計算機アーキテクチャ 101−20
（1993. 8. 20）

高性能プラットフォーム要素マイクロプロセッサ・アーキテクチャ
— 方式検討 —

村上和彰　　山家 陽

九州大学 大学院総合理工学研究科 情報システム学専攻

〒816 春日市春日公園 6-1

E-mail: {murakami, yamaga}@is.kyushu-u.ac.jp

汎用超並列マシンの構築に適した高性能プラットフォーム要素プロセッサ・アーキテクチャを実現するに先だって，当該プロセッサ・アーキテクチャに求められる性質および機能的要件の検討を述べている．まずは，「プラットフォーム要素プロセッサ」ということで次の3つの性質が求められる．すなわち，i) P: ポータビリティ(*Portability*)，ii) M: モジュラリティ(*Modularity*)，および，iii) S: スケーラビリティ(*Scalability*)である．最後に，「高性能」ということで高速性が絶対条件である．

本稿では，まず「プロセッサ間でのデータの授受(通信)およびメモリをいかに構成するか」，すなわちプロセッサ間通信モデルの決定に当たって検討すべき課題を整理し，かつ定性的な検討を行なう．次に，細粒度並列マシンとして求められる諸要件を整理し，筆者らの目指す要素プロセッサの検討を行なう．

High-Performance Platform Node-Microprocessor Architecture
— Architectural Aspects —

Kazuaki Murakami　　Akira Yamaga

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University

6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan

E-mail: {murakami, yamaga}@is.kyushu-u.ac.jp

Platform processing-element (PE) architectures for *general-purpose massively-parallel multiprocessor systems* are discussed. Before starting the design of a particular PE architecture, this paper presents the backgrounds, the position, and the design philosophy on such processing elements. This paper examines properties and processor requirements necessary for platform processing elements.

This paper especially focuses on the next two among many discussed progrems. First, the methods of inter-PE communications and the construction of the memory. So,we discuss inter-PE communication models. Seconds, the properties to be necessary for fine-grain parallel machine. Here, we aim for " the tolerant machine for fine-grain probrems".

1 はじめに

並列マシンは，プロセッサ（P），メモリ（M），および，スイッチ（S：ネットワーク）の3大構成要素から成る．マシン構築に当たっては，上記3要素，特にプロセッサPとスイッチSとをどのように構成するかで次の3つのアプローチに大きく分かれる [12]．第1のアプローチは，プロセッサとスイッチとを分離し，プロセッサにはoff-the-shelfの汎用マイクロプロセッサを流用し，スイッチにはカスタムメイドのLSIを用いるものである．この典型は，表1に示したParagon XP/Sや米Cray Research社のT3D（プロセッサはDEC Alpha）である．第2のアプローチは，プロセッサおよびスイッチともにカスタムメイドのLSI（群）を用いるものである．米NCUBE社のマシンや米Thinking Machines社のCM-1，CM-2は，プロセッサとスイッチとを1チップに集積した専用LSIを開発して用いている．表1のCM-5は，マルチチップ構成となり整数演算用に汎用マイクロプロセッサSPARCを用いているが，基本的にこのアプローチを踏襲している．最近のマシンでは，米Kendall Square Research社のKSR1，富士通VPP500，等がこのアプローチに該当する．第3のアプローチでは，プロセッサとスイッチを1チップに集積したoff-the-shelfの要素マイクロプロセッサを用いる．現在，要素マイクロプロセッサと呼べるのは，英Inmos社のトランスピュータだけである．表1のParsytec GCが，このアプローチに相当する．

第1のアプローチは，汎用マイクロプロセッサの高速化の著しい伸びに乗って，マシン全体の高速化が図れるという長所がある．しかしながら，現在のoff-the-shelfの汎用マイクロプロセッサは，ワークステーションやパーソナル・コンピュータといった逐次マシン，あるいは，共有バス結合の小規模並列マシンでの使用を前提としている．よって，今後ますます大規模化するであろう超並列マシンの要素プロセッサに流用するには，次の2つの理由により限界がある．まず，並列処理固有の諸機能を元々内蔵していないため，要素プロセッサとして用いる際に未装備の機能（たとえば，プロセッサ間通信）を外部回路として設ける必要がある．このため，チップとしての集積度は上がっても，基板レベルの集積度はなかなか上がらない．2つめは，並列マシンの規模により，プロセッサ外部との通信レイテンシが大きく異なってくる．よって，逐次マシンあるいは小規模並列マシンでの使用を前提に設計したプロセッサをまったく環境の異なる大規模並列マシンに流用するのは，性能上無理がある．

第2のアプローチは，プロセッサとスイッチを専用LSIで1チップ化できる場合，実装面では第1のアプローチより有利である．しかしながら，汎用マイクロプロセッサの高速化の伸びに追従して専用LSIを開発していくには，相当の技術力および資金力が必要であり対価格性能比の面で不利である．よって，このアプローチを採り続けることのできるメーカはそう多くはない．

