修 士 論 文

マルチスレッド型 ATM 交換機アーキテクチャ

指導教官　日比野 靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

松田 理絵子

1998 年 2 月 13 日

要 旨

本稿では、ATM 交換機の実現法として、マルチスレッド型 ATM 交換機アーキテクチャを提案する。本方式の ATM 交換機は共有メモリ方式を用いていて、セルスイッチングをマルチスレッド型プロセッサを用いたソフトウェア処理で行なう。

このような ATM 交換機では、仮想回線毎への資源割当がクロック単位で可能になる。共有メモリに対するスレッド毎のアクセスは時分割に行なわれるので、競合がない。仮想プロセッサに相当するハードウェアコンテクストが複数あるので、多様な仮想回線への資源割当が性能を低下させることなく柔軟に行なうことができるとみなすことができる。

本稿では、このマルチスレッド型 ATM 交換機アーキテクチャの仕様、アルゴリズム、性能評価シミュレーション結果を示す。

目 次

第 1 章

序論

近年、ネットワークの広範化に伴い、様々なデータ通信方式が提案され、実現されている。その通信方式の一つに、QoS(Quolity of Service : サービス品質) 保証をする ATM (非同期転送モード) 方式がある。この方式では、固定長パケットであるセル単位でデータを伝送する。ATM 方式では、このセルをスイッチングさせるために、ATM 交換機が必要である。従来の ATM 交換機では、セルの高速転送をして QoS 保証をしようというアプローチをとっていたので、交換機をハードウェア化させることに研究の主眼が置かれていた。しかし、ハードウェアでは QoS に応じた柔軟な制御性に欠けるという問題があった。そこで本研究では、高速転送の代わりに柔軟性を供給することで QoS 保証をするアプローチの ATM 交換機として、マルチスレッド型 ATM 交換機を提案する。本方式では、交換機をハードウェアではなくマルチスレッド型プロセッサを用いたソフトウェア処理で実現する。

本論文では、まず、本章にあたる第 1 章で序論と本論文の構成について述べる。次の第 2 章では、具体的に本研究の概要として、背景や目的、前提となる知識を挙げる。第 3 章では、マルチスレッド型 ATM 交換機の設計として、仕様や処理アルゴリズムなどについて述べる。第 4 章では、設計した ATM 交換機の有効性を検証するために、セルスイッチングをシミュレートし、本設計の ATM 交換機が性能面の仕様を満たせることを示す。最後に、第 5 章では、結論として本研究についてまとめる。

第 2 章

本研究の概要

本研究では、ATM セルのスイッチングをマルチスレッド型のプロセッサを用いたソフトウェア処理で実現し、それが従来のシングルスレッド型プロセッサによるソフトウェア処理に比べてどれほど有効性を持ち得るのかを検証する。そして、最終的にはマルチスレッド型 ATM 交換機による柔軟な制御による QoS 保証を目指す。

2.1 背景と目的

ATM 交換機は高速パケット交換 (53 バイトの固定長パケット交換) を実現する。従来、研究の主流は超高速ハードウェアスイッチに傾いていた。理由はソフトウェアでは性能が出なかったからである。しかし、ハードウェアスイッチングでは柔軟な制御性に乏しいので、結果的に、パケットロスが生じてスループットの低下を招く。こうなると、当然、ATM において、『帯域幅割り当てへの柔軟性の供給とネットワークへの狭帯域から広帯域までのサービスの実現』[4] を達成するのは難しい。このことを達成するには、パケットロスの影響を最小限に抑えるような、柔軟な制御が必要である。そこで、年々プロセッサの処理速度が向上していることを併せて考えると、ATM 交換機をプロセッサによるソフトウェア処理で実現すれば、柔軟に制御することで、パケットロスの影響を最小限に抑えることができ、ネットワークへの狭帯域から広帯域がかないそうである。本研究の着眼点はここにある。

本研究では、ATM セルをスイッチングする ATM 交換機を、ソフトウェア処理で実現し、ATM 交換機に柔軟性を与える。

2.2 各方式の説明

本研究では、共有メモリ型 ATM 交換機をマルチスレッド型プロセッサを用いて実現する。そこで、これらキーワードとなっている通信方式としての ATM、交換機実現方式としてのソフトウェア処理、交換機構成方式としての共有メモリ型を前提知識として説明する。

2.2.1 ATM

ATM とは、Asynchronous Transfer Mode の頭文字をとった言葉で、非同期転送モードのことである。ATM は、B-ISDN(Broadband Integrated Services Digital Network) 実現に必要な伝送／交換技術として、STM(Synchronous Transfer Mode：同期転送モード) に変わって、ITU-T[1]によって採択された技術である。

交換方式には回線交換方式とパケット交換方式とがある。回線交換方式は、1 つの呼が発生するとそれに対して 1 本のチャネルを発着端末間に占有させる方式である。パケット交換方式は、データを一定長以下のパケット (荷札を付けた小包) に区切って送るので、複数の呼で回線を共有することができ、回線交換方式に比べて回線の使用効率がよい。

STM は、1 本のチャネルを周期毎に現われるフレームのタイムスロットに分割して転送する時間位置多重方式である。特定の呼をフレームから抽出するといったチャネルの識別はフレーム内のタイムスロットの時間的位置から判断する。この方式は 1 チャネル当たりの最大通信速度が一定という制約がある。よって、B-ISDN のように、多様な情報、すなわち通信速度が異なったり変化したりする情報を扱う場合は、1 つの呼に対して最大通信速度に合わせた複数のチャネルを割り当てる必要がある。STM は時分割と回線交換を併せ持った技術であるが、送信するデータがない場合にもタイムスロットが割り当てられるので、空きチャネルが生じ、回線使用効率が悪くなるという短所がある。

一方の ATM は、回線交換とパケット交換の長所を併せ持った技術とされる。すなわち、ATM は統計多重をすることで、時分割効果よりも高い多重効率を得られ、個々の通信に割り当てる伝送帯域を自由に設定できる。このような特徴から B-ISDN の中核技術とされる。

ATM では全ての情報を『セル』という 53 バイトの固定長パケットで扱う。すなわち、この固定長データ列 (パケット) であるセルが、交換／多重の単位となる。また、高品質伝送を前提に、プロトコルは簡素化されている。このようにすることで交換処理をハードウェ

[1]International Telecommunication Union-Telecommunication Sector 国際電気通信連合電気通信標準化部門

アで実現することができ、パケット交換の高伝送効率を受け継ぎながら交換遅延も小さくすることができる [6]。1 個の ATM セルは、5 バイトのヘッダ部分と 48 バイト情報フィールド (ユーザペイロード) 部分から構成される。ヘッダには、一般的フロー制御 (GFC)(但し、VPI フィールドになるときもある)、セルが属するコネクションを識別するための仮想チャネル識別子 (VCI)、仮想パス識別子 (VPI)、輻輳が起きた時にセルの廃棄をするかどうかを示すセル優先識別 (CLP)、網制御情報を区別するためのセル種別識別 (PT)、ヘッダ誤り検出／制御 (HEC) などの機能がある。ATM セルはプロトコルスタックでいうと ATM レイヤで扱われる。

ATM 交換の特徴は、ルーティング情報をヘッダに格納しているので、各 ATM 交換機が自立的にセルを中継・交換できることや、交換処理をチップ化 (ハードウェア処理) できるので、交換処理を高められるということである。ATM 交換網は仮想パス (VP) と仮想チャネル (VC) という 2 レベルのネットワークを構成する。

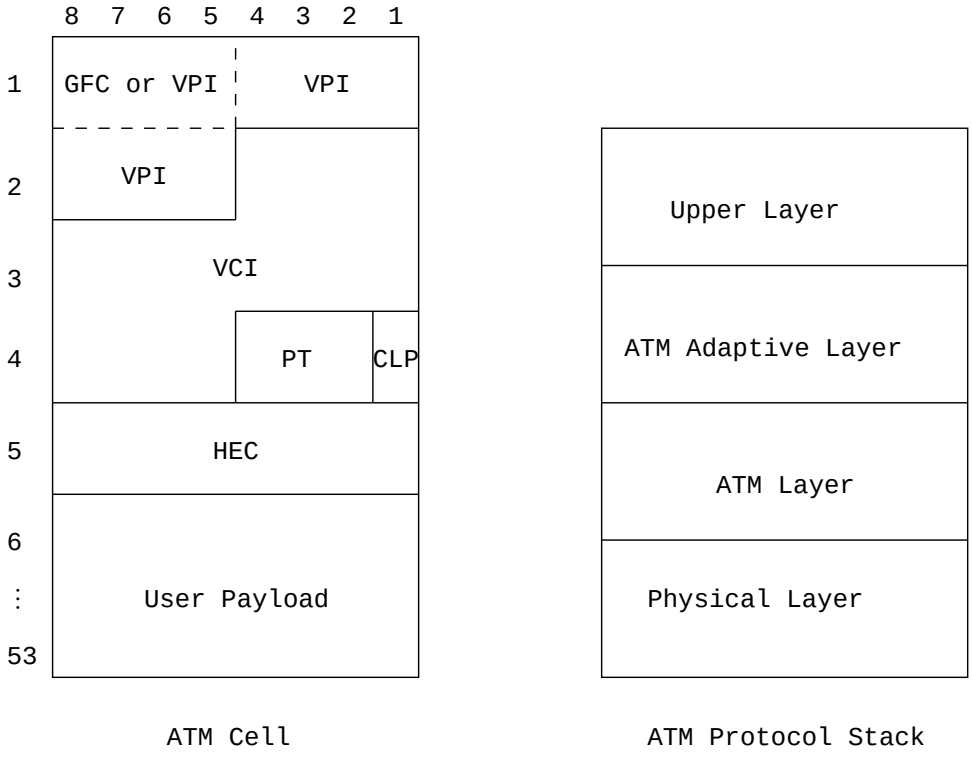

図 2.1: ATM セルの構成図と ATM プロトコルスタック概念図

ATM では、電話、データ伝送、動画像転送などの多種多様なトラヒックを同一ネットワーク上で同時に転送する。ATM ネットワーク上で伝送するトラヒックは、CBR(Constant

Bit Rate)、RT-VBR(Real Time Variable Bit Rate)、NRT-VBR(Non Real Time Variable Bit Rate)、UBR(Unspecified Bit Rate)、ABR(Available Bit Rate) という 5 つの ATM レイヤサービスカテゴリサービスカテゴリに分類されて扱われる。図 2.2 でサービスクラスカテゴリについて説明する。

	CBR	RT/NRT-VBR	ABR	UBR
対象とする トラヒックパターン	ユーザのトラヒックパターンが 事前に予測可能		ユーザのトラヒックパターンが 事前に予測不可能	
対象 アプリケーション	STM 回線の エミュレーション	可変速度符号化 音声/映像	ATM-LAN 内/間で のデータ転送	ATM-LAN 内部で のデータ転送
品質保証の有無	セル損失品質 セル転送遅延等 を保証	セル損失品質 セル転送遅延等 を保証	MCR は保証	品質保証なし
ネットワークに おけるリソース管理	STM 回線と同様 のチャネル提供	予測トラヒックに 基づく統計多重化	トラヒック発生に 合わせて動的制御	単に多重化するだけ リソース管理なし
申告パラメータ	PCR	PCR,SCR,MBS	PCR,MCR	なし

PCR … Peak Cell Rate ピークセルレート
SCR … Sustainable Cell Rate 維持セルレート
MBS … Muximum Burst Size 最大バースト量
MCR … Minimum Cell Rate 最小セルレート

図 2.2: ATM ネットワークのサービスカテゴリ

CBR サービスは、サーキット・エミュレーション用のサービスリアルタイム性が要求される。VBR サービスは、送信レートにバースト性があり、可変するトラヒックを対象にしたサービスであるが、アプリケーションの違いによって更に 2 つのサービスクラス、RT-VBR(Real Time VBR) と NRT-VBR(Non Real Time VBR) に分かれる。RT-VBR はリアルタイム性を要求される動画像転送などのサービスに、NRT-VBR はリアルタイム性の必要のないコネクション型データ転送サービスに使われる。ABR サービスは UBR と同様にリアルタイム性がないが、UBR よりも信頼度を要求されるアプリケーション向きのサービスである。UBR サービスは、ベストエフォートサービスで、リアルタイム性は要求されない。

ATM では、これらのサービスカテゴリにしたがって、QoS を保証するものとされる。よって、交換機は申告パラメータの範囲内でトラヒックを調節して、QoS 保証をしなければならない。

2.2.2 ソフトウェア処理による交換

従来のATM交換の研究の主流は、交換処理をハードウェア処理で実現する、超高速ハードウェアスイッチにあった。それは、高速な交換処理が可能なことが高品質な伝送につながるからである。しかし、チップに専用処理内容を焼き付けてしまうハードウェアでは、交換処理を行なうためのアルゴリズムを状況によって焼き付け直すわけにいかない。そのような状況を判断して処理を変えるような複雑なアルゴリズムをチップ化すればいいかもしれないが、現実的にそのようなことは困難とされる。

そこで、最近ではATM交換機に柔軟性を与えるために、交換処理をソフトウェア処理で実現する、ソフトウェアスイッチの研究も行なわれる[5]。ソフトウェアスイッチは、高速セルスイッチングをするという面では、ハードウェアスイッチには及ばないが、速度と柔軟性のトレードオフを考慮することができる。ハードウェアスイッチでは、高速セルスイッチングを行なうことで、ATMの特徴であるQoS(サービス品質)が守られたデータ通信を実現しようとするが、ソフトウェアスイッチでは、高速性の代わりに、得られた柔軟性によってQoSを実現する。

2.2.3 共有メモリ型

ATM交換機のバッファ方式は主に、共通リソース型(時分割型)と空間分割型に分けられる。(図2.3参照。)共通リソース型には、共通バス型(出力バッファ型)と共有メモリ型があり、空間分割型には、入力バッファ型、入出力バッファ型、クロスポイントバッファ型がある。共通リソース型は時分割にセルスイッチング処理を行なうので交換機内部の衝突が無い。反面、メモリアクセス速度の限界から、実現可能スループットが制限される。一方の空間分割型は入力回線からのセルを多重化せずに宛先毎にスイッチングするので、共通リソース型に比べてメモリアクセス速度による制限が少ない上、より高速・高スループットを実現できるが、出力競合が起きるとスループットが低くなる[1][2][3][4]。

