計算機アーキテクチャ 89−29
（1991. 7. 19）

# 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 —

村上和彰

九州大学 大学院総合理工学研究科

**汎用超並列マルチプロセッサ・システム** (*general-purpose massively parallel multiprocessor system*) の構築に適したプラットフォーム要素プロセッサ (*platform processing element*) アーキテクチャを検討するに当たって，その検討姿勢および方針，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を述べている．

まず，プラットフォーム要素プロセッサに3つの性質 (PMS) を求めている．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．この点に関して，現在唯一のプラットフォーム要素プロセッサである Inmos 社の transputer は良い手本となっている．そこで，transputer を始めとする iWarp，MDP 等のトランスピュータ型要素プロセッサの仕様をまとめ，共通する機能を示している．

また，シングルプロセッサ・システム環境とは異なり，マルチプロセッサ・システム環境ではレイテンシ・トレランス (*latency tolerance*) が大きな技術的課題となる．そのための諸技術を，i) 低レイテンシ化 (*latency reduction*)，および，ii) レイテンシ隠蔽 (*latency hiding*) の2局面で示している．特に，レイテンシ隠蔽のためのマルチスレッド処理プロセッサに着目し，その要検討項目を整理している．

# Processing-Element Architecture for Massively Parallel Multiprocessor Systems — Processor Requirements —

Kazuaki MURAKAMI

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6–1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan
*E-mail: murakami@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 also examines properties and processor requirements necessary for platform processing elements.

Three properties, such as *portability*, *modularity*, and *scalability*, are claimed to be necessary to platform PE architectures. The only commercial platform processor satisfing the properties at present is the Inmos line of transputers. There are also transputer-like processors, such as iWarp and MDP. This paper surveys such generic transputers and indicates common functions for these processors.

In massively parallel multiprocessor systems, inter-PE communications suffer significant latencies. There are at least two ways to lowering the effective latencies: *latency reduction* and *latency hiding*. For techniques for latency hiding, several *multithreaded processors* are investigated.

## 1 はじめに

**汎用超並列マルチプロセッサ・システム** (*general-purpose massively parallel multiprocessor system*) の構築に適した**プラットフォーム要素プロセッサ** (*platform processing element*) アーキテクチャを今後検討していく．まず，本論文で，検討に当たっての姿勢および方針を示し，要素プロセッサ・アーキテクチャに求められる性質および機能的要件を整理する．

"××並列" と言った場合の "××" とプロセッサ台数との間の関係については一般的な定義が存在しないが，ここでは，

- **小並列** ⇔ *O(1)* 台
- **中並列** ⇔ *O(10)* 台
- **高** (*highly*) **並列** ⇔ *O(100)* 台
- **超** (*massively*) **並列** ⇔ *O(1000)-O(10000)* 台

とする [2]．また，"マルチプロセッサ" と言った場合，**MIMD**(*Multiple Instruction stream/Multiple Data stream*) と同義に用いる．すなわち，**共有メモリ型マルチプロセッサ** (*shared-memory multiprocessor*) に限らず，**メッセージ交換型マルチコンピュータ** (*message-passing multicomputer*)[14] をも意味する．

本論文では，まず次章で背景説明を行ない，3章で超並列処理に臨む姿勢を明確にする．4章では，検討の基本方針を示す．そして，5章で，要素プロセッサ・アーキテクチャの現状および現在までの研究を整理し，そこから要素プロセッサに求められる機能的要件を探る．

## 2 背景

マルチプロセッサ・システムは，次の 3 大構成要素から成る．

- P: プロセッサ *(Processor)*
- M: メモリ *(Memory)*
- S: スイッチ *(Switch)* [≡ネットワーク]

マルチプロセッサ構築を可能とした大きな技術的背景の一つが VLSI/マイクロエレクトロニクス技術であることから判る通り，今日までのマルチプロセッサ・システムの大部分がその要素プロセッサ (P) として商用の汎用マイクロプロセッサを採用している．いわゆる，マルチ・マイクロプロセッサ (*multi-microprocessor*) と呼ばれるシステムである．これらのシステム開発においては，マイクロプロセッサの接続，すなわちメモリ (M) およびスイッチ (S) の開発に主力を注ぎ，要素プロセッサの方は VLSI 技術の進歩がもたらす恩恵に任せる，といった開発方針を採るのが一般的であった．