第3のアプローチは，元々並列マシンの要素プロセッサとして設計されたoff-the-shelfの要素マイクロプロセッサを用いることから，前記2つのアプローチにおける問題点は生じない．ただ，ベンダが現在Inmos社1社だけということから，やはり開発競争が激しい汎用マイクロプロセッサの性能向上の伸びに追従していくのは容易ではない．この種の要素マイクロプロセッサがマイクロプロセッサの本流となるためには，Inmos社以外のベンダが市場に参入して市場そのものを拡大する必要がある．

並列処理は逐次処理を包含したより普遍的な処理形態であると考えれば，並列マシン用の要素マイクロプロセッサが逐次マシン用のプロセッサとして流用されてもおかしくない．これはつまり，前記第1のアプローチで逐次処理用の汎用マイクロプロセッサを並列マシンの要素プロセッサに流用していたのと，ちょうど逆向きの流用の仕方である．このような逐次／並列の区別なく大小さまざまな規模の並列マシンおよび広範なアプリケーションに汎用的に適用可能な「プラットフォームと成り得る高性能な要素マイクロプロセッサ」を**高性能プラットフォーム要素マイクロプロセッサ**（*high-performance platform node-microprocessor*）と呼ぶことにする [10]．将来の高性能プラットフォーム要素マイクロプロセッサとなるのは，前記第3のアプローチで用いられてる現在の要素マイクロプロセッサの後継機かも知れない．あるいは，第1のアプローチで用いられてる汎用マイクロプロセッサが大規模並列処理向きに大幅に拡張されたものである可能性もある．いずれにせよ，高性能プラットフォーム要素マイクロプロセッサには，以下の性質が求められよう．

まずは，「要素マイクロプロセッサ」ということで，プロセッサ，メモリ（管理），および，スイッチをチップとして一体化するというモジュラリティ（M: *Modularity*）が必要不可欠である．次に，「プラットフォーム」という要件から，ポータビリティ（P: *Portability*，アプリケーションを限定しない）およびスケーラビリティ（S: *Scalability*，プロセッサ数を限定しない）といった性質が求められる．最後に，「高性能」ということで高速性が絶対条件となる．数ある超並列マシンの中で成功している数少ないマシンに共通していることは，十分高速なプロセッサに適切な容量のメモリをつけて，これを適切な通信バンド巾のネットワークで多数接続している点である．一部の超並列マシンではプロセッサ数を増やす見返りにプロセッサの単体性能を犠牲にしているものが見受けられるが，これは問題のサイズを一定（＝プロセッサ単体性能×プロセッサ数）と仮定したとき成立する話であり，問題サイズも大きくなっていく点を見逃している．やはり，問題サイズに追従してマシン全体の性能を向上させていくには，最も根源に近い要素から高速化しなければならない．

我々は，このような高性能プラットフォーム要素マイクロプロセッサのアーキテクチャを2年前から検討している [12]．これまでに，**共有メモリ型マルチプロセッサ**（*shared-memory multiprocessor*）においてメモリ・アクセスのバッファリングおよびパイプライニングの度合を決定するメモリ・コンシステンシ・モデル（*memory consistency model*）[7, 8, 9]，および，共有メモリ型マルチプロセッサに限らずメッセージ交換型マルチコンピュータ（*message-passing multicomputer*）においても効率の良い実現が課題となっているバリア同期（*barrier synchronization*）[14] に関して基礎的な検討を行なった．

今後は，これらの検討結果を踏まえて，より具体的な方式検討およびアーキテクチャ設計を進めて行く予定である．本論文ではそれに先立ってまず，プロセッサ間通信モデル（2章）および粒度（3章）の2点について定性的な検討を行なう．

表 1: 並列マシン構築へのアプローチ

アプローチ		第1のアプローチ	第2のアプローチ	第3のアプローチ
マシン		CM-5	Paragon XP/S	Parsytec GC
開発元		米 Thinking Machines 社	米 Intel 社	独 Parsytec Gmbh
発表年		1991 年	1991 年	1991 年
要素プロセッサ	プロセッサ (性能)	SPARC(22MIPS)+VU (32MIPS,32MFLOPS) × 4	i860XP(42MIPS, 75MFLOPS) × 4	T9000(200MIPS, 25MFLOPS) × 1
	メモリ容量	32MB	16～128MB	?
	メモリ・バンド巾	512MB/秒	400MB/秒	200MB/秒
	通信バンド巾	20MB/秒	200MB/秒	80MB/秒
要素プロセッサ数		～16384	～1024	～16384
全体性能 (ピーク)		2TFLOPS	300GFLOPS	400GFLOPS

2 プロセッサ間通信モデル

2.1 検討項目

要素マイクロプロセッサ，ひいては，並列マシン全体のアーキテクチャを決定する最も大きな要因は，**プロセッサ間通信モデル**として何を採用するかである．すなわち，プロセッサ間でデータ授受（通信）[1]をどのように行うか，および，メモリをどのように構成するかである．プロセッサ間通信モデルの決定に当たっては，以下に挙げる種々の点を考慮に入れる必要がある．

