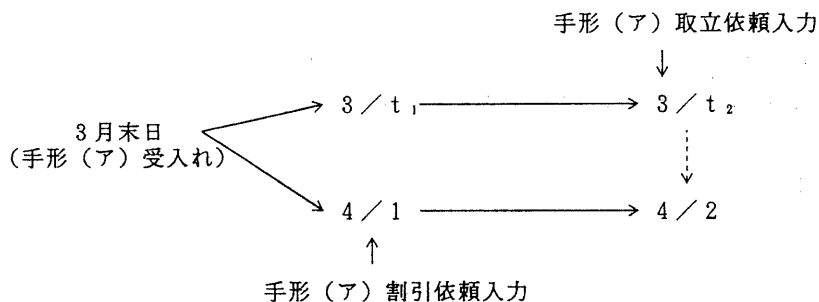


図6： データ依存関係とプロセス間同期の問題

図6の例では単純な並列処理では正常処理となる。すなわち、追加日系では3月末日受け入れた手形(ア)を3月 t_2 日に取立依頼にだすので別段の問題は生じない。

一方期初系では3月末日に受け入れた手形(ア)を4月1日に割引依頼にだすので問題は発生しない。次に4月2日に、追加日3月 t_2 日に入力され追加日系では正常処理された取立依頼のデータが流れてくるが、割引にだした手形(ア)を取立にまわすことは不可能である。従って、この取立データはエラーである。

以上から単純な並列処理では、手形(ア)は追加日では取立の状態、期初では割引の状態となってプロセスの内部状態に不整合が生ずる。帳簿の連続性を維持する上で、入力タイミングのずれから派生する不整合を制御する仕組みが必須となる。

上記のプロセスの内部状態を制御するアプリケーションの仕組みを、下の図7に示し、若干の説明を述べる。²⁾

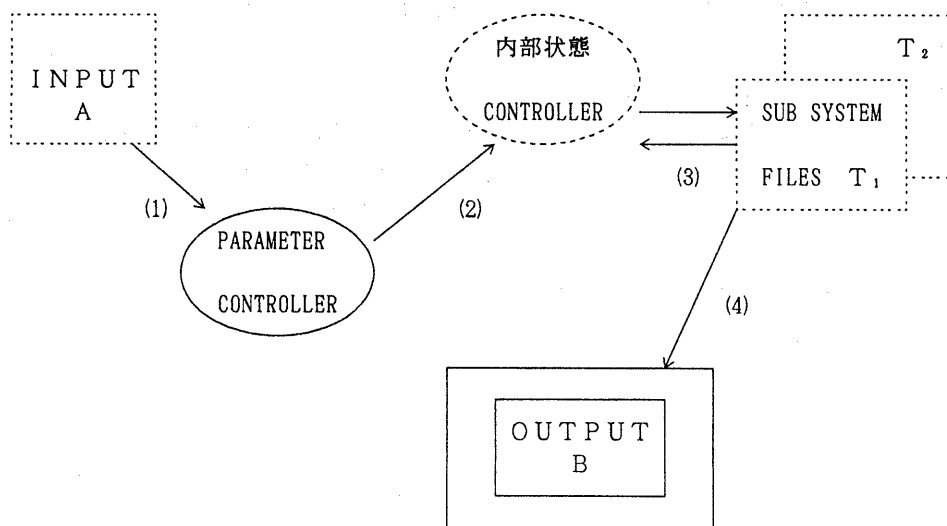


図7： 内部制御システム

4.1 4月1日の入力

- (1) パラメータに4月1日を設定する。
- (2) 内部状態コントローラに「手形（ア）割引」と記録する。
- (3) 期初系はサブシステム・ファイルT₂を更新する。
- (4) 法定帳簿に出力する。

4.2 3月t₂の入力

- (1) パラメータに3月t₂日を設定する。
- (2) 内部状態コントローラには、4.1(2)の「手形（ア）割引」が先行登録されているため3月t₂の取立データはエラー。

以上から3月末日に受け入れた手形（ア）は3月t_m日では受け入れのままとなり、4月1日に割引にだされ、帳簿の連続性は保たれる。

また、かりに4月1日に割引依頼したのが誤りであった場合であれば、4月3日にキャンセル行為を行う。

これにより内部状態コントローラから手形（ア）の登録が抹消され、追加日3月t₄に取立の再入力をすれば良い。

5. おわりに

大規模事務処理システムの並列処理について述べた。大量トランザクションの定型業務処理を主体とする事務処理システムは、アカデミックな話題に乏しいためか、ややもすると軽視されがちである。

さらにバッチ性能を重視する観点から、階層的且つ構造的にシステム分析・設計されていても複雑なプログラム群を大量にかかえているのが現状である。この複雑なプログラムをメンテナンスするプログラマー達もおなじように、大量に存在する。

こうした背景から、今日ではダウンサイジングが加速化している。オープンシステム論者は、この種の大規模システムを含め、メインフレームを「Legacy System」として批判する。⁹⁾

しかし、ダウンサイジング指向の開発のメソドロジーについて述べられた研究成果について、筆者は未だ目にすることがない。

反面、Hidden Cost について述べられた文献はしばしば目にする。

筆者は、長い射程でエンドユーザーコンピューティングやダウンサイジング（ライトサイジングではない）について観察するつもりである。

参考文献

- 1) 高原康彦・飯島淳一：システム理論，共立出版，PP79 - 80(1990)。
- 2) 山田 剛：並列処理プログラムにおけるプログラムデバッグ，情報処理，Vol. 34, No 9, PP1170 - 1178(Sep. 1993)。
- 3) 村井修造：ダウンサイジングとオープンシステム，何がそうさせるのか？，情報処理，Vol. 34, No10, PP1234 - 1239 (OCT. 1993)。