これまでの小〜中並列マルチプロセッサ・システムでは，それはそれで問題がなかったかも知れない．しかしながら，商用の汎用マイクロプロセッサと言った場合，ワークステーション，パーソナル・コンピュータといったシングルプロセッサ・システム，あるいは，ごく小規模な小並列マルチプロセッサ・システムでの使用を前提としたものが大多数である (唯一の例外が Inmos 社の transputer[27] である)．よって，今後ますます大規模化するであろう超並列マルチプロセッサ・システムの要素プロセッサとしては，現在の商用・汎用マイクロプロセッサ (および，その延長線上から逸脱しない後継マイクロプロセッサ) は，以下の理由により適さないと考える．

- **機能の欠如**: シングルプロセッサ・システムでの使用を前提とするマイクロプロセッサには，
  - ネットワークを介したグローバルなメモリ・アクセス，
  - 上記メモリ・アクセスに伴って必要となるデータ・コヒーレンス保証
  - 上記メモリ・アクセス，上記データ・コヒーレンス保証，同期，メッセージ伝達，等を実現するプロセッサ間通信，

  といった並列処理固有の諸機能が元々備わっていない (小並列マルチプロセッサ・システムでの使用をも前提として，上記の機能の一部が徐々に実装されつつはある)．よって，要素プロセッサとして用いるには，未装備の機能を外部回路としてマイクロプロセッサ周辺に外付けする必要がある．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といった問題が生じる．
- **環境の相違**: シングルプロセッサ・システムとマルチプロセッサ・システム，さらに，マルチプロセッサ・システムでも小〜中〜高〜超並列とでは，プロセッサを取り巻く環境が異なる．たとえば，プロセッサがメモリ・アクセスを行なった際のレイテンシは，シングルプロセッサ・システムと超並列マルチプロセッサ・システムとでは 1 桁から 2 桁ほど差がある．よって，シングルプロセッサ・システムでの環境を想定して設計されたプロセッサをまったく異なった環境で使用するのは無理がある．
- **トレードオフ点の変化**: さらに，命令セット・アーキテクチャ自身も単一&逐次処理を前提として定められているので，並列/並行処理には適さない．これには，プロセッサ間通信命令等の並列処理固有の諸命令の不備といった問題に加えて，並列化コンパイラの能力を活かすような命令体系になっていないという問題もある．現在の商用・汎用マイクロプロセッサの主流である RISC アーキテクチャが 10 年前に (単一&逐次処理用) 最適化コンパイラとのトレードオフで誕生したように，超並列マルチプロセッサ用の要素プロセッサ・アーキテクチャは並列&並行化コンパイラとのトレードオフの洗礼を受けて新たに定める必要がある．

上記の事情により，将来の超並列マルチプロセッサ・システムを見据えた要素プロセッサ・アーキテクチャの検討をいま始めるべきであると考える．しかも，特定のアプリケーションあるいは超並列マルチプロセッサ・システムに特化するのではなく，広範なアプリケーションおよび種々の超並列マルチプロセッサ・システムに適用可能なプラットフォームと成り得る要素プロセッサ・アー

表 1: キーワード $(PMS)^3$

| | 構成要素 | コンパイラ技術 | 要素プロセッサに必要な性質 |
|---|---|---|---|
| P | *Processor* | *Partitioning* | *Portability* |
| M | *Memory* | *Mapping* | *Modularity* |
| S | *Switch* | *Scheduling* | *Scalability* |

キテクチャの策定が必要である．この意味において，現在唯一のプラットフォーム要素プロセッサである Inmos 社の transputer[24] は良い手本となる．