1.論理的にメモリを共有するか否か?:
この選択如何で，次の2つのモデルが存在する．

- **共有メモリ**（*shared memory*）**モデル**：論理的にメモリを共有する．すなわち，「複数のプロセッサが通常の Read/Write 操作によりアドレス指定で直接的に読み書き可能な共有メモリ」が存在し，その共有メモリ上の共有変数に対する読み書きという形でデータ授受を行なう．
- **メッセージ交換**（*message passing*）**モデル**：論理的にせよメモリは共有しない．すなわち，共有メモリを介することなく，メッセージの交換という形でデータ授受を行なう．共有メモリ・モデルとの相違は，メモリ共有を否定している点であり，データ共有までも否定しているわけではない．後述の UMA や NUMA との対比で，**NORA**（*NO Remote Access*）モデルとも呼ぶ．

2.1.1 共有メモリ・モデル

まず，共有メモリ・モデルの場合，さらに以下の項目を検討する必要がある [6].

2.物理的に共有メモリを集中配置するか否か?:
この選択如何で，次の2つのモデルが存在する．

- **集中共有メモリ**（*centralized shared memory*）**モデル**：共有メモリを物理的に一ヶ所にグローバル・メモリとして集中配置する．すべてのプロセッサから同一時間で共有メモリにアクセス可能であることから，**UMA**（*Uniform Memory Access*）モデルとも呼ぶ．
- **分散共有メモリ**（*distributed shared memory*）**モデル**：共有メモリを物理的に一ヶ所に集中配置するのではなく，ある単位（たとえば，プロセッサ）毎にローカル・メモリとして分散配置する．各プロセッサから見た共有メモリの距離はアドレスにより遠近が生じアクセス時間も異なってくることから，**NUMA**（*NonUniform Memory Access*）モデルとも呼ぶ．

3.共有メモリ上のデータ（共有データと呼ぶ）をプロセッサにプライベートなキャッシュ・メモリへキャッシングするか否か?

共有データをキャッシングする場合，さらに以下の項目を検討する必要がある．

4.共有データのキャッシュ・コピー間の一貫性をプログラム不可視な手段により保証するか否か?:

NUMA モデルで共有データのキャッシュ・コピー間の一貫性を保証する場合，さらに以下の項目を検討する必要がある．

5.共有データのキャッシュ・コピーのホーム・メモリという概念が存在するか否か?[24]:
この選択如何で，次の2つのモデルが存在する．

- **CC-NUMA**（*Cache-Coherent* NUMA）**モデル**：個々の共有データについて，1個の「本籍地」（ホーム・メモリ）とそのキャッシュ・コピー（0個以上）各々に対応した0個以上の「現住所」とが存在する．本籍地はローカル・メモリで現住所はキャッシュ・メモリという2レベルのメモリ階層を構成する．
- **COMA**（*Cache Only Memory Architecture*）**モデル**：個々の共有データについて，「本籍地」が存在せず，そのキャッシュ・コピー（1個以上）各々に対応した1個以上の「現住所」のみが存在する．すなわち，ローカル・メモリとキャッシュ・メモリとが一体化した1レベルのメモリ階層を構成する．

[1]本稿を通して，通信と言った場合は，メッセージ通信のみならず，リモート・メモリ・アクセスや同期も含んでいる．

2.1.2 メッセージ交換モデル

一方，メッセージ交換モデルの場合，さらに以下の項目を検討する必要がある．

6. メッセージの送信者と受信者が同期するか否か?:
この選択如何で，次の2つのモデルが存在する．
   - **同期メッセージ交換モデル**：メッセージ送信者が送信可能状態に，また，メッセージ受信者が受信可能状態に，それぞれ同時に入っている期間中にのみメッセージの送受が行なえる．
   - **非同期メッセージ交換モデル**：メッセージ送信者は，メッセージ受信者の状態に関係なくメッセージを送信できる．同様に，メッセージ受信者は，メッセージ送信者の状態に関係なくメッセージを受信できる．

非同期メッセージ交換モデルの場合，さらに以下の項目を検討する必要がある．

7. メッセージ送受信者がメッセージ送受信の際にブロック（封鎖）されるか否か?:
この選択如何で，送受信それぞれに次の2手段が存在する．

(a)送信：
   - **封鎖型送信**（*blocking send*）：メッセージ送信者は，メッセージを送信したら当該メッセージがメッセージ受信者に受信されるまで，次の処理に進まずに待つ．
   - **非封鎖型送信**（*nonblocking send*）：メッセージ送信者は，メッセージを送信したら直ちに次の処理に進める．つまり，メッセージ受信者が当該メッセージを受信するまで待つ必要はない．