高速性という面では空間分割型の方が優位であるが、ある程度の速度性能が出せれば良いと言うなら、共通リソース型でも速度面で全く問題はない。

本研究では、共通リソース型の共有メモリ型を採用する。採用の理由は、

- (上記の理由から)速度性能面は問題にならない
- 基本的なアーキテクチャは簡単に変更可能

- 異なるサービス要求を処理可能
- マルチキャスティングやブロードキャスティング等も比較的簡単に実現可能
- メモリの使用効率が一番良い

からである。図 2.4 に、共有メモリ型 ATM 交換機の概念図を示す。

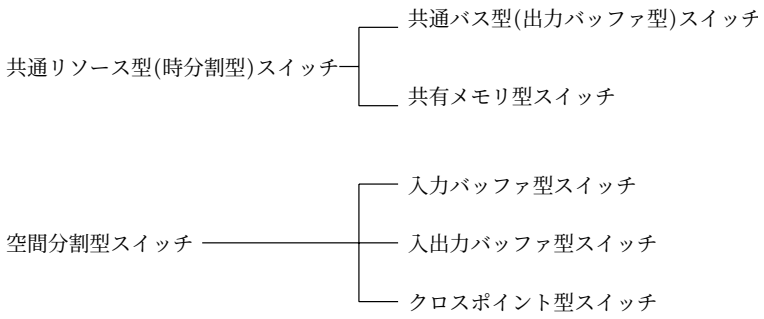

図 2.3: ATM 交換機のバッファ方式

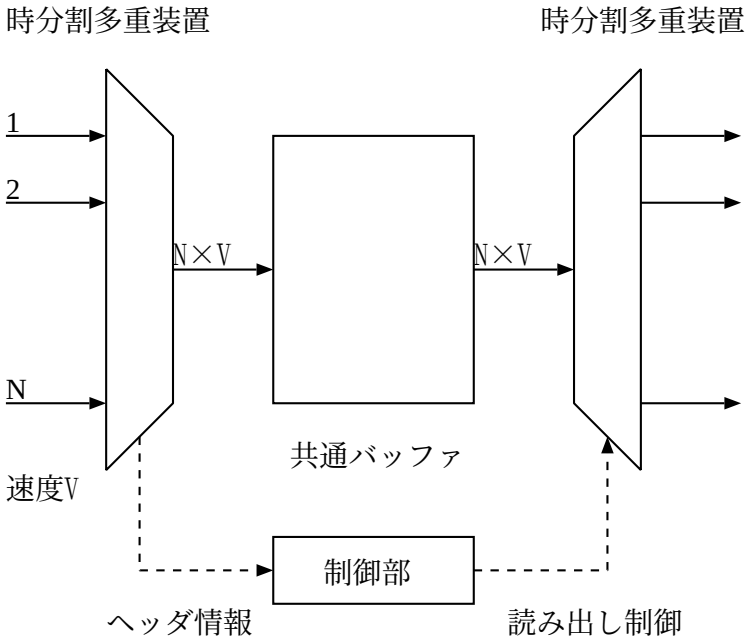

図 2.4: 共有メモリ型 ATM 交換機の概念図

2.2.4 マルチスレッド型プロセッサ

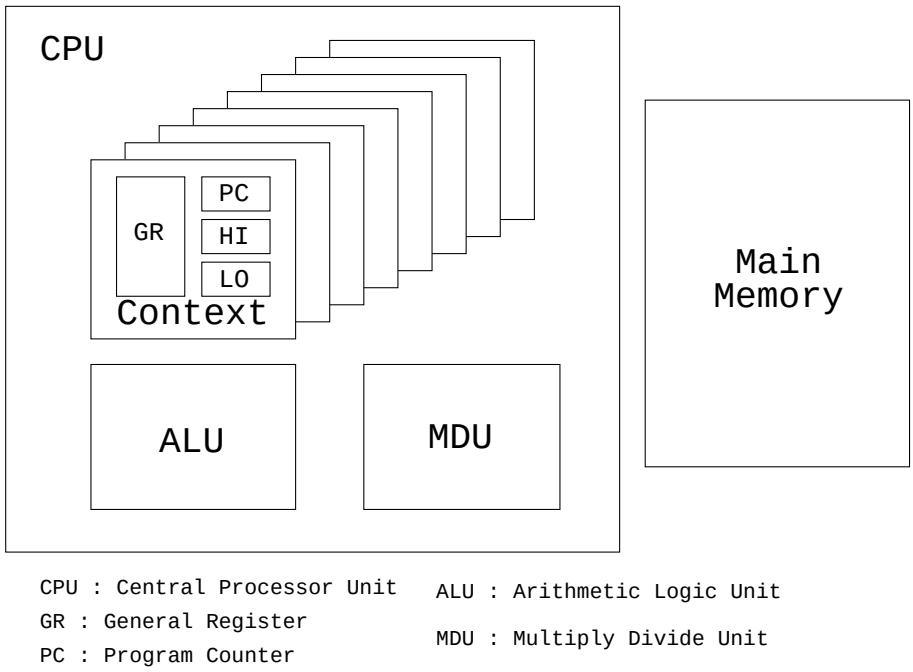

図 2.5: マルチスレッド型プロセッサの概念図

マルチスレッド型プロセッサの特徴は、図 2.5 に見られるように、

- ハードウェアコンテクストを複数持つ

ことである [7]。よって、データの流れるデータパスをパイプライン化することで、マルチスレッド型プロセッサは、コンテクストスイッチのオーバーヘッド無しに、クロック毎にスレッドを切替えることが可能になる。

一般に、パイプライン方式を採用しているプロセッサ上では、ロード命令や分岐命令の実行結果を使えるようになるまでに、数クロックの遅延が生じる [8][9][10]。このため、下記のような命令列では、ロード命令のすぐ直後にロード先のレジスタの値を使用する命令がある。言い換えると、ある命令の結果を 1 クロック以上待たなければならない次の命令がある、というような互いに依存関係を持つ命令がパイプラインステージ内に存在している。

例えば次の例題を考える。この例題を実行したときのパイプラインの様子を図 2.6 に示す。

```
lui     $1, 100         /* 即値 100 を$1 にロードする */
lw      $2, 1000($1)    /* メモリ[1000+$1] の値を$2 にロードする */
beq     $2, $0, 10000   /* $2 = $0 ならば、命令列の 10000 番地に分岐する */
```

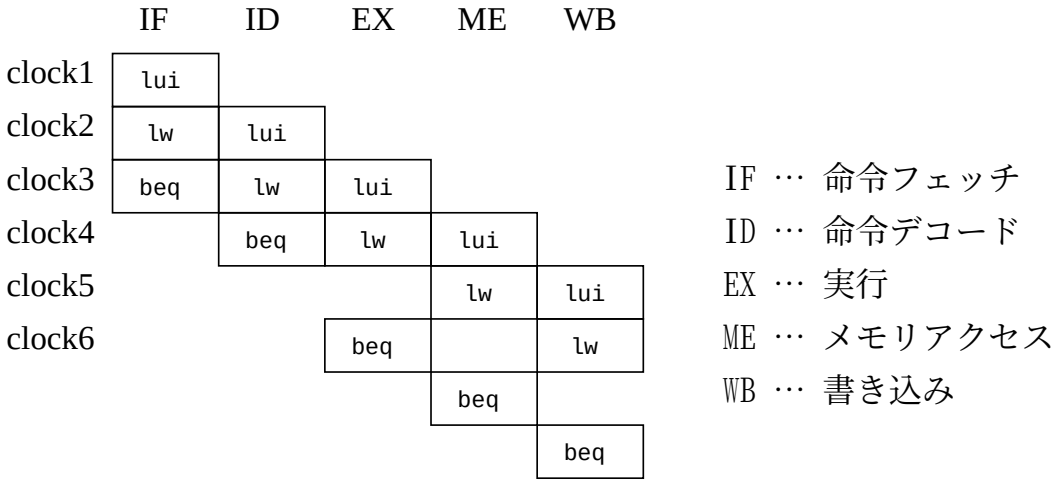

図 2.6: 例題を実行したときのパイプラインの様子

上述の命令を正しく処理するために遅延を生じさせる機構としては、プロセッサ側が、結果を使われる命令が結果を出すまで、結果を使う命令をストール (停止) させるインターロック機能がある。または、コンパイラ側が無効命令 (nop) をロード命令の次に挿入するなどして、バブルを入れて次の命令が実行されるのを遅れさせる等の方法が適用される。

しかしこのような遅延は、命令毎にかかる平均クロック数 (CPI) を増加させてしまい、結果的にスループット等の性能面に影響がでてしまう。

一方、パイプライン方式を使ったマルチスレッド型プロセッサでは、パイプラインステージ数以上のスレッドを処理する場合、各ステージに別々のスレッドの処理を行なうことができる (図 2.7 参照)。つまり、パイプラインステージには依存関係のない命令が入っているので、スレッドは遅延を生じさせることなくスムーズに処理を行なうことができる。さらに、パイプラインの各ステージが全部有効な命令で埋められるので、CPI が 1 となり、プロセッサ性能を最大に発揮することができる (図 2.8 参照)。

本研究では、互いに依存関係を持たないセルスイッチング処理を行なうスレッドを複数走らせることで、このマルチスレッド型プロセッサの特徴を活かす。

マルチスレッド型がスイッチング処理に向く理由は次の通りである。

- 仮想回線毎の処理要求への CPU 資源の割り当てがクロック単位で可能になる。
- 共有メモリに対するスレッド毎のアクセスは時分割に処理されるので、競合が無い。よって、プロセッサのクロックとメモリのスループットを適当に設計すれば、簡単な構成で高性能が期待できる

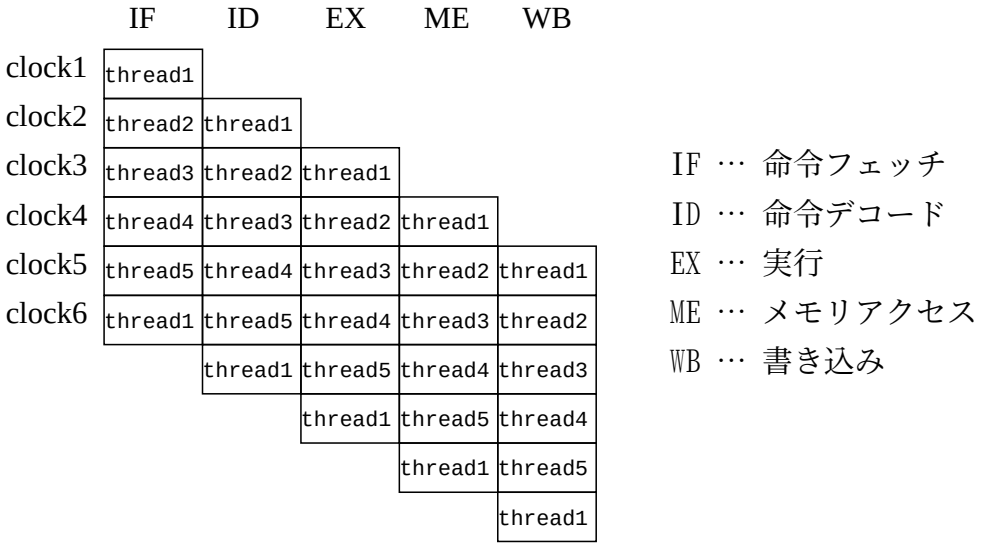

図 2.7: マルチスレッド型プロセッサでのスレッド処理と時間の関係 1

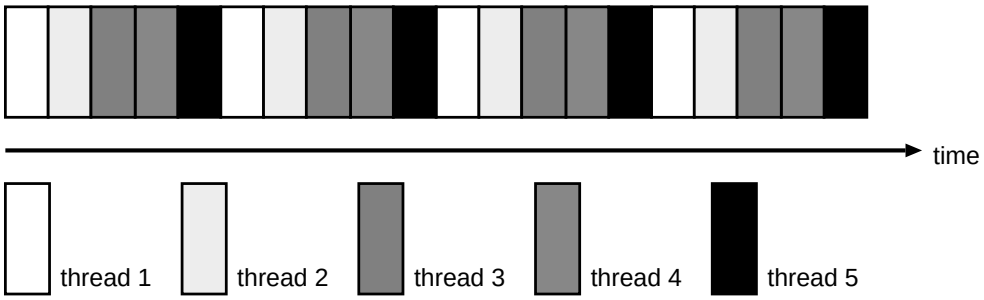

図 2.8: マルチスレッド型プロセッサでのスレッド処理と時間の関係 2

- 仮想プロセッサに相当するハードウェアコンテクストが複数あるので、多様な仮想回線への資源割り当てが、性能を低下させずに柔軟に行うことができる。

以上から QoS に応じた柔軟な制御が実現できると見込まれる。

2.2.5 RISC とパイプライン

RISC とは、縮小命令セットコンピュータ (Reduced Instruction Set Computer) のことで、CISC(複合命令セットコンピュータ：Complex Instruction Set Computer) とは反対の立場の考えである。複雑な命令を数多く備えている CISC アーキテクチャは、マイクロコードを少しずつ追加してさらにきめ細かい演算を可能にする形で進化してきたが、基本的な簡易命令しか持たない RISC アーキテクチャは、命令数を減らすことによって、制御系を簡単にし、ハードウェアの設計を容易にし、処理速度の高速化を図るような設計手法である。命令セットには、使用頻度の高い基本命令を選ぶ。RISC では固定長に命令を統一し、各々の命令は同じサイクル時間で実行される。つまり、RISC では「サイクル／命令」が固定なので、CPU をパイプライン化すると１マシンサイクル／１命令の実行速度が達成可能になるなど、パイプライン化によりもたらされる利点が生かされる。一方、CISC では命令パイプラインが非常に複雑になる上、「サイクル／命令」が可変であるので、パイプラインの利点が生かされにくい。