なお，プロセッサ・アーキテクチャと並列/並行処理との関係としては，

- **プロセッサ内命令レベル並列処理** [8][9]
  - ① 命令パイプライン処理方式
  - ② スーパースカラ方式
  - ③ VLIW 方式
  - ④ ベクトル処理方式
  - ⑤ ハイパースカラ方式 [10]
- **プロセッサ内並行処理** (*multithreading*)
- **プロセッサ間並列&並行処理**

等が存在するが，ここでは後二者のみを検討の対象とする．

表 1に，本論文を通じて登場する 3 組のキーワード $(PMS)^3$ を示す．

## 3 姿勢

まず，『超並列処理に臨む姿勢』を明確にする．

### 3.1 並列処理は高性能を得るための「必要悪」技術である

ここで言う "**並列処理** (*parallel processing*)" とは物理的かつ空間的にプロセッサを多重化した処理形態であり，アルゴリズム・レベルないしプログラミング・レベルにおいて論理的に処理体を多重化した "**並行処理** (*concurrent processing*)/**並行プログラミング** (*concurrent programming*)" とは異なる[1]．並列処理および並行処理にそれぞれ対峙するのが，"**単一処理** (*serial processing*)" および "**逐次処理** (*sequential processing*)" ということになる (後述するように，単一処理↔並列処理と逐次処理↔並行処理とは直交関係にある)．

さて，高性能を得るための「必要悪」技術の代表格としては，次の 3 大技術が挙げられる．

① 集積度を上げる．
② クロック周波数を上げる．
③ 並列度を上げる (プロセッサ台数を増やす)．

[1] 並行処理/プログラミングは「必要善」技術である．

なぜ，これらが「必要悪」技術なのか？理由は，以下の設問に対する解を考えれば自明であろう．

『もしシステムが所望性能を十分満足している場合，次はコストを下げたい，あるいは，信頼性を上げたい．このとき，何をなすべきか？』

おそらく，大部分のシステム設計者は以下のいずれか (または，すべて) の処置を採るに違いない．

① コスト削減および歩留り向上のために，ダイ・サイズを小さくする．つまり，集積度を下げる．
② 消費電力削減および耐雑音性向上のために，クロック周波数を下げる．
③ コスト削減，サイズ縮小，消費電力削減，および，MTBF(平均故障間隔時間) 向上[2]のために，プロセッサ台数を減らす．

ただ現在のところ，『システムが所望性能を十分満足している』といった状況がまず存在しないので，以上のことが広く一般に認識されるには至っていない．

### 3.2 プロセッサ台数はシステム設計の "結果" であって "目標" ではない

システム設計の第一義の "目標" は，性能 (*performance*) あるいは価格対性能比 (*cost/performance*) の向上に他ならない．専用システムと汎用システムとでは，次のようにシステム設計 " 目標" に関する事情が若干異なる．というか，この事情の相違こそが，システムの専用/汎用の切り分けそのものであると考える．

- **専用システム**: システム設計時に，アプリケーションおよび所望性能が明確に定まっている．このときの性能は，応答時間 (*response time*) で与えられることが多い．よって，専用システム設計の第一義の "目標" は，『与えられたアプリケーションを決められた性能で実行する』こととなる．つまり，システムが実用に耐え得るための絶対的な目標性能というものがあり，それを達成することがシステム設計者に課せられた使命である．そのためには，手段を選ばない．すなわち，性能向上のための 3 大「必要悪」技術を駆使することになる．したがって，"目標" としての性能が先にあり，それを達成しようとした "結果" としてプロセッサ台数が存在するのである．決して，プロセッサ台数が "目標" として先にあるわけ