(b)受信：
   - **封鎖型受信**（*blocking receive*）：メッセージ受信要求を出したら当該メッセージが届くまで，メッセージ受信者は次の処理に進まずに待つ．
   - **非封鎖型受信**（*nonblocking receive*）：メッセージ受信要求を出した時点で当該メッセージが届いていない場合，メッセージ受信者は当該メッセージの到着を待つことなく次の処理に進む．

送信および受信に関する上記の選択肢はそれぞれ直交関係にあるので，以下の4通りの組合せが可能である．

- 封鎖型送信–封鎖型受信
- 非封鎖型送信–封鎖型受信
- 封鎖型送信–非封鎖型受信
- 非封鎖型送信–非封鎖型受信

2.2 検討

プロセッサ間通信モデルとして上記の共有メモリ・モデルおよびメッセージ交換モデルのいずれを採用するかという選択と，プロセス間通信モデルあるいはプログラミング・モデルとして共有メモリ・インタフェースおよびメッセージ交換インタフェースのいずれを提供するかという選択は相互に直交関係にあり，以下の4通りの組合せが可能である．

- **共有メモリ・インタフェース on 共有メモリ・モデル**：素直な組合せで，適用例多数．
- **メッセージ交換インタフェース on 共有メモリ・モデル**：例えば，共有メモリ・マルチプロセッサ上に実装された各種メッセージ交換ライブラリ（PVM (Parallel Virtual Machine), Express, P4, 等）[5].
- **共有メモリ・インタフェース on メッセージ交換モデル**：例えば，Kai Li らの共有仮想メモリ（*shared virtual memory*）[22].
- **メッセージ交換インタフェース on メッセージ交換モデル**：素直な組合せで，適用例多数．

したがって，プログラミング・モデルとはある程度独立してプロセッサ間通信モデルを選択することが可能である．そこで，ここではプログラマビリティは無視して性能の観点からのみ，共有メモリ・モデル（NUMA モデル）とメッセージ交換モデルを比較検討する．各モデルにおける通信の一般的なプロトコルは次のようになる．

- 共有メモリ（NUMA）モデル：
  1. プロセッサがリモート・メモリ・アクセス命令を発行したら，対応するハードウェア・モジュールは当該アクセス要求を記述したハードウェア・メッセージ[2]を生成して，相互結合網経由で対応するリモート・ノードへ送出する．このとき，必要なら何らかのアドレス変換操作を行なう．
  2. 上記ハードウェア・メッセージを受信したリモート・ノードの対応するハードウェア・モジュールは，当該ハードウェア・メッセージを解読して指定されたメモリ・アクセスを遂行する．そして，アクセス結果（ロードの場合はロード・データ，ストアの場合は ack，等）を格納したハードウェア・メッセージを生成して，相互結合網経由で要求元ノードへ返信する．このとき，必要なら何らかのアドレス変換操作を行なう．また，共有データのキャッシュ・コピー間の一貫性を保証する CC-NUMA モデルの場合は，さらに必要な措置をとる．
  3. 上記ハードウェア・メッセージを受信した要求元ノードの対応するハードウェア・モジュールは，当該ハードウェア・メッセージを解読して当該リモート・メモリ・アクセスを完了させる．完了条件は，採用したメモリ・コンシステンシ・モデルにより定まる [8].
- メッセージ交換モデル：

[2] メッセージ交換モデルで用いる「メッセージ」と区別するため，ソフトウェアからは不可視なメッセージを「ハードウェア・メッセージ」と呼ぶことにする．

1.送信ユーザ・プロセスはメッセージを作成したら，対応するハードウェア・モジュールに対して直接あるいはシステム・プロセスを介して間接にメッセージ送信を依頼する．システム・プロセスが介在する場合，必要に応じて（例えば，送信メッセージをバッファリングする場合）当該メッセージをユーザ領域からシステム領域にコピーする．

2.メッセージ本体の転送に先立って受信領域の確保が必要な場合，送信ノードの対応するハードウェア・モジュールはその旨を要求するハードウェア・メッセージを生成して，相互結合網経由で受信ノードへ送出する．当該ハードウェア・メッセージを受信した受信ノードの対応するハードウェア・モジュールは，当該ハードウェア・メッセージを解読して必要な措置（受信領域の確保，あるいは，プロセッサへの割込み発生）をとる．受信領域を確保したら，受信ノードの対応するハードウェア・モジュールはその旨を知らせるハードウェア・メッセージを生成して，相互結合網経由で送信ノードへ応答する．

3.送信ノードの対応するハードウェア・モジュールはメッセージ本体を送信する．当該メッセージがバッファリングされていた場合は，当該送信バッファを解放する．受信ノードの対応するハードウェア・モジュールは，当該メッセージ本体を受信して定められた受信領域内に格納する．必要ならプロセッサに対して割込みを発生する．

4.受信ユーザ・プロセスが上記メッセージを自領域内に受け取る．このとき，必要に応じて（例えば，受信領域がシステム領域に存在する場合，さらには受信メッセージをバッファリングする場合）当該メッセージをシステム領域からユーザ領域にコピーする．当該メッセージがバッファリングされていた場合は，当該受信バッファを解放する．