本研究で採用されている MIPS プロセッサは、RISC の考え方に基づいて設計されている。

2.3 本章のまとめ

本章では、ATM セルをスイッチする交換機を構築する方式として、マルチスレッド型 ATM 交換機を提案した。本方式の ATM 交換機では、ATM セルスイッチングをマルチスレッド型プロセッサを用いたソフトウェア処理で実現する。そこで、本研究を始めるにあたって前提となる知識との関係を与えた。

ATM とは、STM に代わって B-ISDN 実現に必要な伝送／交換技術として ITU-T によって採択された通信方式の一種である。ATM は回線交換とパケット交換の長所を併せ持った技術で、統計多重効果によって伝送帯域を効率良く使うことができる。ATM では全ての情報を 53 バイトの固定張パケットであるセル単位で扱う。また、高品質伝送を前提にプロトコルは簡素化されていて、プロトコルスタックで言うと、ATM セルは ATM レイヤで扱われる。ATM ネットワーク上で伝送されるトラヒックは、CBR、RT-VBR、NRT-VBR、ABR、UBR という 5 つのサービスカテゴリクラスに分類され、これらのサービスカテゴリにしたがって QoS は保証される。

ATM 交換機は上述の特徴を持つ ATM セルをスイッチングさせる機器である。従来の ATM 交換機の主流は超高速ハードウェアスイッチでセルスイッチングを実現し、高速セルスイッチングによって高品質伝送を実現しようというものであった。しかしハードウェアでは制御に柔軟性を持たせるのが困難なことから、最近では ATM 交換機に柔軟性を与えるために、ソフトウェアスイッチの研究も行なわれている。ソフトウェアスイッチでは、高速性の代わりに得られた柔軟性によって QoS を実現しようとする。

ATM 交換機のバッファ方式は共通リソース型と空間分割型に分けられる。共通リソース型では交換機内部で衝突は起きないが、メモリアクセス速度に限界がある。一方の空間分割型ではメモリアクセス速度による制限が少ないが、出力競合が起きるとスループットが低下する。本研究で採用する共有メモリ型は共通リソース型の一種である。共有メモリ型交換機では、基本的なアーキテクチャの変更が簡単、異なるサービス要求を処理可能、メモリの使用効率が良いという利点がある。

本研究でソフトウェア処理をするのに用いるマルチスレッド型プロセッサは、ハードウェアコンテクストを複数を持つ。よって、データパスをパイプライン化することで、マルチスレッド型プロセッサは、コンテクストスイッチのオーバーヘッド無しに、クロック毎にスレッドを切替えることができる。本研究では、互いに依存関係を持たないセルスイッチング処理を行なうスレッドをこのプロセッサ上に複数個走らせてこの特徴を活かす。セル

スイッチング処理にこの特徴を活かせば、仮想プロセッサに相当するハードウェアコンテクストが複数あるので、クロック毎に多様な回線対応の処理要求に対して CPU 資源割り当てを性能を低下させずに柔軟に行なうことができるようになると考えられる。

本研究で用いるこのマルチスレッド型プロセッサは MIPS アーキテクチャに基づいているものと想定する。MIPS アーキテクチャは RISC の考えに基づいて設計されている。RISC とは縮小命令セットコンピュータのことで、基本的な簡易命令しか持たないアーキテクチャを指す。複雑な命令が多い CISC とは違って、RISC では命令数を減らすことによって、制御系を簡単にし、処理速度の高速化を図る。RISC では「サイクル／命令」が固定なので、容易に CPU をパイプライン化することができ、さらに 1 マシンサイクル／ 1 命令の実行速度が達成可能になるなどパイプライン化による利点が活かされ易い。本研究では RISC によるパイプライン化の利点を活かすので、RISC の一種である MIPS アーキテクチャを採用した。

第 3 章

マルチスレッド型 ATM 交換機の設計

3.1 仕様

提案するマルチスレッド型 ATM 交換機アーキテクチャがどのような形態の交換機であるのかを述べるために、ある仕様をもったマルチスレッド型 ATM 交換機を挙げる。これを示すことで、抽象的な交換機アーキテクチャのイメージから実際の交換機をイメージしやすくなる。本節では ATM 交換機と I/O プロセッサの仕様を述べる。

3.1.1 ATM 交換機の仕様

- **回線速度×入出力回線本数：** 155Mbps × 8 本
- **バッファ方式：** 共有メモリ方式
- **スイッチプロセッサ：** MIPS アーキテクチャに基づくマルチスレッド型プロセッサ
- **スイッチプロセッサのハードウェアコンテクスト数：** 8 個
- **スイッチプロセッサのクロック周波数：** 1GHz
- **共有メモリのバンド幅：** 340Mbyte/sec 以上
- **入出力部：** I/O プロセッサを置く

これらのうち、共有メモリにアクセスするのに必要なバスのバンド幅は、入出力回線本数と回線速度から算出した。この算出方法を説明する。

1 セルあたりのスイッチング時間は入出力回線本数と回線速度から以下のようになる。

$$\frac{53[\mathrm{byte/cell}] \times 8[\mathrm{bit/byte}]}{155 \times 10^6[\mathrm{bit/sec}]} \doteqdot 2730[\mathrm{nsec}]$$

共有メモリの読み書き時間は、セルの構造 (ヘッダ 5[byte]+ペイロード 48[byte]) から、

$$2730[\mathrm{nsec}] \geq \text{共有メモリに読み書きする時間 } [\mathrm{nsec}].$$

$$\begin{aligned} \text{共有メモリに読み書きする時間} &= \text{セルの入出力時間} + \text{ヘッダの読み書き時間} \\ &= \frac{(53[\mathrm{byte}] \times 2 + 5[\mathrm{byte}] \times 2) \times 8[\mathrm{lines}]}{\text{バスのバンド幅 } [\mathrm{byte/sec}]} \end{aligned}$$

となり、これよりバスのバンド幅は

$$\text{バスのバンド幅} = \frac{928[\mathrm{byte}]}{2730[\mathrm{nsec}]} = 340[\mathrm{Mbyte/sec}].$$

というように見積もることができる。

図 3.1 に以上の仕様を満たすような ATM 交換機の物理的な構成図を示す。

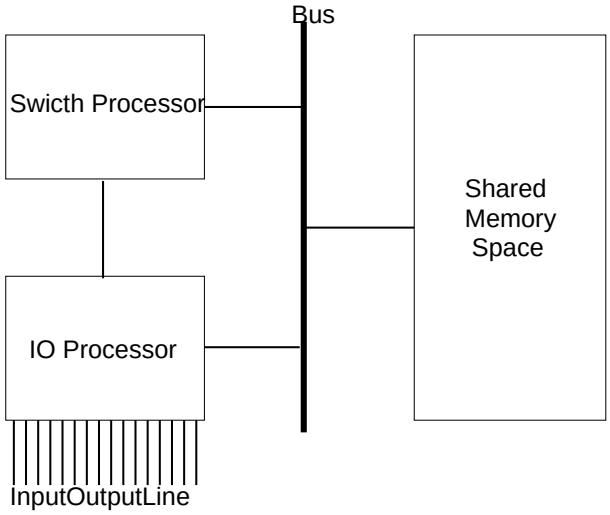

図 3.1: ATM 交換機の物理的な構成概念図

3.1.2 I/O プロセッサの仕様

- I/O プロセッサ 1 台でセルの入力と出力を担当
- 入力バッファがある
- **状態表示フラグ：** input_flag と output_flag
- **処理性能：** 6Mcell/sec

I/O プロセッサは 1 台で 8 回線全部の巡回して、入力セルの書き込み、出力セルの読み出しを行なう。このように全回線を巡回してサービスするので、他の回線を巡回中に到着したセルを保持しておくために入力バッファを持つ。その際、各入力回線にある input_ flag と出力回線にある output_ flag の値によって処理を進める。この状態表示フラグの input_flag と output_flag には、I/O プロセッサとスイッチプロセッサの両方が読み書きできる。この 16 個のフラグによって、I/O プロセッサとスイッチプロセッサ間の処理の連携をスムーズに進めることができる。

また、処理性能の値は次のように考えて算出した。ATM 交換機の仕様から、1 回線あたりの毎秒の入力セル数は

$$\frac{155 \times 10^6[\mathrm{bit/sec}]}{53[\mathrm{byte/cell}] \times 8[\mathrm{bit/byte}]} \doteq 0.366[\mathrm{Mcell/sec}]$$

となる。I/O プロセッサは、8 回線分の入出力セルを処理しなければいけないので、この値に 8 回線分の 8 倍と入出力の 2 倍をかけ合わせた値が処理性能となる。よって、目標となる処理性能値は

$$\frac{155 \times 10^6[\mathrm{bit/sec}] \times 8[\mathrm{line}] \times 2}{53[\mathrm{byte/cell}] \times 8[\mathrm{bit/byte}]} \doteq 5.85[\mathrm{cell/sec}]$$

となり、上記に 6 [Mcell/sec] と挙げた。

また、この I/O プロセッサはかなり理想的な動作をしてくれるものと想定する。すなわち、

- スイッチプロセッサの共有メモリアクセスを妨げない
- スイッチプロセッサの動作を妨げるような行為はしない

というものである。このようなI/Oプロセッサを想定するのはシミュレーションを簡単にするためである。この想定は3.1.1節で出したように共有メモリに十分なバンド幅を与えているので、スイッチプロセッサとI/Oプロセッサのアクセス競合は、簡単なバッファリング機構により解決できる。

図3.2にI/Oプロセッサの構成概念図を示す。I/OプロセッサのバッファにためられたATMセルは、Write Addressにかいてあるアドレスにセルを格納される。これが入力処理である。出力処理ではRead Addressにかいてあるアドレスからセルを読み出される。これらのアドレスはスイッチプロセッサから指示される。また、input_flagとoutput_flagは処理に関わるフラグである(後に詳しく説明する)。

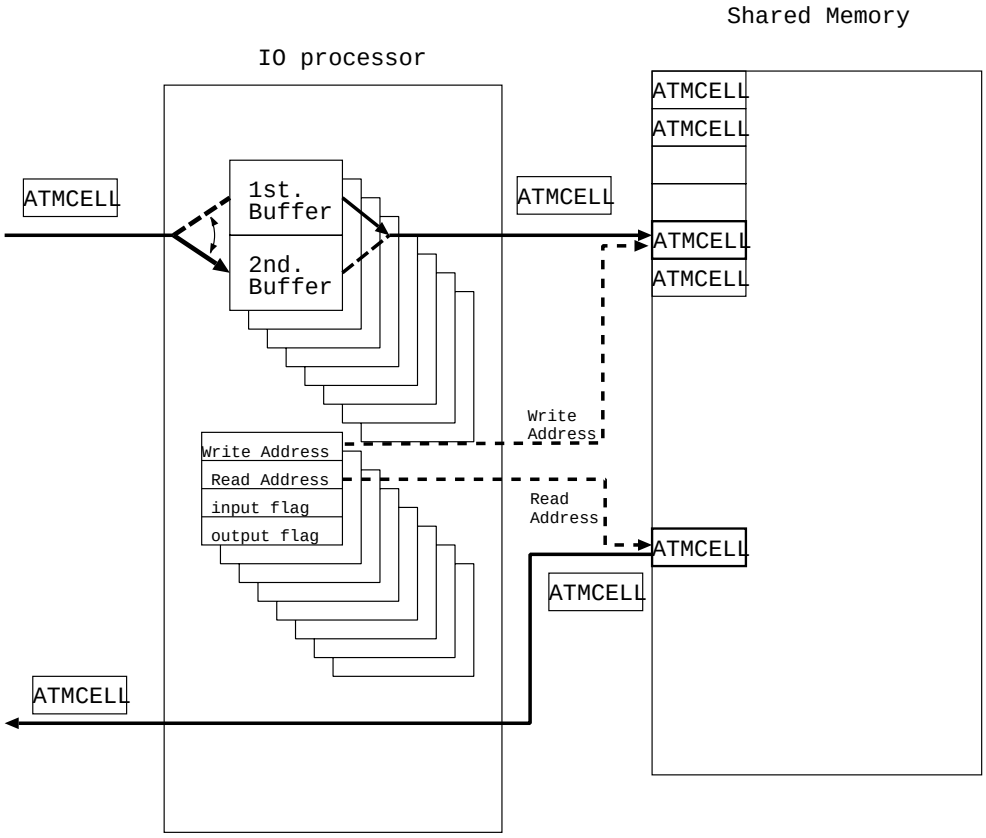

図3.2: I/Oプロセッサの構成概念図

I/Oプロセッサの構成例

本仕様のI/Oプロセッサを実現する一方法を簡単に例として示す。

まず、入力側について述べる。1つの方法は、1本の入力回線毎に入力バッファを少なく

とも 2 つ用意する。そうなっていれば、一方のバッファからセルを読み込み中に到着したセルはもう一方のバッファにためられ、ゆえに全ての到着セルがバッファにためられ、バッファ溢れで損失するということはなくなる。但し、この場合は、2.735 [μsec] の周期で 8 回線全部の入力セルを共有メモリに書き込まれないといけない。この場合は、1 つのバッファに 1 セルがためられる。また、1 回線につき入力バッファが 1 つしかない場合もある。この場合は、バッファには 1 ワード分しかためられないけど、1 ワードたまるとセルが共有メモリに書き込まれることになる。

次に、出力側について述べる。出力側でも入力側と同様に 2.735 [μsec] の周期で 8 回線全部の出力セルを処理できなければならない。この場合は出力バッファのはたらきを共有メモリがするので、出力バッファは必要ない。