[2] 耐故障性向上のためにプロセッサを多重化する場合は，この限りではない．

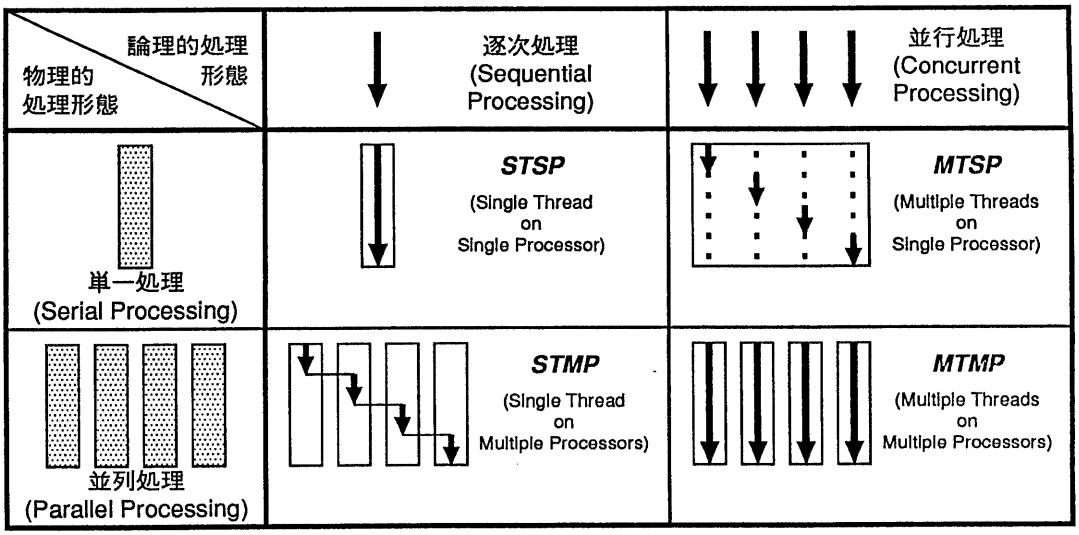


図 1: 並行&並列処理形態

ではない．なお，コストは言わば二の次であり，所望性能を達成した場合にのみコスト削減が話題となり得る．

- 汎用システム: 上記の定義による専用システム以外のシステムはすべて汎用 (非専用) システムである．性能はもはや絶対的な目標ではなく，コストに対する相対的な目標でしかない．すなわち，汎用システム設計の第一義の“目標”は価格対性能比にある．ここで言う性能は，応答時間の場合もあればスループット (*throughput*) の場合もある．いずれにせよ，汎用システムでは，集積度，クロック周波数，および，プロセッサ台数は，性能とコストとの間のトレードオフ“結果”により決まることになる．このようにプロセッサ台数は本来，性能-コスト間のトレードオフ点を表すものだが，性能あるいはコストの代名詞として独り歩きしている場合が多い．たとえば，
  - 汎用システムにおいてはアプリケーションを特定できないので，システムの性能評価指標としてプロセッサ台数あるいはピーク性能 (= プロセッサ単体性能×プロセッサ台数) を持ち出す，あるいは，
  - システムの評価評価指標として台数効果 (*efficiency*; = 速度向上比/プロセッサ台数) を用いる際，プロセッサ台数をあたかもコストの如く扱う，

  といった具合である．上記前者により，システム設計の第一義の“目標”が『プロセッサ台数を増やす』ことにすり替わっているような汎用システムも多い．あくまでも，汎用システム設計の第一義の“目標”は価格対性能比にあるのであって，プロセッサ台数にあるわけではない．

### 3.3 並行処理の計算モデルと並列処理の計算モデルとは独立である

結局，『同一性能のシステム同士なら，

① 集積度が低く，
② クロック周波数が低く，
③ プロセッサ台数が少ない

システムの方がコストおよび信頼性の面で優れたシステムである』ということになる．だが，この主張に対しては，次の反論が予想される．

『並列度の高いアプリケーションを実行するには，システムも同じだけの並列度 (プロセッサ台数) を有する必要がある．』

しかし，この論理は成立しない．いまさら述べるまでもなく，並行処理と並列処理とは別物である．『並列度の高いアプリケーション』を記述するには，アルゴリズム・レベルないしプログラミング・レベルにおいて，論理的にそれだけの並列度が表現できればよい．これが，並行処理/並行プログラミングである．一方，当該アプリケーションを実行するには，必ずしも物理的かつ空間的に同じだけの並列度 (プロセッサ台数) が必要なわけではない．極端な話，プロセッサ 1 台の単一処理でも実行は可能である．『量が質を変える』は並行処理に対して言えるのであって，並列処理に対しては当てはまらない．