同期メッセージ交換モデルの場合は一般に送受信ともにメッセージのバッファリングは不要だが，メッセージ本体の転送に先立って受信領域をユーザ領域内に確保する必要がある．非同期メッセージ交換モデルで非封鎖型送受信を採用する場合，送信側および受信側でメッセージのバッファリングを行なう．なお，メッセージ交換プロトコルには，その高速化のために種々のバリエーションが提案されている（例えば，MDP[16]，MPC[19]，Line-sending & Buffer-receiving[3]，Active Message[17]，等）

メッセージ交換モデルにおける1メッセージ送受信当りの通信オーバヘッド[3]は，共有メモリ・モデルにおける1リモート・メモリ・アクセス当りの通信オーバヘッドよりも一般に大きくなる．よって，

- サイズの小さいデータの授受を頻繁に行なうような場合は共有メモリ・モデルの方が，一方，
- サイズの大きいデータの授受をたまにしか行なわないような場合はメッセージ交換モデルの方が，

それぞれ適していると言える．また，相互結合網とプロセッサ間通信モデルとの親和性という点では，

- ネットワーク・レイテンシ重視型設計（ネットワーク・スループットはそれほど高くないが，ネットワーク・レイテンシは極めて小さい）の相互結合網には共有メモリ・モデルの方が，逆に，
- ネットワーク・スループット重視型設計（ネットワーク・レイテンシはそれほど小さくないが，ネットワーク・スループットは極めて高い）の相互結合網にはメッセージ交換モデルの方が，

それぞれ適していると言えよう．このように一意に一方のプロセッサ間通信モデルが優れていると断定できないことから，また，ハードウェアの機能および構成としてどちらのモデルを採用しても大きな相違がないことから，プロセッサ間通信モデルとして共有メモリ・モデルとメッセージ交換モデルの双方を同一ハードウェアにより提供するマシンも既にある（例えば，KRPP[13]，Alewife[21]，等）．プラットフォーム要素マイクロプロセッサもその性質上，プロセッサ間通信モデルとして共有メモリ・モデルとメッセージ交換モデルの双方を備える必要があろう．

3 粒度

3.1 粒度の諸定義

粒度（*granularity*）とは「粒の大きさ（*grain size*）」の意で，ある問題を並列に解く際の実行単位（*run-time quantum*）の大きさの程度を表す．一方，並列に実行可能な実行単位の数の多さの程度を**並列度**（*parallelism*）という．要素マイクロプロセッサ・アーキテクチャの設計に当っては，どの程度の粒度を想定するのかを明確にする必要がある．まず，「粒度」という用語自身にもさまざまな定義が存在するので，以下でその主なものを整理する．

3.1.1 並列度の対比としての粒度

いま解くべき問題が与えられていて，それを解くのに十分な数のプロセッサが存在するものとする．この問題が並列実行可能かつ均一な任意の機能レベル（上はタスクから下は機械命令まで）の実行単位に一様に分割可能ならば，粒度と並列度との間には以下の関係が成り立つ．

$$\text{粒度} \times \text{並列度} = \text{問題の大きさ}$$

$$\text{粒度} = \frac{\text{問題の大きさ}}{\text{並列度}}$$

すなわち，問題の大きさが一定だとすると，粒度と並列度とは反比例の関係にあることになる（図??参照）．

3.1.2 実行時間と通信時間の比としての粒度

H. S. Stone[25] は，次のように粒度を定義している．

$$\text{粒度} = \frac{R}{C}$$

ここで，R（Run length）は実行単位の計算に要する時間，C（Communication）は当該実行単位の通信に要する時間である．このR/Cを尺度とすることで，問題に内在する並列性を次のように相対的に分類できる．