この入出力セルを処理する I/O プロセッサをマルチスレッド型プロセッサでも実現することができる。本仕様の場合には、ハードウェアコンテクストを 16 個持っているマルチスレッド型プロセッサであれば、専有のハードウェアコンテクストを持つ 16 個のスレッド、すなわち、入力側に 8 個の入力処理スレッド、出力側に 8 個の出力処理スレッドをプロセッサ上に走らせて、細粒度レベルでの入出力逐次処理が実現でき、事実上、並列に 8 回線分の入出力処理が可能となる。

この処理を 16 台のシングルスレッド型プロセッサで並列に処理しようとする場合を考える。これらのプロセッサは並列に動作できるが、処理対象が共有メモリにアクセスしなければならない入出力処理なので、同期処理が必要となり、他のプロセッサとの通信や、アイドル時間が増えるので、プロセッサ性能を十分に発揮できないことになる。

シングルスレッド型プロセッサ 1 台で処理しようとする場合は、粗粒度レベルでの並列動作、つまり逐次処理で各入出力回線のセルを処理すると、各回線へのサービス時間の間隔が長くなってしまう。また、このサービス間隔を短くしようとしてコンテクストスイッチを頻繁に行なうとコンテクストスイッチのオーバーヘッドが大きくなってしまい、マルチスレッド型プロセッサ以上の処理性能がないと入出力処理が滞ってしまうことになる。

3.2 ATM 交換機の構成

次に ATM 交換機の構成を示す。図 3.3 に ATM 交換機の構成概念図を示す。交換機でスイッチング処理をする過程を簡単に説明する。

1. 入力回線から入って来たセルはI/O プロセッサによってセルの格納領域である共有メモリにストアされる。

2. スイッチプロセッサであるマルチスレッド型プロセッサは処理するセルがあると、共有メモリからセルヘッダだけを取り出し、テーブルを見て宛先決定処理をし、出力回線毎にあるサービスカテゴリに該当する出力キューにそのセルの格納番地を追加し、その後、出力スケジューリングをし、それによって決定された次の出力セルをI/O プロセッサに教える。

3. I/O プロセッサが出力キューを次に出力すべきセルの情報 (格納番地) を得て、そのセルを取り出すことで出力をする。

このようにしてセルスイッチングを実現する。出力キューは『回線本数×サービスクラス数』分だけ存在する。こうすることで、出力においてサービスクラス毎に優先順位付けをすることができる。

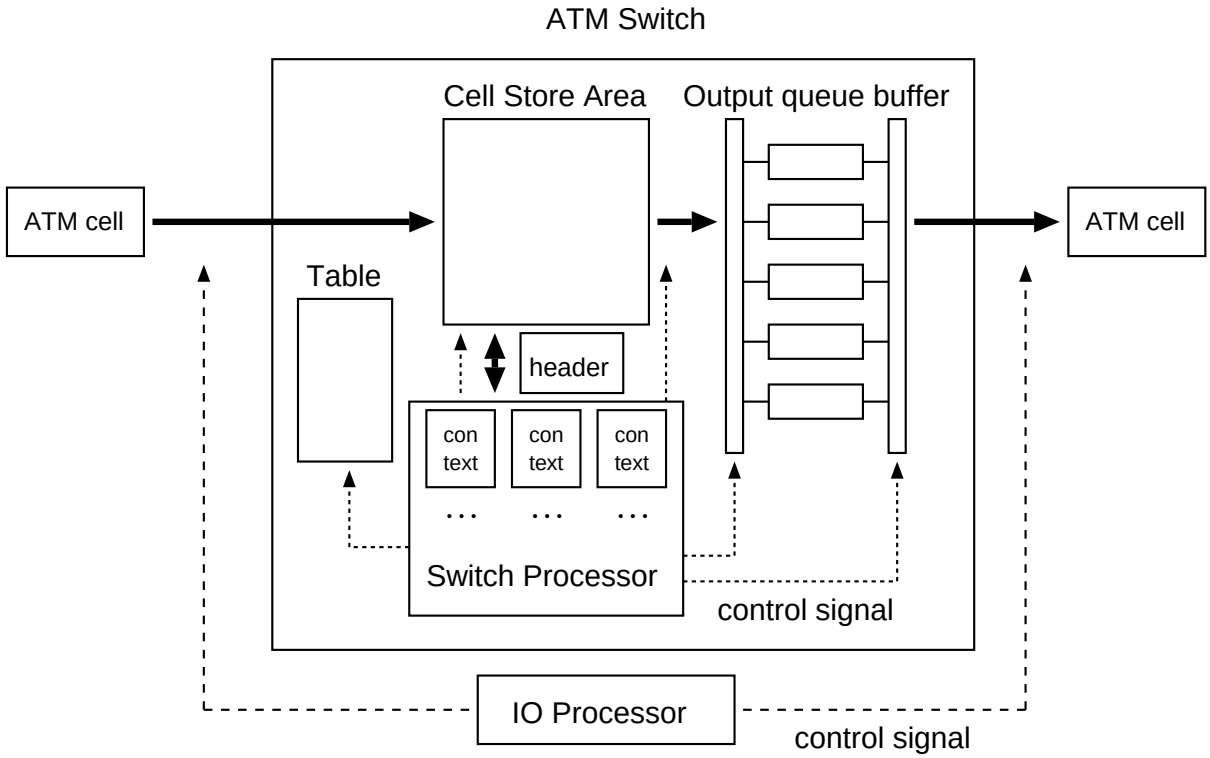

図 3.3: ATM 交換機の論理的な構成概念図

3.3 基本的な処理過程

本節では、セルスイッチングを実現するためにスイッチプロセッサで行なわれる処理とI/O プロセッサで行なわれる処理を説明する。

3.3.1 スイッチプロセッサ

スイッチプロセッサであるマルチスレッド型プロセッサでは回線本数分のセルスイッチングスレッドが連動して、ATM 交換処理を実現する。本仕様の交換機では入出力回線が各8 本であることから、8 つのセルスイッチングスレッドが必要となる。セルスイッチングスレッドは幾つかの処理部分から構成される。各々の処理が実行されるかどうかは、input_flag の値と output_flag の値に依存する。

- 前ヘッダ処理 (入力側のキュー処理)

  前ヘッダ処理は、次にI/O プロセッサが入力セルを共有メモリに書き込む場所を示すために、セルの入っていない空きセル領域から成る空きセルキューの先頭を次に進める。

  この処理は、新しく入力セルが共有メモリに書き込まれる、すなわち、そのことを示すinput_flag の値が 2 のときに実行される。下に処理過程を示す。

  1. 入力セルを書き込む場所を示す空きセルキューの先頭を次にする。
  2. 空きセルキューの新しい先頭をI/O プロセッサが見ることのできる場所に書き込む。

  この処理が実行された後、input_flag の値を 0 にする。

- ヘッダ処理

  ヘッダ処理では、ヘッダの仮想パス識別子 (VPI)、仮想チャネル識別子 (VCI) を書き換え、各回線にサービスカテゴリ毎にある出力キューにセルを入れる。この処理は前ヘッダ処理の後に行なわれる。下に処理過程を示す。

  1. 処理するセルのヘッダを共有メモリから読み出す。
  2. セルヘッダの値から宛先決定処理等をする。

3. セルヘッダの値を書き換える。
4. セルのサービスカテゴリに対応した出力キューに処理されたセルのアドレスを追加する。

- 前出力スケジューリング処理 (出力側の空きセル処理)

  前出力スケジューリング処理はセルが出力された後の空きセル (セルが出力された後の領域) を入力側の空きセルのキューに戻すために行なわれる。

  この処理は、ヘッダ処理の後に、output_flag の値が 2 のときに実行される。下に処理過程を示す。

  1. 空きセル (セルが出力された後の空いた領域) を空きセルキューに戻す。
  2. 空セルがあった出力キューの先頭を次に進める。

  この処理が実行された後、output_flag の値を 0 にする。

- 出力スケジューリング処理

  出力スケジューリング処理は、各回線毎にセルのサービスカテゴリに対応してある 5 つのキューをスケジューリングして次にセルが出力されるキューを決定するために行なわれる。この処理は QoS に応じた制御に関係する処理でもある。

  この処理は、前出力スケジューリングの後、もしくは output_flag の値が 0 のときに実行される。下に処理過程を示す。

  1. 回線毎にあるサービスカテゴリに対応した 5 つの出力キューから次にセルが出力されるキューを選ぶ。
  2. 選んだ出力キューの先頭セルのアドレスをコマンドレジスタ (I/O プロセッサがみることのできる場所) に書き込む。

  この処理が実行された後、output_flag の値を 1 にする。

1 つのスイッチングスレッドは、これらの 4 つの処理部分を実行する。各処理とフラグとの関係を示すスイッチング処理のフローチャートを図 3.4 に表す。

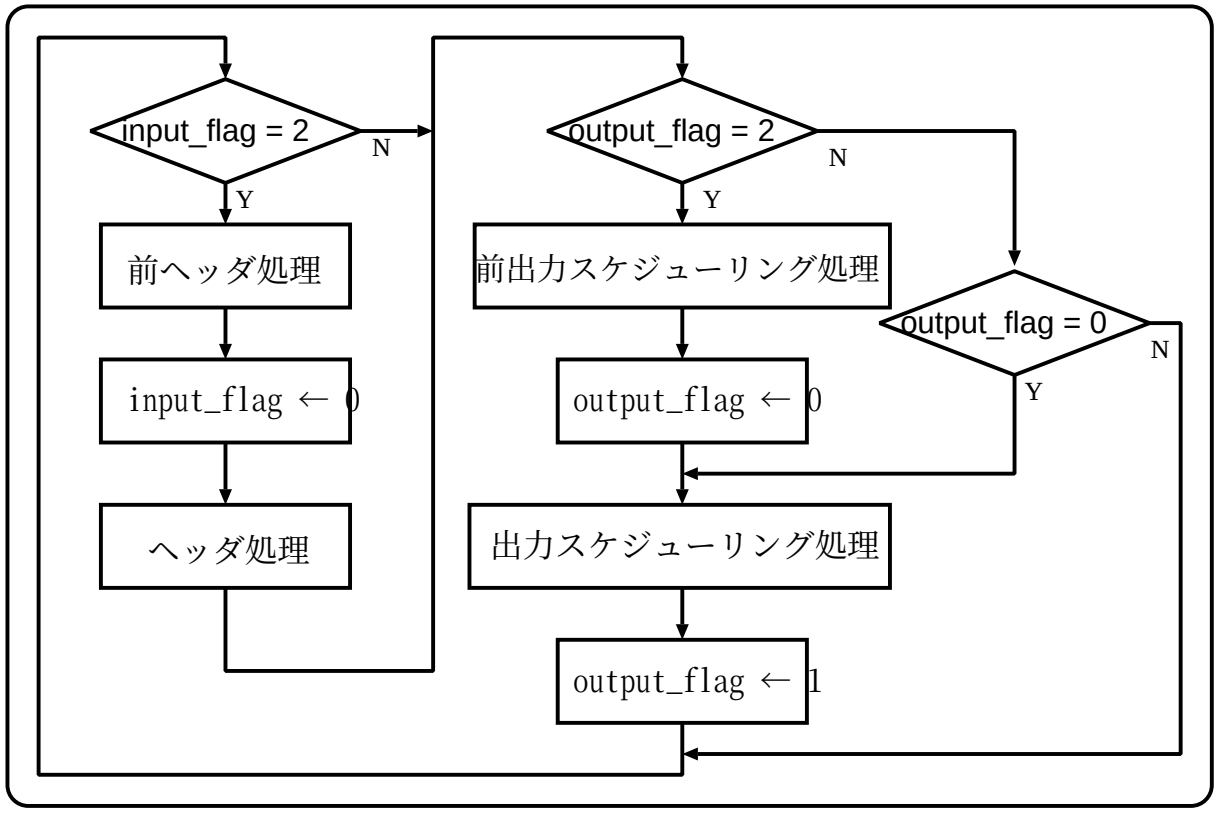

図 3.4: スイッチプロセッサにおけるスイッチング処理のフローチャート

次に、スイッチングスレッドが各回線においてどのような役割を負っているのかについて説明する。このスイッチングスレッドは各々担当する入出力回線が決まっている。8 つのスイッチングスレッドの仕事を次のように定義する。

スイッチングスレッド $n(n = 0 \cdots 7)$

- 入力回線 n の前ヘッダ処理とヘッダ処理を担当する。
- 出力回線 n の前出力スケジューリング処理と出力スケジューリング処理を担当する。

このようなスイッチングスレッドの仕事形態と分担を決定するにあたって、次のような検討をし、上述のように決定をした。

仕事形態

まずは仕事形態の検討について説明する。

- 1 セルにつき 1 スレッドを割り当てる場合
- 回線につき 1 スレッドを割り当てる場合

前者では、1 つのセル入力につき 1 スレッドが生成され、セルスイッチングが終えるとスレッドが消滅するというものである。後者では、回線毎に存在するスレッドが入力回線をずっと監視しながら入力セルを待ち、やって来た入力セルスイッチングを終えると、また入力セル待ちの状況に戻る。前者の場合は、1 セル毎に 1 スレッドが生成／消滅するので、スレッド生成／消滅のオーバーヘッドが大きいと考えられるが、入力セル待ち状況がないので、スレッドがアイドル状態になるということはない。一方、後者の場合は、入力セル待ちの状況があるので、スレッドがアイドル状態になることはあるが、設定時に 1 度スレッドを生成した後は、(特に何もなければ、) 入力セルをスイッチングする処理をループ文で廻って処理するので、スレッド生成／消滅のオーバーヘッドは考えなくてもよい。一般に、多数の同じトランザクション処理を実行するときは、多少 CPU 資源が無駄になるが、後者のように、処理内容をループ文で廻して実行する方がよいとされる。交換機のする仕事は、多数のセルをスイッチングするので、まさに、多数の同一トランザクションを処理する仕事にあてはまる。よって、本研究では後者を選択した。