単一処理↔並列処理と逐次処理↔並行処理との間には，図 1 に示すように，以下の 4 組合せが可能である．

- 単一&逐次処理: **STSP**(*Single Thread on Single Processor*)
- 単一&並行処理: **MTSP**(*Multiple Threads on Single Processor*)

表 2: システム構成要素 PMS–コンパイラ/ライブラリ技術 PMS 間の技術的関係

| | P: 分割 (*Partitioning*) | M: 写像 (*Mapping*) | S: スケジューリング (*Scheduling*) |
|---|---|---|---|
| P: プロセッサ (*Processor*) | タスク分割 | タスク割付け | タスク・スケジューリング |
| M: メモリ (*Memory*) | データ分割 | データ配置 | データ・ステージング |
| S: スイッチ (*Switch*) | 通信分割 | トポロジ写像 (ルーティング) | イベント・オーダリング |

- 並列&逐次処理: **STMP**(*Single Thread on Multiple Processors*)
- 並列&並行処理: **MTMP**(*Multiple Threads on Multiple Processors*)

『並列度の高いアプリケーションを実行するには，システムも同じだけの並列度 (プロセッサ台数) を有していれば十分である.』

上の論理は成立しよう．しかし,『システムも同じだけの並列度 (プロセッサ台数) を有している』のがはたして最適か否かはまた別の問題である．このことは，プロセッサ台数に限らず，プロセッサ間結合形態，プロセッサ–メモリ間結合形態，メモリ配置形態，プロセッサ機能，命令セット・アーキテクチャ等，様々な局面においても同様に言える．つまり，並行処理の計算モデルと並列処理の計算モデルとは独立したものであって，両者が必ずしも完全一致する必要はないと考える．両モデル間のセマンティック・ギャップを埋めるために，単に両者を一致/接近させればよいとする短絡的アプローチは，過去の高級言語計算機の過ちを繰り返しているように思える．むしろ重要なのは，以下のことである．

① 並行処理の種々の計算モデルから共通かつ普遍的な特徴/機能を抽出・抽象化する [20].
② 並列処理の計算モデルとしてハードウェア上何が可能かを見極め，上記特徴/機能の個々についてそれをアーキテクチャに組み込むか否かのトレードオフ決定を行う [20].
③ そして，並行処理と並列処理の両計算モデル間のセマンティック・ギャップを埋めるために，以下のコンパイラ/ライブラリ技術を開発する.

- P: 問題(タスク，データおよび通信)分割(*Partitioning*)
- M: 分割された個々のタスク，データおよび通信の空間内位置に関する写像(*Mapping*)
- S: 分割された個々のタスク，データおよび通信の時間内位置に関するスケジューリング(*Scheduling*)

なお，上記のコンパイラ/ライブラリ技術の PMS と，マルチプロセッサ・システムの 3 大構成要素である PMS との間には，表 2に示す技術的関係がある.

### 3.4 速度は機能および構造を凌駕する

前節で『過去の過ち』の例として引合に出した高級言語計算機には，次の 2 タイプが存在した.

- **高機能型**: "△△言語" マシンや CISC
- **柔構造型**: ユニバーサルホスト・マシン

現在の状況を見る限り，一部の専用分野を除いた汎用分野では，高級言語計算機に比べて低機能かつ硬構造ではあるが高速な RISC が市場を席巻している．機能の高低および構造の柔硬は相対的指標であり，速度のような絶対的数値では表せない．よって，機能および構造と速度との間の因果関係を立証することは出来ないが，現象としては上記の標題『速度は機能および構造を凌駕する』は正しいと見る.