[3]通信レイテンシからネットワーク・レイテンシを差し引いたものをここでは通信オーバヘッドと呼ぶ．

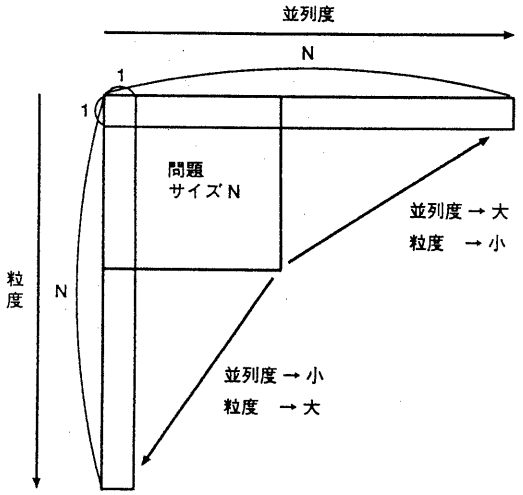

図 1: 並列度と粒度

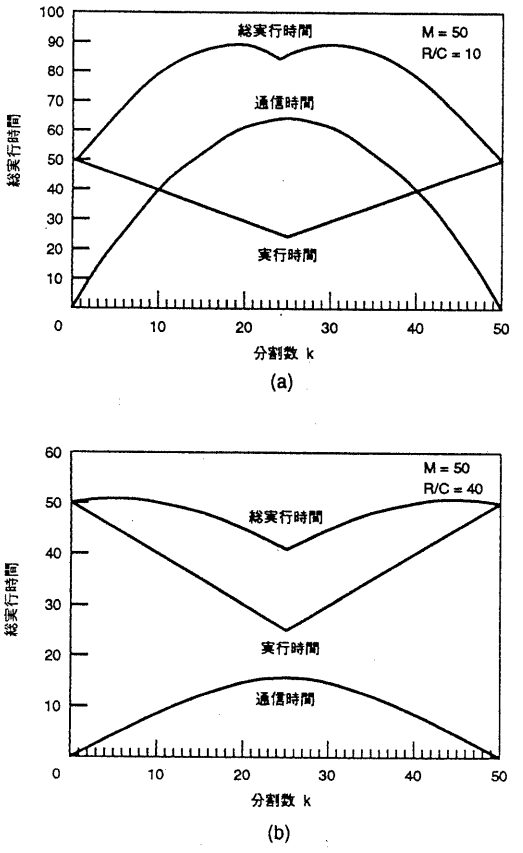

図 2: R/Cと最適タスク割付け

- **粗粒度並列性**（*coarse-grain parallelism*）：R/Cが比較的大きい．
- **細粒度並列性**（*fine-grain parallelism*）：R/Cが比較的小さい．

問題を実際に解くには，対応するプログラムを並列マシン上で実行することになる．このとき，問題に内在する並列性をマシンにどのように写像するか，具体的には，分割した個々の部分問題である実行単位をマシンにどのように写像するかが重要となる．この様子を次の例で見てみよう [25]．

いま，N台の均一なプロセッサから成る並列マシン上で，並列実行可能な M 個の実行単位（ここでは，タスクと呼ぶことにする）から成るプログラムを実行する．プロセッサ i には，k_i個のタスクが割り付けられる．各タスクの計算時間はすべて等しく，これを Rとする．また，各タスクは，異なるプロセッサに割り付けられたすべてのタスクと，通信時間 Cを費やして通信を行う．なお，タスク実行とタスク間通信，ならびに，タスク間通信どうしもオーバラップしないものとする（あるいは，オーバラップしたとしても完全には通信を隠蔽できずに，通信時間 Cが見えてしまうものとする）．このとき，プログラムの実行時間 $T(N)$ は下式で与えられる．

$$T(N) = R \times \max_{i=1}^{N}\{k_i\} + \frac{C}{2} \times \sum_{i=1}^{N}(k_i \times (M - k_i)) \quad (1)$$

この例では，各プロセッサに割り当てられたタスクの集まりが，マシンから見た処理単位，すなわち粒度ということになる．ここで課題となるのは，処理単位としての最適な粒度 k_i，すなわち，式1の $T(N)$ を最小とするような最適なタスク割当を見つけることである．この問題には，「$k_i \neq 0$ であるような k_iの値をすべて等しくする」という均等割付けの条件下で，次の最適解が存在する．

- $R/C \geq M/2$ の場合，$k_i = M/N$（簡単化のため Mは Nの倍数とする）．すなわち，タスクを全プロセッサに一様に均等配分する．
- $R/C < M/2$ の場合，ある 1 台のプロセッサ jに全タスクを割り付ける．つまり，$k_j = M, k_{i \neq j} = 0$.

プロセッサ数 $N = 2$ の場合における上記の様子を図??に示す [25]．

これらの最適タスク割当は，(直感に反するかも知れないが) 次のように言い換えることができる．

- 問題の粒度が $R/C \geq M/2$ のような粗粒度の場合，タスクを全プロセッサに一様に均等配分して処理するような**最大並列度並列処理**が適する．
- 問題の粒度が $R/C < M/2$ のような細粒度の場合，全タスクを 1 台のプロセッサで処理するような**最小並列度並列処理**が適する．

最適タスク割当問題に関しては，タスク・スケジューリング問題の一貫として種々のアルゴリズムが提案されている [1, 20, 23]．

3.1.3 継続実行時間としての粒度

式1において，実行単位が実行開始時点と終了時点でしか通信を行わないとする．また，その通信時間 C を単位時間1とする．すると，「一定量の通信を実行開始時点ないし終了時点で行う以外には，途中で通信することなしに継続して実行することが可能な時間間隔」としての粒度 CRL （Continuous Run Length）が定義できる．この CRL は，以下のように，問題の有する参照の局所性や局所計算可能性を反映する尺度であり，CRL を最適化することが重要な課題となる [2, 4]．

- 参照の時間的局所性が大きいと，一旦読み込んだデータをある一定時間間隔の間に何回か参照することになり，新たな通信を必要とする度合が小さくなる．よって，結果的に CRL が大きくなる．