仕事分担

次に、1 スレッドの仕事分担の検討を行なう際、以下のような選択肢があった。

- 各スレッドの担当を入力回線番号で固定する。
- 各スレッドの担当を入力回線番号と出力回線番号とで固定する。

前者は、担当入力回線の前ヘッダ処理とヘッダ処理、入力セルの出力先の回線の前出力スケジューリング処理と出力スケジューリング処理を行なう。この場合、後ろの 2 つの処理が入力セルに依存して出力回線番号が決定形となる。一方、後者は、担当入力回線の前ヘッダ処理とヘッダ処理、担当出力回線の前出力スケジューリング処理と出力スケジューリング処理を行なう。但し、担当する入力回線と出力回線番号は同じとする。この場合、前者と違って、完全に担当が固定する形となる。

ATM セルスイッチングでは、コネクション契約毎に 1 つの仮想回線が設定される。よって、各回線を流れるセル (トラヒック) は各仮想回線を流れるセル (トラヒック) の総和になるので、回線のトラヒックは各コネクション契約に依存したものとなる。このようなトラヒック傾向をもつことから、前者の場合に、契約トラヒック量の総和が大きい入力回線を担当するスレッドは後者の場合よりも忙しいことになる。コネクション契約によってトラヒック傾向が依存するので、ビジーなことが多いスレッドとアイドルなことが多きスレッドとの間の格差が生じることはやむを得ない。しかしながら、前者に比べ後者は、処理の半分は入力に依存していないために、このような格差が縮小すると考えられる。よって、本研究では後者を選択した。

3.3.2 I/O プロセッサ

I/O プロセッサは単純にセルの入出力動作を行なう。まず、入力では入力バッファにためられた入力セルを共有メモリの指定されたセルの格納番地 (Write Address) に書き込む。この指定される格納番地 (Write Address) は、スイッチプロセッサが I/O プロセッサ内のレジスタに書き込む。出力では、セルの格納されている共有メモリがいわば出力バッファとなっている。よって、I/O プロセッサは出力バッファでもある共有メモリの指定されたセルの格納番地 (Read Address) から出力回線へと読み出す。この指定される格納番地 (Read Address) もスイッチプロセッサが I/O プロセッサ内のレジスタに書き込む。

各処理をするかどうかは input_flag と output_flag の値に依存する。この input_flag と output_flag は I/O プロセッサとスイッチプロセッサの両方が読み書き可能なものとする。

すなわち、スイッチプロセッサと I/O プロセッサ間には共有メモリへのバス以外に線があって、その線を使って、格納番地の書き込みや input_flag と output_flag の読み書きが行なわれる。

簡単に input_flag と output_flag の値の意味を説明する。

- input_flag

  input_flag は入力処理に関するフラグで、$\cdots\rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 0 \rightarrow\cdots$ の順番に値が変わるものとする。

  0 … 次の入力セルを書き込む領域を指定されている状態

1 … 入力バッファにセルが到着している、すなわち新しい入力セルを書き込み可能な状態

2 … 入力バッファからセルを取り出した後、まだ入力側の空きセルキューを次に進めていない、すなわち次に入力セルを書き込む領域を指定されていない状態

- output_flag

  output_flag は出力処理に関するフラグで、input_flag と同様に…→ 0 → 1 → 2 → 0 →… の順番に値が変わるものとする。

  0 … 次の出力セルを指定できる、すなわち前のセルが出力された後なので、出力スケジューリングをして出力セルを決定しなければならない状態

  1 … 次に出力するセルが決まっている、すなわち出力スケジューリングされた後、まだそのセルが出力されていない状態

  2 … セルが出力された後の空きセル領域を入力側の空きセルキューにつないでいない状態

今、どんな状態かがこれらの値によって決まり、その状態に応じてセルの書き込み処理、読み出し処理を行なう。

次にI/O プロセッサが行なう基本的な処理過程を示す。

- 入力側の書き込み処理

  input_flag が 1 のときに、入力セルの書き込み処理を実行する。

  1. 入力セルを共有メモリ上の指定された領域に書き込み、input_flag を 2 にする。

- 出力側の読み出し処理

  output_flag が 1 のときに、出力セルの読み出し処理を実行する。

  1. 出力セルを読み出し、output_flag を 2 にする。

I/O プロセッサでは、各回線を巡回して入力側と出力側のセルを上述のように処理して、セルの入出力処理を実現する。そしてこの各処理とフラグとの関係を示すI/O プロセッサの入出力処理のフローチャートを図3.5 に示す。

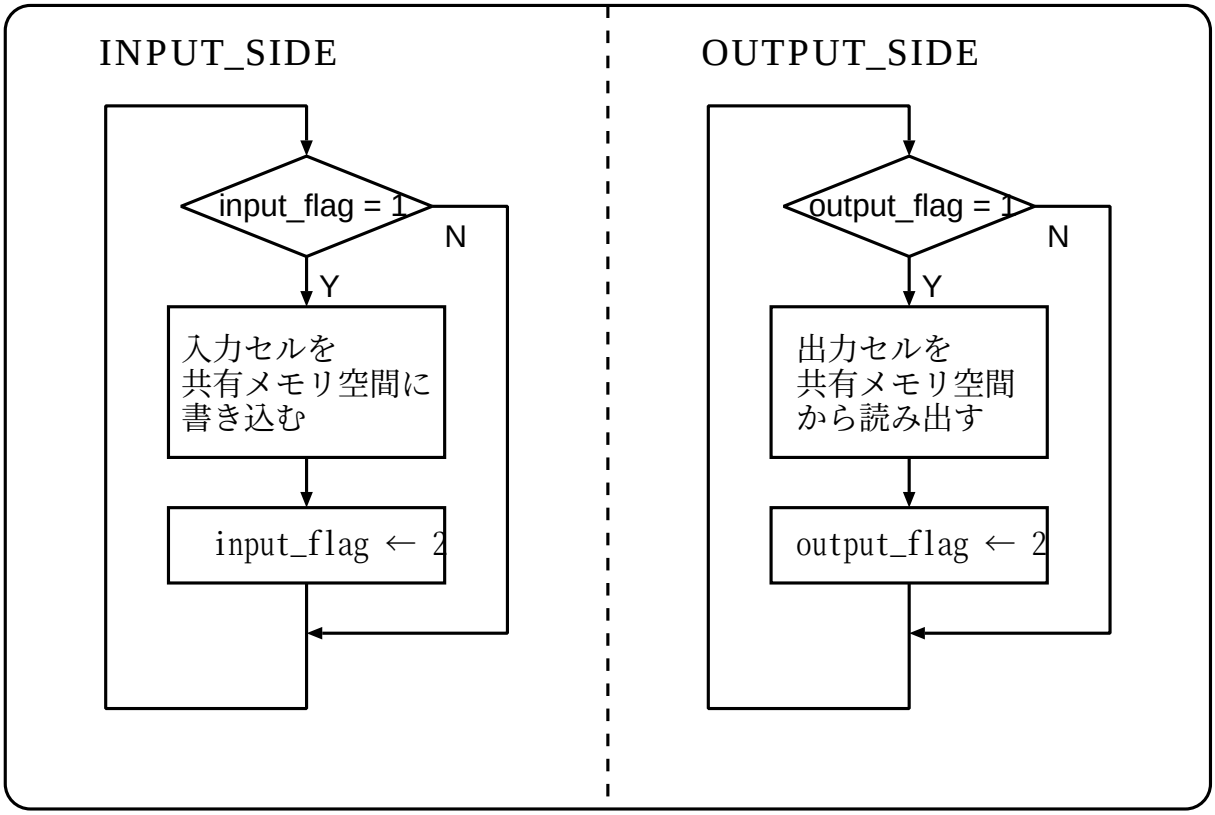

図 3.5: I/O プロセッサにおける入出力処理のフローチャート

3.4 排他的な共有メモリアクセスと交換機と入出力の連動

前節では交換機の内部処理とI/Oプロセッサの処理について説明した。本節ではこれらの処理がどのように連動して処理するのかを説明する。

図3.6に、1つのスレッド処理中にある共有メモリアクセスを示した。この図からわかるように、共有メモリアクセスはヘッダ読み出しとヘッダ書き込みしかない。[1]ここでいうヘッダ読み出しとヘッダ書き込みでは、実際ではVPI(8 or 12 bit)、VCI(16 bit)しか扱っていない。よって、セルスイッチングスレッドの処理中のメモリアクセス部分が高々ロード／ストア4回で読み書きが完了できるとみなせ、セルスイッチングスレッドの他の処理部分はI/Oプロセッサの入出力処理によるメモリアクセスとオーバーラップできる。

したがって、図3.7のように、スイッチプロセッサ側の共有メモリアクセスとI/Oプロセッサ側の共有メモリアクセスの時間軸上での関係をあらわすことができる。つまり、スイッチプロセッサとI/Oプロセッサは排他的に共有メモリアクセスをしながら処理の連動を進めていくのである。

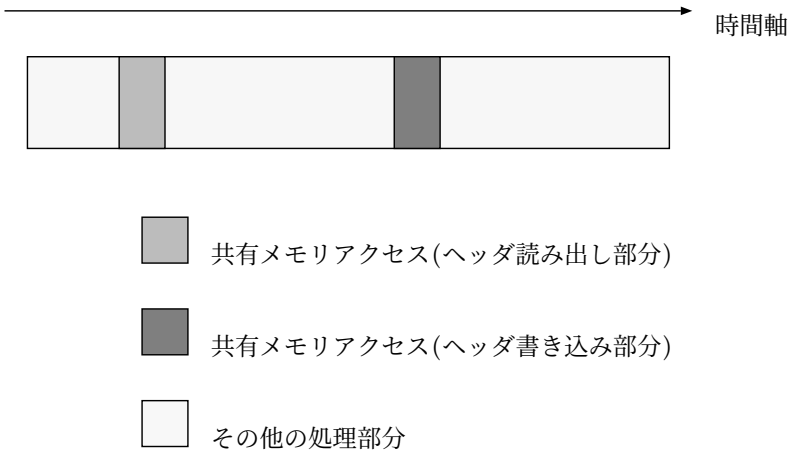

図3.6: スレッド処理中の共有メモリアクセス

[1]キューの管理は、共有メモリとは別のスイッチングプロセッサのアクセスするメモリ領域で行なわれる。

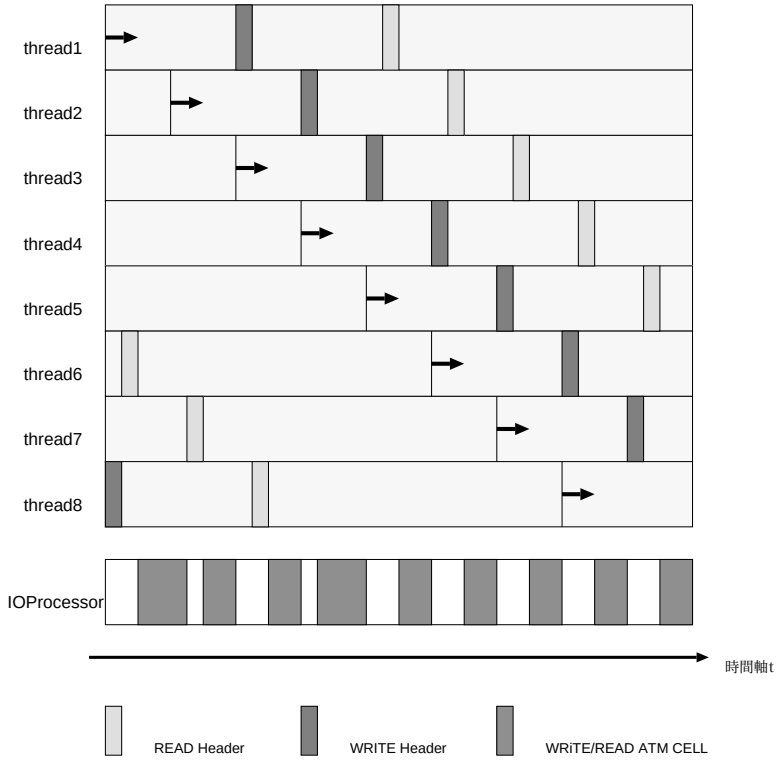

図 3.7: 共有メモリアクセスの交換機側の処理と入出力側の処理の時間的関係

3.5　スイッチプロセッサとソフトウェアの関係

図 3.8 に、スレッドのデータが使用する共有メモリとマルチスレッド型プロセッサ内の各資源 (ALU,REGISTERS,MAIN MEMORY) を示す。

スイッチプロセッサにおいて、各スレッドは１つのハードウェアコンテクストを与えられ、その中にある 32 個の汎用レジスタ (GR)、1 個のプログラムカウンタ (PC)、2 個の乗除算レジスタ (HI と LO) を用いて処理をする。本仕様のマルチスレッド型プロセッサにはハードウェアコンテクストが 8 つあるので、ゆえにプロセッサには 8 つのスレッド各々にハードウェアコンテクストが与えられることになる。

スイッチプロセッサにあるメインメモリには、低位メモリアドレスにプログラムのコード (PROGRAM CODE)、データ (DATA) が置かれ、高位メモリアドレスにスタックが置かれる。このスタックはスレッド毎に１つ持たないといけないので、この場合は 8 つのスタックが高位部に置かれる。各スタックは高位メモリアドレスから低位メモリアドレス方向に伸張する。

具体的にメインメモリに入るデータの説明をする。プログラムコードにはセルスイッチングプログラムが入り、これをスレッド間で共有することになる。データにはプログラムで宣言されたグローバル変数が置かれ、プログラムコードと同様にスレッド間で共有される。データには、セルスイッチングに必要なテーブルや、スレッド間で共有する変数、共有メモリ空間のセルの入る領域を管理するための情報 (空きセルキューと出力キュー) などが入る。各スレッドが 1 個ずつ持つスタックにはローカル変数のようなローカルデータが置かれる。(ゆえに、スレッド毎にスタックを 1 個持つ必要がある。)

共有メモリ空間はセルを格納する場所で、各スレッドはセルヘッダの情報を読み書きする。(共有メモリ空間の管理情報はメインメモリのデータ部に置かれる。)