汎用マルチプロセッサ・システムにおいても同様である．前節で述べたように，並行処理の計算モデルに並列処理の計算モデルを一致/接近させるがためのアーキテクチャの盲目的高機能化は，逆に低速化を招く危険がある．また，システム構成要素を柔構造化したシステム (たとえば，[29]) はテストベッドとしては価値があるかも知れないが，実用システムとして見た場合は，構造–速度間のトレードオフが構造の方に寄り過ぎていて速度を犠牲にしているきらいがある.

一般に，transputer や後述する J-Machine[21] は，以下のように並行処理の計算モデルをアーキテクチャに直接写像したように捉えられている.

- CSP/Occam ⇒ transputer
- actors/CST ⇒ J-Machine

しかし，これらのアーキテクチャを検証してみれば，確かに上記計算モデルを "指向" してはいるが，機能–速度間のトレードオフが十分なされており，決して『盲目的高機能化』によるものではないことが判る.

## 4 方針

前章で示した『超並列処理に臨む姿勢』を基に，要素プロセッサ・アーキテクチャの検討に際しての基本方針をまとめる.

- **方針1**: 汎用超並列マルチプロセッサ・システム [36] を対象として，プラットフォーム要素プロセッサと成り得る要素プロセッサ・アーキテクチャを策定する．プラットフォーム・アーキテクチャであるからには，汎用システムはもちろん，性能さえ満足すれば専用システムまで，多様な超並列マルチプロセッサ・システムに適用可能なポータブル (*portable*) な要素プロセッサでなければならない．
- **方針2**: 上記方針1により，システム設計の "目標" を価格対性能比に置く．プロセッサ台数は，性能とコストとの間のトレードオフ "結果" により決まるものであり，『プロセッサ台数を増やす』ことをシステム設計の "目標" とはしない．
- **方針3**: しかし，システム設計の "結果" プロセッサ台数が何台になってもよいように，つまり，プロセッサ台数に制限を与えないようなスケーラブル (*scalable*)[3] なシステム・アーキテクチャおよび要素プロセッサ・アーキテクチャとする．
- **方針4**: 上記方針3はさらに，要素プロセッサに対して，プロセッサ (P)-メモリ (M)-スイッチ (S) を一体化したモジュラ (*modular*) なアーキテクチャたることを要求している．
- **方針5**: 並列処理の計算モデルは，並行処理の計算モデルとは独立に定める．このとき，要素プロセッサがプラットフォームと成り得るためには，以下の作業が必要である．すなわち，並行処理のための複数の計算モデルにおいて頻繁に用いられ，かつ，速度を落さすことなく最低限のハードウェアで実現可能な特徴/機能を抽出・抽象化して実装する [20]．
- **方針6**: アーキテクチャの高機能化および柔構造化に走らず，あくまでシステム全体の性能および価格対性能比の向上に注力する．その結果として，アーキテクチャが低機能あるいは硬構造となっても構わない．
- **方針7**: 上記の方針5および方針6の結果，並行処理の計算モデルと並列処理の計算モデルとの間のセマンティック・ギャップは大きいものとなる可能性がある．このギャップを埋めるためには，表2に示した機能を有する高度な並行/並列化コンパイラが必要不可欠である．逆に，このような高度な並行/並列化コンパイラの存在を前提として，アーキテクチャおよびシステムの設計を行なう必要がある．このようなアプローチを "**CAMP**(*Compiler-Aided MultiProcessing*)" と呼ぶことにする．

以上の検討方針より，プラットフォーム要素プロセッサに求められる3つの性質がまず明らかになった．

- P: ポータビリティ (*Portability*)[方針1より]
- M: モジュラリティ (*Modularity*)[方針4より]
- S: スケーラビリティ (*Scalability*)[方針3より]

[3] Hill が試みた厳密な定義 [25] に基づくものではなく，Scott が与えた実用的な定義 [32] による．

# 5 要件

汎用超並列マルチプロセッサ・システム (いわゆるマルチマイクロプロセッサは除く) の要素プロセッサ・アーキテクチャの現状および現在までの研究を整理し，そこから要素プロセッサに求められる機能的要件を見てみよう．要素プロセッサ・アーキテクチャには，次の2つの大きな流れがある．