- 参照の空間的局所性が大きいと，1回の通信で読み込んだデータ内に必要とするデータが多く含まれていることになり，通信の必要頻度が小さくなる．よって，結果的に CRL が大きくなる．

- 局所計算可能性が大きいと，通信の必要な相手が少なくなり，通信の必要頻度そのものが小さくなる．よって，結果的に CRL が大きくなる．

3.2 検討

以上見てきたように，R/C および CRL のいずれの尺度にしろ「細粒度」な問題に対する取り扱いが並列マシンの大きな課題となっており，事実「細粒度並列マシン」というキーワードが並列マシンの1つの挑戦目標として今日重要となってきている．ただし，粒度の定義が種々存在するように，細粒度並列マシンにもいろいろな定義が存在する．その主なものを以下に示す．

1. 細粒度な問題にのみ適したマシン：各プロセッサが保持できるコンテキスト容量（レジスタ・ファイル容量，キャッシュ容量，メモリ容量，等）が小さくて，実行単位の CRL が大きい粗粒度な問題の実行には適さない．言い換えると，CRL が小さい細粒度な問題の実行にしか適さないマシンである．例えば，シストリック・アレイ，コネクション・マシン（CM-1，CM-2），等．

2. 細粒度な問題に耐えられるマシン：実行単位の R/C や CRL が大きい粗粒度な問題の実行にもともと適しているが，それと同時に R/C や CRL が小さい細粒度な問題の実行にも耐えられる．具体的には，R/C や CRL が小さいと通信頻度および通信量が大きくなるので，これらに効率良く対処するための工夫を採っているマシンである．例えば，マルチスレッド処理マシン，等．

3. 実行単位を細粒度にしてしまうマシン：解くべき問題が本来有している CRL や R/C の大きさに関わらず，実行単位の CRL や R/C を強制的に小さくする．極端な例としては，実行単位が1機械命令で1命令毎に通信を必要とする純粋なデータフロー・マシンや SIMD マシン．

我々が目指すのは2番目の意味での細粒度並列マシン，すなわち，「細粒度な問題に耐えられるマシン」である．

一般に，並列マシンの物理的規模が大きくなると，プロセッサ外部との通信に伴うレイテンシの絶対値が大きくなる．また，通信レイテンシの絶対値が一定でも，プロセッサの動作周波数が向上すると，相対的に通信レイテンシが増加してしまう．このように増大の一途をたどる通信レイテンシに如何に対処するか，すなわち，**レイテンシ・トレランス技術**（*latency tolerance*）が，並列マシンの構築においては重要となる [18]．レイテンシ・トレランス技術は，その目的により**レイテンシ低減**（*latency reduction*：例えば，キャッシング，プリフェッチング，等）および**レイテンシ隠蔽**（*latency hiding*：例えば，バッファリング，パイプライニング，マルチスレッド処理，等）の2つのアプローチに分類される．

マルチスレッド処理（*multithreading*）は，レイテンシ隠蔽のための1技術である．現在，以下の2つのアプローチに合致するアーキテクチャとして，マルチスレッド処理プロセッサ（*multithreaded processor*）に関する研究が盛んに行われている．

- 従来の MIMD マシンにおいて，計算のスループット向上のためにコンテキスト（プロセス）切替え間隔を細粒度化する．

- 従来のデータフロー・マシンにおいて，計算のレイテンシ短縮のためにコンテキスト切替え間隔を粗粒度化する．

また，**PRAM**（*Parallel Random Access Machine*）をより実用的にした計算モデルとして近年注目されている **BSP**（*Bulk Synchronous Parallel*）モデルにおいて，マルチスレッド処理がアーキテクチャの備えるべき機能の1つとして挙げられている [26]．マルチスレッド処理プロセッサの設計に当っては，文献 [10, 15] にあるように多くの要検討項目が存在する．レイテンシ・トレランス技術に関する検討はまた別に機会に行なう．

4 おわりに

以上，高性能プラットフォーム要素マイクロプロセッサのアーキテクチャ設計に先立って，プロセッサ間通信モデルおよび粒度の2点について定性的な検討を行なった．今後はより定量的な検討を行なう必要がある．また，レイテンシ・トレランス技術についても精力的な検討が必要である．

謝辞

日頃ご討論頂く，九州大学 大学院総合理工学研究科 安浦寛人 教授，同 工学部 弘中哲夫 助手，ならびに，安浦研究室の諸氏に感謝します．

本研究は一部，文部省科学研究費補助金 重点領域研究「超並列原理に基づく情報処理体系」による．

参考文献

[1] 笠原博徳，並列処理技術，第3章，pp.104–168，1991年．

[2]坂井, 児玉, 佐藤, 山口, "超並列計算機における粒度最適化機構の検討," 並列処理シンポジウム *JSPP'92*論文集, pp.235–240, 1992 年 6 月.