このように資源割り当てされたプロセッサ上に置かれた 8 つのセルスイッチングスレッド各々は、パイプライン化された CPU の各パイプラインステージに収まりながら次々に処理を進める。すなわち、各ハードウェアコンテクストに入っているレジスタとメインメモリのコードとデータを用いて、CPU で演算を行なってセルスイッチングを実現するのである。

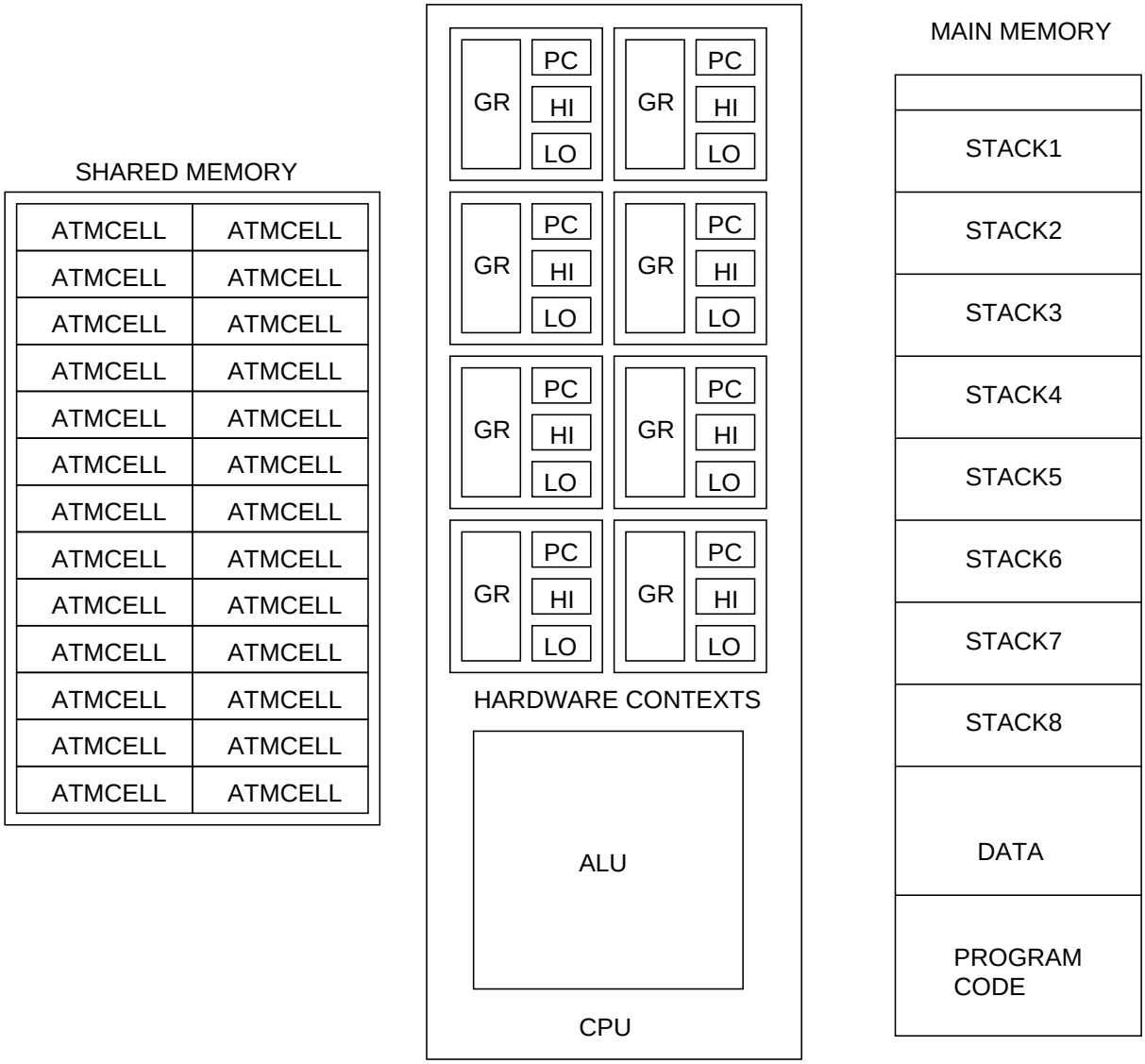

図 3.8: 共有メモリとマルチスレッド型プロセッサ上のデータの置き場所

3.6 本章のまとめ

本章では、本研究で実現するマルチスレッド型 ATM 交換機の設計について説明した。

まず、ATM 交換機は 155Mbps の回線を 8 本接続されるものとし、セルスイッチングはマルチスレッド型プロセッサを用いたソフトウェア処理で実現する。このマルチスレッド型プロセッサはハードウェアコンテクストを 8 つ持っているものする。各々のハードウェアコンテクストは仮想プロセッサに相当するので、仮想プロセッサ毎に 1 つのセルスイッチングプログラムを動かせば、各回線にセルスイッチングを行なう仮想プロセッサを 1 つ持つとみなすことができる。また、ATM 交換機のセルの入出力処理は I/O プロセッサが担当する。この I/O プロセッサは各回線の入力側と出力側にある 16 個のフラグを見て入出力処理を進める。

このセルスイッチングの実体となるスレッドプログラムは、入力側のキュー処理を行なう前ヘッダ処理部分、ヘッダの書き換え処理を行なうヘッダ処理部分、出力側の空きセル処理を行なう前出力スケジューリング処理部分、出力セルを決定する出力スケジューリング部分から構成される。前ヘッダ処理とヘッダ処理は入力回線に対して固定される処理で、前出力スケジューリング処理と出力スケジューリング処理は出力回線に対して固定される処理である。すなわち、8 つのスイッチングスレッドは各々自分の担当する入力回線と出力回線を持つ。これらの 8 つのスレッドをマルチスレッド型プロセッサに走らせてセルスイッチングを行なう。

したがって、本交換機ではスイッチプロセッサであるマルチスレッド型プロセッサと I/O プロセッサが連動して入力・交換・出力の一連の処理を行なう。その際、スイッチプロセッサと I/O プロセッサは共有メモリアクセスを起こす。よって、共有メモリアクセスの競合による衝突を防ぐために、2 つのプロセッサ間では共有メモリアクセスに対して同期をとる必要がある。すなわち、共有メモリに対して排他的にアクセスしながら処理を連動しなければならない。

今度はこれらについてスイッチプロセッサであるマルチスレッド型プロセッサの計算機資源の観点から眺める。スイッチプロセッサ内の 8 つのハードウェアコンテクストは各々 1 つのスイッチングスレッドに与えられる。メインメモリには、低位部にスレッドプログラムのコードとデータが、高位部にスタックがある。コードとデータはスレッド間で共有されるものであるが、スタックはローカル変数が入るのでスレッド毎に 1 つ持つ必要がある。したがって、この場合スタックは 8 つ存在しなければならない。このように資源割り

当てされたプロセッサ上で、8 つの各スレッドはパイプライン化された CPU の各ステージに収まりながら次々と処理を進め、こうしてセルスイッチングは実現される。

第 4 章

有効性の検証

マルチスレッド型プロセッサを用いたソフトウェア処理で ATM 交換機の有効性を検証するために、本研究では、ソフトウェアシミュレーションを行う。

4.1 シミュレーションの方法

シミュレーションには幾つか方法がある。例えば、実際に実験をするのが困難なものをモデル化して疑似的に実験をするシミュレーションや、ハードウェアの動作を模倣するシミュレータ上に実際のハードウェア上でも動くようなプログラムを動かすようなシミュレーションなどがある。

本研究においては、マルチスレッド型プロセッサが我々の研究室で開発段階にあり、実際に動くマルチスレッド型プロセッサが手に入らないという状況である。よってシミュレーションは、MIPS アーキテクチャの命令セットを実行するプロセッサの動作を模倣したアーキテクチャシミュレータ上にセルスイッチングプログラムを動かして行なう。

ソフトウェアシミュレーションによって検証する項目は以下の通りである。

1. マルチスレッド型プロセッサによるソフトウェア処理の有効性
2. シングルスレッド型プロセッサとの比較
3. 8 段パイプラインのときのソフトウェア処理
4. 同期処理

これらに対してシミュレーションを幾つか行なうのだが、各シミュレーションの方法に共通する手順を下に説明する。

1. まず、MIPS アーキテクチャに基づいたプロセッサ上でセルスイッチングを行なうスレッドのプログラムを作成する。そのスレッドプログラムは 3.3.1 に示した 4 つの処理からなるが、そのうち、出力スケジューリング処理は QoS に応じた制御をしなければならないという位置付けにあるので、本来ならトラヒックに応じて制御アルゴリズムが変わらなければいけないものであるが、現段階のシミュレーションでは可変アルゴリズムを用意する必要はないので、出力スケジューリングアルゴリズムは、サービスカテゴリクラスのうち、優先的に処理されなければならないものから順に出力順位を決定するというものにした。

2. そのプログラムを mips-gcc(GNU C Compiler(v2.7))[11] というクロスコンパイラを使用して、最適化をするようにコンパイルする。

3. 本来は、コンパイル後の実行ファイルをマルチスレッド型プロセッサで実行して評価を行ないたいところである。しかし、本研究で想定するマルチスレッド型プロセッサが手に入らないので、想定するプロセッサをシミュレートするシミュレータを用いて、評価を行なう。具体的には、マルチスレッド型プロセッサのシミュレータ上に、作成したスイッチングプログラムを走らせて、アセンブラレベルでの実行された命令数 (ステップ数) を算出する。その際に、次の主な 3 処理をした場合 ($\surd$) としない場合 (空白) の計 8 通りについてステップ数を計測するものとする。

   - 処理 1 … 入力側のキュー処理＋ヘッダ処理
   - 処理 2 … 出力側の空セル処理
   - 処理 3 … 出力スケジューリング

   このように分けた理由は、この 3 処理はお互いに依存し合わない関係にある最大の処理単位であるからである。

このようにして算出された代表的な 8 つの処理パターンの場合の実行命令数を基にして、各検証事項について考察する。

また、セルスイッチングスレッドはマルチスレッド型プロセッサとシングルスレッド型プロセッサで実現する場合、コンテクストスイッチの起こる粒度が異なるようにした。す

なわち、マルチスレッド型プロセッサでは、1 命令毎に次のスレッドの命令が実行されるように切替えをして細粒度レベルでのコンテクストスイッチが起きるようにしたが、一方のシングルスレッド型プロセッサでは、1 セル分のセルスイッチングが完了すると次のスレッドにコンテクストを渡して次のスレッドのセルスイッチング処理が実行されるというような粗粒度レベルのコンテクストスイッチ、言い換えると逐次処理で実現している。

このようにした理由について述べる。マルチスレッド型プロセッサでは、コンテクストスイッチによるレジスタの退避・復元によるオーバーヘッドがかからないので、細粒度レベルでコンテクストスイッチが起きても特に問題はない。一方のシングルスレッド型プロセッサでは、コンテクストスイッチによるレジスタの退避・復元によるオーバーヘッドが単純に汎用レジスタ 32 個と PC1 個と HI と LO の 2 個のレジスタセットにおいてかかるとすれば、ロード／ストアが 35 回ずつ必要になる。セルスイッチングスレッドのプログラムはそれほど大きなプログラムではないので、1 セルスイッチングをするのに 70 命令分のオーバーヘッドがかかってしまうのは具合が悪いので、本シミュレーションにおいてはコンテクストスイッチのタイミングを 1 セルスイッチング分の処理が終わった時にした。

セルスイッチングスレッドの内部構造を本シミュレーションでは次のようになっている。

＜マルチスレッド型プロセッサの場合＞

```
void thread(void)
{
   while(1)
    cellswitchingprocess();
}
```

＜シングルスレッド型プロセッサの場合＞

```
void thread(void)
{

    cellswitchingprocess();
}
```

このように定義されるスレッドプログラムをマルチスレッド型プロセッサでは回線本数分走らせる。マルチスレッド型プロセッサの場合にループ文でスレッドにセルスイッチング処理をさせることが可能なのは、各スレッド毎に 1 つのハードウェアコンテクストを専有できるからである。一方のシングルスレッド型プロセッサでは、全スレッドで 1 つのハードウェアコンテクストを共有するのでマルチスレッド型プロセッサのようにループ文でセルスイッチング処理をさせることはできない。

それではこのようなセルスイッチングプログラムを使用してシミュレーションを行なう。

4.2 シミュレーション 1

シミュレーション 1 では、単純な方法で本仕様の ATM 交換機においてソフトウェア処理によるセルスイッチングが実現可能であることを示す。

4.2.1 方法

スイッチングプログラムを mips アーキテクチャの基本的なセットである mips1 に基づいてコンパイルし、それをシミュレータ上に動かす。mips1 では、プロセッサにインターロック機構があることを想定していないので、コンパイラが依存関係のある命令間に無効命令 (nop) を挿入して遅延を入れる [10]。シミュレータ上にスイッチングプログラムであるスレッドを走らせ、実行された命令数 (単位：ステップ) から無効命令数を引いて正味の実行命令数を算出し、さらにその数から実行にかかる時間を導出し、その値によって本仕様の交換機性能を満たせるかどうかを評価する。

注意として、この方法でシミュレートするセルスイッチングプログラムは、他にスレッドが走っている場合を想定していない。あくまで、単独でスレッドプログラムが走っているときのかかる実行命令数を調べて、単純に評価することに主眼を置いている。

4.2.2 結果

マルチスレッド型プロセッサを用いた場合のソフトウェア処理のセルスイッチングをシミュレートし、実行命令数、無効命令数、正味命令数を下の表に示した。

処理 1	処理 2	処理 3	実行命令数	無効命令数	賞味命令数
√	√	√	208 step	25 step	183 step
√	√		166 step	21 step	145 step
√		√	155 step	12 step	143 step
√			110 step	8 step	102 step
	√	√	119 step	21 step	98 step
	√		77 step	17 step	60 step
		√	67 step	9 step	58 step
			22 step	5 step	17 step