- トランスピュータ型要素プロセッサ
- マルチスレッド処理プロセッサ

## 5.1 トランスピュータ型要素プロセッサ

現在，商用の汎用マイクロプロセッサでプラットフォーム要素プロセッサと呼べるのは，唯一 Inmos 社の **transputer** ファミリ (T800, T425, 等[4])[27] だけである．transputer は，1チップ内にプロセッサ (P)，メモリ (M) およびスイッチ (S) を集積しており，要素プロセッサとしてのモジュラリティ (M) を満足している．また，商用/研究用を問わず，小並列から超並列まで実に多くの様々なマルチプロセッサ・システム (ほとんどは，メッセージ交換型マルチコンピュータ) の要素プロセッサとして採用されていることから [24]，そのポータビリティ (P) およびスケーラビリティ (S) とも問題ない．

商用ではないが，以下の研究用の3種類の要素プロセッサは，transputer に通ずるところがある．

- **iWarp**[15][16]: CMU の Kung 等および Intel 社
- **MDP(J-chip)**[19][21]: MIT の Dally 等および Intel 社
- **Mosaic**[14]: Caltech の Seitz 等

これらの要素プロセッサを "**トランスピュータ型要素プロセッサ**" と呼ぶことにする．

表3に，主なトランスピュータ型要素プロセッサの仕様を示す．これらすべて (または，一部) に共通する特徴的な機能は，次の通りである．

- 並列処理の計算モデル: メッセージ交換
- 1チップ内にプロセッサ (P)，メモリ (M:または，キャッシュ) およびスイッチ (S) を集積
- ハードウェア (HW) によるマルチスレッド処理支援
- 直接網を想定し，複数の物理チャネルを装備
- 仮想チャネルおよびルーティング機能のサポート

## 5.2 マルチスレッド処理プロセッサ

マルチプロセッサ・システム環境では，通信 (リモートメモリ・アクセス，キャッシュライン・フェッチ，キャッシュ・コヒーレンス保証，同期，メッセージ伝達，等) に伴うレイテンシ (*latency*) に対して如何に対処するか (レイテンシ・トレランス: *latency tolerance*) が重要な課題である．これには，次の方法がある．