[3]清水, 堀江, 石畑, "高速メッセージハンドリング機構 — AP1000 における実現 —," 情処論, vol.34, no.4, pp.638–647, 1993 年 4 月.

[4]関口, 佐藤, "細粒度並列処理における性能評価モデル — 節度ある並列性を求めて —," 並列処理シンポジウム *JSPP'92*論文集, pp.249–254, 1992 年 6 月.

[5]関口, 長嶋, 日向寺, "ワークステーションクラスタとメッセージパッシングライブラリ," 情処研報, HPC–47–3, 1993 年 6 月.

[6]寺澤, 天野, 工藤, "マルチプロセッサの記憶システム," 情報処理, vol.34, no.1&2, pp.96–105&233–243, 1993 年 1&2 月.

[7]徳永, 村上, 山家, "高性能プラットフォーム要素マイクロプロセッサ・アーキテクチャ— 通信モデルに関する検討 —," 信学技報, CPSY92–20, 1992 年 8 月.

[8]徳永, 村上, "メモリ・コンシステンシ・モデル— フレームワーク化および新モデルの提案 —," 情処研報, ARC–97–5, 1992 年 12 月.

[9]徳永, 村上, "メモリ・コンシステンシ・モデル— 新しいモデルの提案, および, その能力比較 —," 並列処理シンポジウム *JSPP'93* 論文集, pp.253–260, 1993 年 5 月.

[10]村上和彰, "超並列マルチプロセッサ・システム向き要素プロセッサ・アーキテクチャ— 要件 —," 情処研報, 91–ARC–89–29, 1991 年 7 月.

[11]村上和彰, "マイクロプロセッサ・アーキテクチャと並列処理技術 — マイクロプロセッサ用並列処理と並列処理用マイクロプロセッサ —", 信学技報, ICD91–91, 1991 年 9 月.

[12]村上和彰, "21 世紀の高性能プラットフォーム要素マイクロプロセッサ — 命令レベル並列処理技術およびレイテンシ・トレランス技術 —," 電子材料, vol.31, no.6, pp.54–57, 1992 年 6 月.

[13]森, 蒲池, 濱口, 村上, 福田, 末吉, 富田, "可変構造型並列計算機の PE 間メッセージ通信機構," 情処論, vol.30, no.12, pp.1593–1602, 1989 年 12 月.

[14]山家, 村上, "バリア同期モデル— Taxonomy と新モデルの提案, および, モデル間性能比較 —," 並列処理シンポジウム *JSPP'93* 論文集, pp.119–126, 1993 年 5 月.

[15]Boothe, B. and Ranade, A., "Improving Multithreading Techniques for Hiding Communication Latency in Multiprocessors," *Proc. 19th Int'l. Symp. Computer Architecture*, pp.214–223, May 1992.

[16]Dally, W. J. et al., "Architecture of a Message-Driven Processor," *Proc. 14th Int'l Symp. Comupter Architecture*, pp.189–196, June 1987.

[17]von Eicken, T., Culler, D. E., Goldstein, S. C., and Schauser, K. E., "Active Messages: a Mechanism for Integrated Communication and Computation, " *Proc. 19th Int'l. Symp. Computer Architecture*, pp.256–266, May 1992.

[18]Gupta, A., Hennessy, J., Gharachorloo, K., Mowry, T., and Weber, W.-D., "Comparative Evaluation of Latency Reducing and Tolerating Techniques," *Proc. 18th Int'l. Symp. on Computer Architecture*, pp.254–263, May 1991.

[19]Hsu, J.-M. and Banerjee, P., "A Message Passing Coprocessor for Distributed Memory Multicomputers," *Proc. Supercomputing'90*, pp.720–729, Nov. 1990.

[20]Kruatrachue, B. and Lewis, T., "Grain Size Determination for Parallel Processing," *IEEE Software*, vol.5, no.1, pp.23–32, Jan. 1988.

[21]Kubiatowicz, J. and Agarwal, A., "Anatomy of a Message in the Alewife Multiprocessor," *Proc. 1993 Int'l. Conf. Supercomputing*, pp.195–206, July 1993.

[22]Li, K. and Hudak, P., "Memory Coherence in Shared Virtual Memory Systems," *ACM Trans. Computer Systems*, vol.7, no.4, pp.321–359, Nov. 1989.

[23]McCreary, C. and Gill, H., "Automatic Determination of Grain Size for Efficient Parallel Processing," *Comm. ACM*, vol.32, no.9, pp.1073–1078, Sept. 1989.

[24]Stenström, P., Joe, T., and Gupta, A., "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures," *Proc. 19th Int'l. Symp. Computer Architecture*, pp.80–91, May 1992.

[25]Stone, H. S., *High-Performance Computer Architecture*, Sec. 6.2, pp.283–299, Addison-Wesley, 1987.

[26]Valiant, L. G., "A Bridging Model for Parallel Computation," *Comm. ACM*, vol.33, no.8, pp.103–111, Aug. 1990.