4.2.3 考察

この算出した正味の命令ステップ数から、例えば、1GHz の MIPS プロセッサにおいて 1 クロック 1 命令で動くと仮定すると、その処理にかかる時間は、ステップ数 [nsec] で表すことができる。このことより、処理 1〜3 全部を行なうフルの状態でスイッチングするときは、1 セル当り正味 183[nsec] かかる。

まず、本交換機の仕様 (155Mbps × 8 本) から、1 セルをスイッチングさせるのに最悪かかってよい時間は、

$$\frac{53[\text{byte}] \times 8[\text{bit/byte}]}{155 \times 10^6[\text{bit/sec}]} \doteq 2730[\text{nsec}]$$

である。この時間内に 8 回線のセルをさばかないといけないので次式が成り立たないと、仕様を満たすことができない。

$$2730[\text{nsec}] \geq 1 \text{ セルあたりのスイッチング時間 } [\text{nsec}] \times 8[\text{本}][\text{nsec}]$$

スイッチプロセッサが 8 セルをスイッチングするのにかかる時間は、算出した正味命令数 (但し、最も多く実行される処理パターンでかつ正味命令数の多い場合で) より、

$$183[\text{nsec}] \times 8[\text{本}] = 1464[\text{nsec}]$$

である。さきほどの式にあてはめると、

$$2730[\text{nsec}] \geq 1464[\text{nsec}]$$

という関係が成り立つことがわかる。但し、この場合は、他のスレッドが走っている場合を想定していない、すなわち、他のスレッドと共有メモリアクセス時に起こり得る競合のことは考えていない。しかし 8 回線分合わせて 1200step 分の余裕があるので、競合を起こさないようなロック機構をこのスレッドに導入したとしても、競合を防ぐためのオーバーヘッドは十分に小さいと考えられる。

よって、本仕様の交換機がソフトウェア処理によっても十分に可能である見通しを示すことができた。

4.3 シミュレーション 2

シミュレーション 2 では、シミュレーション 1 との比較として、シングルスレッド型プロセッサを用いた場合のマルチスレッドプログラミングによるセルスイッチングの性能と、マルチスレッド型プロセッサを用いた場合の性能とを比較する。

4.3.1 方法

シミュレーション 2 では、シミュレーション 1 で作成した mips プロセッサ上でセルスイッチングを行なうスレッドプログラムを、シングルスレッド型プロセッサ上で動くように改造し、それをシミュレータ上で動かす。但し、シミュレーション 2 では、シングルスレッド型プロセッサ上で動かすという想定なので、シミュレーション 1 では考慮しなかった遅延を考慮する必要がある。したがって、コンパイラが遅延を考慮して挿入してくれた nop 命令を省く必要はここではない。また、mips1 ではコンパイラ側が遅延を考慮してアセンブラコードを生成するので、新たに遅延計算をする必要はなく、実行命令数がそのまま正味命令数となる。

4.3.2 結果

シングルスレッド型プロセッサを用いた場合のソフトウェア処理のセルスイッチングをシミュレートし、実行にかかった命令数を算出した。処理内容の各行はシミュレーション 1 と同じである。今回の結果には、実行された命令数の他に、シミュレーション 1 との増分の正味命令数も示した。

処理 1	処理 2	処理 3	実行命令数	増分命令数
√	√	√	247 step	64 step (35 %)
√	√		191 step	46 step (32 %)
√		√	184 step	41 step (29 %)
√			127 step	25 step (25 %)
	√	√	149 step	51 step (52 %)
	√		93 step	33 step (55 %)
		√	86 step	28 step (48 %)
			29 step	12 step (71 %)

4.3.3 考察

マルチスレッド型プロセッサ上で動かした処理と同じ処理をシングルスレッド型プロセッサで実行させた場合、全ての処理においてステップ数が増えた結果となった。この要因は次の 2 つであると考えられる。

- マルチスレッド型プロセッサでは、各回線にあるスレッドは各々ハードウェアコンテクストを持っているので、各スレッドが自分の環境を整えた後 (初期化後) は、スイッチング処理だけに専念できる。一方、シングルスレッド型プロセッサでは、ハードウェアコンテクストが 1 つしかないので、各回線にあるスレッド全部でハードウェアコンテクストを共有して使うことになる。よって、マルチスレッド型の場合とは違って、コンテクストスイッチが起こる度に各スレッドは自分の環境を整え、それからスイッチング処理を行なうことになる。よって、シングルスレッド型プロセッサ上でスイッチング処理を行なうと、マルチスレッド型プロセッサに比べて環境設定の分だけ実行命令数が多くなる。

- マルチスレッド型では各パイプラインステージに別々のスレッドが入っている。つまり、互いに依存関係を持たない命令でパイプラインステージが満たされているので、無効命令 (nop) を挿入して遅延を考慮する必要がない。すなわち、有効な命令だけでスムーズに命令が実行される。一方、シングルスレッド型では各パイプラインステージに同じスレッドが入っているので、命令間の依存関係を考慮して命令順序を変えたり、無効命令 (nop) を挿入しなければならない。よって、シングルスレッド型では、マルチスレッド型に比べて無効命令 (nop) が入った分だけ同じ処理をするのに実行命令数が増えてしまう。

よって、シングルスレッド型プロセッサではマルチスレッド型の場合よりも同じ処理をするのに約 3 割も時間がかかってしまう。つまり、マルチスレッド型プロセッサを用いたセルスイッチング処理はシングルスレッド型プロセッサを用いた場合に比べると約 30%の性能向上が見られることがわかった。

4.4 シミュレーション 3

シミュレーション 1 ではマルチスレッド型プロセッサによるソフトウェア処理が有効であることを示し、本仕様の交換機がの実現可能性をもつことがわかった。そこでシミュレーション 3 では、シミュレーション 1 を少し発展させたシミュレーションをする。具体的には、シミュレーション 1 では 5 段だったパイプラインステージ数をスレッドと同数の 8 段にしてセルスイッチングスレッドを走らせた場合の実行ステップ数を算出する。

ここでは、mips プロセッサでパイプラインステージ数が 8 段である R4000 でセルスイッチングプログラムが実行されると想定する。よって、スイッチングプログラムは R4000 で使われる mips2 命令セットアーキテクチャ[9] に基づいた命令セットになるようコンパイルをし、それを R4000 プロセッサに基づいたシミュレータ上に動かす。

4.4.1 方法

mips2 アーキテクチャの命令セットに基づいたセルスイッチングプログラムであるスレッドをシミュレータ上で動かし、8 通りの場合のステップ数を算出する。シミュレーション 3 の場合でもシミュレーション 1 と同様に、マルチスレッド型プロセッサ上でのセルスイッチングを想定しているので、無効命令 (nop) を実行命令数から引いて正味命令数を算出する。

4.4.2 結果

パイプラインステージ数が 8 段のときの実行結果を次に示す。ここでもシミュレーション 1 の結果と同様、実行命令数、無効命令数、正味命令数を示す。

処理 1	処理 2	処理 3	実行命令数	無効命令数	正味命令数
√	√	√	187 step	3 step	184 step
√	√		146 step	2 step	144 step
√		√	148 step	3 step	145 step
√			104 step	2 step	102 step
	√	√	102 step	3 step	99 step
	√		62 step	2 step	60 step
		√	62 step	3 step	59 step
			19 step	2 step	17 step

4.4.3 考察

シミュレーション 1 より実行命令数は減っているが、その減った分だけ無効命令の nop 命令が減っているので、全体的に正味命令数は変わらない結果となった。正味命令数はセルスイッチング処理を行なうために必要な命令の数なので、変わらないのは当然と言えば当然でもあるが、mips1 の命令セットアーキテクチャと mips1+mips2 の命令セットアーキテクチャの差異がここではほとんど見られなかった。これはマルチスレッド型プロセッサのパイプラインステージ数が 8 段に増えても性能が変わらない、性能が落ちないということである。

しかし、パイプラインステージ数が 5 段から 8 段に増えるということは、同じクロック周波数でプロセッサを実現するのに、パイプラインステージ数が多い方が比較的容易になるということがある。よって、パイプラインステージ数が増えても正味の実行命令数が変わらないのに、実現テクノロジーが容易になる、すなわち、本仕様のマルチスレッド型 ATM 交換機の実現可能性が高まると言える。

4.5 シミュレーション 4

シミュレーション 4 では、パイプラインステージ数が 8 段のシングルスレッド型プロセッサ上でマルチスレッドプログラミングによるセルスイッチングをマルチスレッド型プロセッサを用いた場合とを比較する。

4.5.1 方法

セルスイッチングを行なうスレッドプログラムを mips2 の命令セットアーキテクチャに基づいた命令コードになるようにコンパイルし、その命令コードを R4000 シミュレータ上に実行させて、各処理における実行命令のステップ数を算出する。その算出した実行命令のステップ数に、命令間の依存関係によって起こるインターロック分の遅延サイクルを足して、正味の命令ステップ数、つまり実行時間を算出する。

命令間の依存関係による遅延は、R4000 プロセッサで想定する遅延と同様に考えて、

- ロード遅延 2 サイクル
- 分岐遅延 3 サイクル

とし、遅延計算を行なう。この 1 サイクルは 1 ステップと同じ時間分、つまり 1 [nsec] かかるものとする。

また、シングルスレッド型プロセッサでのセルスイッチングを実現する場合、本方式では 1 スレッドが 1 回線の処理を終えた時に、次のスレッドにハードウェアコンテクストの所有権が移るので、スレッド間で同期を取る必要がないので、同期をとるためのロック命令を使う必要がないものとする。

4.5.2 結果

次に実行ステップ数に遅延サイクル数を考慮した正味ステップ数を含めた結果を示す。

処理 1	処理 2	処理 3	実行命令数	遅延サイクル数	正味命令数	増分命令数
√	√	√	230 step	41 cycle	271 step	87 step (47 %)
√	√		177 step	41 cycle	218 step	74 step (51 %)
√		√	178 step	21 cycle	199 step	54 step (37 %)
√			123 step	21 cycle	144 step	42 step (41 %)
	√	√	137 step	30 cycle	167 step	68 step (69 %)
	√		84 step	26 cycle	110 step	50 step (83 %)
		√	83 step	12 cycle	95 step	36 step (61 %)
			28 step	10 cycle	38 step	0 step (0 %)

4.5.3 考察

MIPS プロセッサのパイプラインステージ数を 8 段にし、mips2 の命令セットアーキテクチャを採用すると、マルチスレッド型プロセッサに対してシングルスレッド型プロセッサでは全体的にセルスイッチングの実行時間が増えることがわかった。このことは見方を変えると、シングルスレッド型プロセッサに対してマルチスレッド型プロセッサでは最大で 83%の性能向上が見られるということでもある。ここでは、まだセルスイッチングスレッド間での共有メモリ上のデータの読み書きで起きる競合については考慮していないので、この結果が十分に有効であるとは言えないが、スレッド間でプロセッサ同期を取る必要がないようなプログラムを動かしたときには、パイプラインステージ数が 5 段の時よりも 8 段の時の方がシングルスレッド型プロセッサに対するマルチスレッド型プロセッサの有効性が格段に高まることがわかった。

4.6 シミュレーション 5

シミュレーション 3 ではパイプラインステージ数が 8 段のときのスレッドの各処理パターンの実行命令数を算出した。しかし、シミュレーション 3 ではスレッド間での競合を防ぐための同期をとる機構を考えていなかった。そこで、シミュレーション 5 ではこの同期をとる機構をプロセッサに導入し、その場合の実行命令数を調べ、その値から同期をとるための命令を追加した場合の増分 (同期をサポートするために増えたオーバーヘッド) を調べる。

4.6.1 方法

スレッドプログラムにおいて、スレッド間で共有するメインメモリのデータの中で競合が起き得るクリティカルセクションを lock() と unlock() で囲んでデータをロックしてから読み書きをしその後、ロックを解除 (アンロック) する。

```
lock();
クリティカルセクション
unlock();
```

この同期命令はスピンロックで実現される。次に示す例のように、ロックを獲得できないと獲得できるまでスピンするので、そのスピンした分だけステップ数が増えることになる。そして権利を獲得するとスピンから抜けて共有変数を処理できるようになる。このスピンは、ll と sc 命令を使って実現する。

- ll

  lw(load word) のように、メモリからレジスタに値を読む。但し、プロセッサがもっている llbit を 1 にして lw したアドレスを記憶しておく。(llbit が 1 になっているときは、そのアドレスの内容を他のスレッドに変えられたかどうかをチェックする。他のスレッドが変えてしまうと、llbit が 0 になる。)

- sc

  sw(store word) のように、レジスタの内容をメモリに書く。但し、llbit が 0 のときは、書き込みに失敗して、指定したレジスタを 0 にする。llbit が 1 で書き込みに成功した場合は指定したレジスタが 1 になる。

まず、クリティカルセクションが単純な足し算命令だけの場合のコードを示す。

```
1b:     ll      $8, %0
        addi    $8, %1
        sc      $8, %0
        beq     $8, $0, 1b
```

この場合は、ll で%0 から$8 に値を読んで、その値に%1 の値を足して、その結果を書き込むとき、他のスレッドが値を変えてなければ llbit は 1 だから sc が成功し、$8 は%0 に書き込まれる。sc が失敗すると$8 は 0 になるので、次の分岐命令で真だから、また ll 命令に戻る。この場合はセマフォを使ってないので、これだけで済む。

次に、クリティカルセクションが数命令から成る場合を示す。この場合はセマフォを使うので、ロック部分とアンロック部分が要る。最初にロック部分を示す。

```
1b:     ll    $8, %0
        bne   $8, $0, 1b
        li    $9,  1
        sc    $9, %0
        beq   $9, $0, 1b
```

ll でセマフォである%0 から$8 に値を読んで、セマフォの値が 0 でなければ他のスレッドがセマフォを獲得しているので、ll 命令に戻る。セマフォの値が 0 であれば他のスレッドがセマフォを獲得していないので、sc でセマフォを 1 にすることができる。セマフォの値を 1 にするとき、他のスレッドがセマフォの値を変えていなければ、llbit が 1 なので sc が成功し、セマフォの値が 1($9) になり、$9 が 1 になる。llbit が 0 で sc が失敗すると$9 が 0 になるので、分岐命令で真となり、ll に戻る。また、アンロックは、次の命令で実現される。

```
        sw    $0, %0
```

このようにしてセマフォの値は 0 に戻され、他のスレッドがロックできる状態になる。

本シミュレーションでは、これらの 2 つのスピンロックを使って同期処理をする。

4.6.2 結果

下にプロセッサ同期をサポートした場合の実行結果を示す。但し、この実行結果は 1 回で同期変数を獲得してロックできる、つまり、競合することなくデータのアクセス権を得られる場合のものである。増分ステップ数には、左側にプロセッサ同期をサポートしていないマルチスレッド型プロセッサに対してステップ数が増えた分 (シミュレーション 3 との比較) を右側にシングルスレッド型プロセッサと比べて減った分 (シミュレーション 4 との比較) を示す。

処理 1	処理 2	処理 3	実行命令数	無効命令数	正味命令数	増分命令数	減分命令数
√	√	√	217 step	10 step	207 step	23 step	65 step
√	√		174 step	7 step	167 step	23 step	51 step
√		√	158 step	8 step	150 step	5 step	49 step
√			115 step	5 step	110 step	8 step	34 step
	√	√	121 step	7 step	114 step	15 step	53 step
	√		80 step	4 step	76 step	16 step	34 step
		√	62 step	5 step	57 step	-2 step	38 step
			19 step	2 step	17 step	0 step	21 step

このシミュレーション結果は処理を行なえる最小のステップ数なので、実際には余分なスピン分のステップ数が加算されたものが実行ステップ数となる。

4.6.3 考察

セルスイッチング処理において最も実行される場合が多いと考えられる処理 1〜3 全部を行なう場合で比較する。するとこのフルの場合では、同期をとるために必要なロック命令を入れることで、ロック命令がないときと比べてマルチスレッド型プロセッサでは約 15%の性能低下となるが、それでもシングルスレッド型プロセッサに比べると、まだ 30%の性能向上となることがわかる。それでもこの比較はロックにかかる時間が最も少ない場合の比較なので、ロックにかかる時間が大きければ大きいほど、同期をサポートするマルチスレッド型プロセッサを用いたセルスイッチング処理ではシングルスレッド型プロセッサを用いる場合よりも受ける恩恵が少なくなってしまう。

ロックする場所はスレッドがアクセス競合する場所で、全部で 4 ヶ所ある。それらを次に示す。

- 出力キュー競合
  - 出力キューへのセル挿入 (処理 1)
    - ロックからロック解除までの最短実行命令数 … 19
    - 各回線毎に最大全スレッドが競合を起こし得る
  - セル出力後の出力キューの空きセルの後始末 (処理 2)
    - ロックからロック解除までの最短実行命令数 … 29
    - 各回線毎に最大 1 スレッドが競合を起こし得る
- 出力セルのカウンタ[1]競合
  - 出力回線の出力セルのカウンタを増やす (処理 1)
    - ロックからロック解除までの最短実行命令数 … 3
    - 各回線毎に最大全スレッドが競合を起こし得る
  - 出力回線の出力セルのカウンタ減らす (処理 3)
    - ロックからロック解除までの最短実行命令数 … 3
    - 各回線毎に最大 1 スレッドが競合を起こし得る

となっている。これからわかるように、出力キュー競合が起きる最悪の場合では、全 8 つのスレッドが同じ出力キューにセルを挿入しようとし、さらにその同じキューの空きセルの後始末をしようとして出力キュー操作をする場合である。単純計算ではこの場合、

$$19 \times (8 - 1) + 29 \times (1 - 0) = 162$$

または

$$19 \times (8 - 0) + 29 \times (1 - 1) = 152$$

つまり、162 または 152 ステップがロックを試みたスレッドが最後にロックを獲得できるまでにかかる最悪のオーバーヘッドのステップ数となる。

[1] 各出力キューの出力セル数を示すカウンタで、出力スケジューリングの可否の目安になる

同様に、出力セルのカウンタ競合が起きる最悪の場合も、全 8 スレッドが同一出力セルのカウンタの増加を試み、さらにその同一出力セルのカウンタの減少をしようとする場合である。この場合も、

$$3 \times (8 - 1) + 3 \times (1 - 0) = 24$$

または

$$3 \times (8 - 0) + 3 \times (1 - 1) = 24$$

つまり、24 ステップが最後にロックを試みたスレッドがロックを獲得できるまでにかかる最悪のオーバヘッドとなるステップ数である。

この最悪な場合が起きたとき、マルチスレッド型プロセッサでは単純に考えて

$$207 \times 8 + 162 + 24 = 1842$$

1842 ステップもかかってしまう。これをシングルスレッド型プロセッサと比較する。シングルスレッド型プロセッサではロックをして同期をとる必要がないので、通常通りに処理する。そのときのステップ数は

$$271 \times 8 = 2168$$

2368 ステップかかってしまう。したがって、最悪の場合でさえもマルチスレッド型プロセッサの方がまだスループットが高く、性能がいいと言えることがわかった。

もっとも、処理 1〜3 全部を行なう処理の場合、単純な確率で考えると、出力キューが全部で 40 本であるからこの最悪の場合というのは

$$\left(\frac{1}{40}\right)^8 \times 40 \doteq 6.1 \times 10^{-12}$$

という低い確率で生じるので、この最悪の場合についてはあまり考えなくてもよいとみなすことができる。

4.7 本章のまとめ

本章では、提案されたマルチスレッド型ATM交換機アーキテクチャの有効性を検証した。

検証方法は、各検証事項について、MIPS命令を実行するプロセッサの動作を模倣したシミュレータ上にセルスイッチングプログラムを動かして実行命令のステップ数を算出し、考察をする。このセルスイッチングプログラムはC言語で書き、それをmips-gccでコンパイルしたものをシミュレータ上で実行させる。

最初に、マルチスレッド型プロセッサによるソフトウェア処理によって、本仕様の交換機が実現可能であるかを検証した。1個のセルをスイッチングさせるのに、もっとも時間がかかる場合で183[nsec]であった。本仕様の交換機を満たすためには、1セル当たりのスイッチング時間を8倍した時間が2730[nsec]以下であればよい。よって、$2730[\mathrm{nsec}] \geq 1464[\mathrm{nsec}]$という関係が成り立つことから本仕様の交換機が実現可能、つまり、マルチスレッド型プロセッサによるソフトウェア処理が有効であることがわかった。

次に、マルチスレッド型プロセッサを用いたソフトウェア処理をシングルスレッド型プロセッサを用いた場合とを比較する。すると、マルチスレッド型プロセッサでは183[nsec]で済んだ処理がシングルスレッド型プロセッサでは247[nsec]もかかった。よって、マルチスレッド型プロセッサはシングルスレッド型プロセッサに対して約3割の性能向上が期待できるということがわかった。

これまでは単純にmips1アーキテクチャで考えていたので、パイプラインステージ数が5段であった。今度はスレッド数とパイプラインステージ数が同数である場合、つまり、パイプラインステージ数が8段のときのソフトウェア処理にかかる実行命令数について検証した。マルチスレッド型プロセッサが5段の場合と8段の場合では正味実行命令数にさほど変化が見られなかった。これは、同じ処理をマルチスレッド型プロセッサ上でするのであるから、当然の結果とも言える。しかし、8段の場合でシングルスレッド型プロセッサとマルチスレッド型プロセッサとを比較すると、最大で83%もの性能向上が見られるなど、ステージ数が5段の場合よりも8段の場合の方がシングルスレッド型プロセッサに対する性能向上比が高まることがわかった。

最後に、マルチスレッド型プロセッサに同期命令を加えた場合の実行命令数を算出して、同期命令によるオーバーヘッドの算出とシングルスレッド型プロセッサに対する比較をした。この同期をとる機構としては、スピンロックを用いた。すると、ロックをスピンなしで獲得した場合には同期によるオーバーヘッドは高々3割で収まった。また、全スレッド

による同一の同期変数へのアクセスが集中する最悪の場合でさえも、まだシングルスレッド型プロセッサに比べて性能が良いこともわかった。

したがって、本シミュレーションを通して判明したことは、

- マルチスレッド型プロセッサはパイプラインステージ数が多いほど、シングルスレッド型プロセッサに対する性能向上比が高まる。
- マルチスレッド型プロセッサに同期機構を加えても、シングルスレッド型プロセッサよりも性能が悪くならない。すなわち、同期機構はそれほどのオーバーヘッドではない。

ことであった。

本シミュレーションの課題として挙げられることは、スイッチプログラムのアセンブラコード生成である。今回は mips-gcc コンパイラを使用してコード生成をしたが、1 つ 1 つ命令を追ってコードを調べてみると、無駄な命令 (無意味なストア命令や、アドレス計算など) が実行されていることがあった。また、同期に関する課題もある。一般に、ロックをしてからロックを解除するまでのクリティカルセクションに入っている時間が短いと、ロックを獲得できずにスピンしているスレッドにとっては都合が良い。しかし、今回はコンパイラにそのことを認識させずにコンパイルさせたので、クリティカルセクション内でアドレス計算を行なうなど、冗長な処理が多く、結果的にクリティカルセクション内の命令数が多くなってしまった。したがって、クリティカルセクション内の実行命令数を最小にすれば、競合による待ち時間が減って、性能向上比がもっと高められると考えられる。そして更に性能が向上すると、現仕様の 8 本の入出力回線が 16 本に増やしたり、回線速度を 155Mbps 以上にすることも可能となる。

また、マルチスレッド型プロセッサには命令間の依存関係による nop の挿入や命令の入れ換えが必要ないので、マルチスレッド型プロセッサ用のコンパイラにはそれらは要らない。コンパイラには、それが要らない分、無駄な命令の排除、クリティカルセクションの最小化に努めることが望まれる。

第 5 章

結論

本論文で提案した、マルチスレッド型 ATM 交換機アーキテクチャは

- ATM 交換の特徴
- マルチスレッド型プロセッサの特徴

を生かした、双方の利点の相乗効果を狙ったアーキテクチャであった。

このような交換機アーキテクチャでは、

- マルチスレッド型プロセッサによるスループット向上
- ソフトウェア処理による柔軟性の供給

が期待され、結果として ATM に QoS 保証が可能な基盤を与えられると見込まれた。

本論文は、セルスイッチングのようなスレッド間で依存性が無いプログラムでは、マルチスレッド型プロセッサによるソフトウェア処理の方がシングルスレッド型プロセッサによるソフトウェア処理に比較して十分に有効性を持ち得ることを示した。しかし、本研究で行なったシミュレーションはかなり大雑把なシミュレーションなので、次のような問題点がある。(これに関しては今後の課題となる。)

- 実際にスイッチプロセッサと I/O プロセッサが連動して次々と流れて来る ATM セルをスイッチさせていない
- I/O プロセッサの仕様を詳細に決める必要がある

- セルスイッチングプログラムの無駄を省く必要がある

また、マルチスレッド型 ATM 交換機アーキテクチャへのもう一つの期待事項である『ソフトウェア処理による柔軟性の供給』についてはまだ検証していない。よって、このことに関しても今後の課題となる。

また、マルチスレッド型プロセッサを用いたソフトウェア処理のシミュレーションを通して得られた以下の指針を我々の研究室で開発段階にあるマルチスレッド型プロセッサに対して与える。

- 同期命令 (LL や SC) をサポートする基盤の提供、またはそれにとって代わるソフトウェアライブラリの提供。
- nop 命令の挿入や命令の入れ換えが要らない代わりに、無駄な命令を省くように最適化を施すコンパイラの提供。

これらが提供されれば、ユーザはかなり楽にプログラミングを行なえ、かつ実行時間の短縮化やスループットの向上を図ることができ、マルチスレッド型プロセッサの利点をフルに活用できると思われる。

将来的には、我々の研究室で開発中のマルチスレッド型プロセッサ上にセルスイッチングプログラムを実装させてマルチスレッド型 ATM 交換機を実現したい。

謝辞

本研究を行なうにあたり、日頃温かく御指導いただいた日比野靖教授に深く感謝致します。また、有益な御助言や御討論をいただいた堀口進教授、横田治夫助教授、丹康雄博士、杉野栄二博士、宮崎純博士ならびに研究室の諸兄、諸姉に感謝致します。

また、仕事が忙しいにも関わらず、マルチスレッドの概念、プロセッサの構成、シミュレータ作成、プログラミング方法などの本研究全般における多くの知識を私に与え、一緒に研究について考えて下さった日本電気の青澤尚氏に感謝致します。

参考文献

[1] 石川宏 監修 三宅功 編著 『絵とき ATM ネットワークバイブル』, オーム社, 1995

[2] 田尻頼彦 著 『ATM&マルチメディア絵とき基本用語』, オーム社雑誌局, 1996

[3] 青木利晴 青山友紀 濃沼健夫 監修 『広帯域ISDN と ATM 技術』, 電子情報通信学会, 1997

[4] Jean Garcia-Hara & Andrzej Jajszczyk. “ ATM Shared-Memory Switching Architectures ”,IEEE Network, pp.18-27, July/August 1994

[5] 高畠、橋本、辻田、武田、正畑 『ソフト処理による ATM 交換処理の実機評価』, 信学技報 SSE95-112, pp.19-24,1995-11

[6] 96 年度版 情報・通信新語辞典 日経 BP 出版センター ,1996

[7] Gregory T. Byrd & Mark A. Holliday “ MULTITHREADED Processor ”,IEEE SPECTRUM, pp.38-46, August 1995

[8] PATTERSON & HENNESSY ”COMPUTER ORGANIZATION & DESIGN” ,Morgan Kaufmann Publishers Inc. ,1994

[9] MIPS R4000 Microprocessor User's Manual , MIPS Technologies, Inc. , 1994

[10] Gerry Kane 著 前川守 監訳 『mips RISC アーキテクチャ- R2000/R3000』, 共立出版株式会社 , 1994

[11] GNU CC manual , Free Software Foundation, Inc. , 1991, 1992, 1993