[4] さらに，次世代 transputer T9000(コード名 H1) が発表済みである [3]．

表 3: トランスピュータ型要素プロセッサ

| 諸元 | | T800-30[27] | T9000[27][3] | iWarp[15][16] | MDP[19][21] |
|---|---|---|---|---|---|
| 計算モデル | 並行処理 | CSP/Occam 他 | | systolic array 他 | actors/CST 他 (PMI) |
| | 並列処理 | メッセージ交換 | | メッセージ交換 | メッセージ交換 |
| 電気的・機械的仕様 | トランジスタ数 | 25 万? | 230 万 | 65 万 | ? |
| | ダイ・サイズ | ? | $18.5 \times 9.95 mm^2$ | $14 \times 14 mm^2$ | ? |
| | パッケージ | 84 ピン PGA | 208 ピン QFP | ? | ? |
| | 動作周波数 | 30MHz | 50MHz | 20MHz | ? |
| プロセッサ (P) | 命令形式 | 8/16/24/32-bit 命令<br>スタック演算 | | 32/96-bit 命令<br>レジスタ-レジスタ演算 | 17-bit 命令<br>レジスタ-メモリ演算 |
| | レジスタ数 | 32-bit×6 語×1 組 | | 32-bit×128 語×1 組 | 36-bit×9 語×2 組 |
| | プロセッサ内並列処理 | 命令パイプライン | スーパースカラ<br>[最大 8 命令] | VLIW<br>[最大 9 操作] | 命令パイプライン? |
| | プロセッサ内並行処理 | 粗粒度マルチスレッド処理<br>[2 優先度 HW スケジューラ] | | — | 粗粒度マルチスレッド処理 [2 優先度 HW スケジューラ] |
| | ピーク性能<br>(単精度/倍精度) | 30MIPS<br>4.3MFLOPS | 400MIPS<br>25MFLOPS<br>40/32MWIPS | 20MIPS<br>20/10MFLOPS | 4MIPS |
| メモリ (M) | 内部 ROM | — | — | 2KBytes | 有 |
| | 内部キャッシュ | — | 16KBytes | 1KBytes(命令) | — |
| | 内部 RAM | 4KBytes | — | — | 36-bit×4K 語 |
| | 内部 RAM バンド巾 | 120MBytes/s | — | — | 3Gbits/s |
| | 外部ローカルメモリ・アクセス [バンド巾] | 可<br>[40MBytes/s] | 可<br>[200MBytes/s] | 可<br>[160MBytes/s] | 可<br>[?] |
| | リモートメモリ・アクセス [バンド巾] | — | — | — | — |
| | 仮想記憶 | — | — | — | サポート |
| | 記憶保護 | — | サポート | — | サポート |
| スイッチ (S) | 通信モデル | 同期式メッセージ伝達 | | 同期式/非同期式<br>メッセージ伝達 | 非同期式メッセージ伝達 |
| | 転送方式 | DMA 転送<br>[ユーザ領域→ユーザ領域] | | • DMA 転送;<br>– システム領域→システム領域<br>– ユーザ領域→ユーザ領域<br>• プログラム・モード転送<br>[レジスタ→レジスタ] | プログラム・モード転送<br>[レジスタ/メモリ→メモリ内受信キュー] |
| | 物理チャネル数 | 4 双方向チャネル<br>(2 単方向ポート/チャネル) | | 8 単方向チャネル<br>(=4 入力+4 出力) | 6 双方向チャネル? |
| | 仮想チャネル [数] | — | サポート<br>[?] | サポート<br>[最大 20 入力+20 出力] | サポート<br>[?] |
| | ルーティング | — | ? | • 仮想チャネル+ wormhole<br>[トポロジ適応型]<br>• バッファ・プール+ store-and-forward | 仮想チャネル+ wormhole<br>[非適応型?] |
| | 通信バンド巾 (/ポート)<br>(/チャネル) | 20Mbits/s<br>2.35MBytes/s | 100Mbits/s | 40MBytes/s | 450Mbits/s |
| システム | 想定ネットワーク [トポロジ] | 直接網 | | 直接網<br>[2-D torus 等] | 直接網<br>[3-D mesh] |
| | 想定プロセッサ台数 | | | 1,024 | 65,536 |

- 低レイテンシ化 (*latency reduction*): レイテンシそのものを低減する．たとえば，キャッシュの多階層化，キャッシュ・コヒーレンス保証方式の改良 [18]，同期方式の改良 [4]，転送の高速化，等による．
- レイテンシ隠蔽 (*latency hiding*): レイテンシがあたかも存在しないように，これを隠蔽する．これには，次の方法がある．
  - ノンブロッキング化 (*non-blocking*): たとえば，以下の技術がある．
    * キャッシュライン・プリフェッチ
    * ノンブロッキング・キャッシュ
    * ノンブロッキング・ロード命令
    * スプリット転送 (*split-phase transaction*) 方式
    * パイプライン転送方式
    * 非同期式メッセージ伝達
  - マルチスレッド処理 (*multithreading*): レイテンシを伴う通信を行ったスレッドをサスペンドし，別のスレッドに切り替える．すなわち，プロセッサ内並行処理を行なう．

上記の各技術は他のいずれとも排他的関係にはなく，任意の組合せが可能である．このうち，マルチスレッド処理をハードウェアで支援するアプローチが近年注目を集めている．これは，論理的な並行度が物理的な並列度より十分大きい場合，レイテンシが完全に隠蔽できるからである[5]．

ハードウェアによりマルチスレッド処理を支援するプロセッサを "マルチスレッド処理プロセッサ (*multithreaded processor*)" と呼ぶことにする．これらマルチスレッド処理プロセッサは，その歴史的背景の相違により以下の 2 つのアプローチに分類される．

- *shared resource MIMD*[22]:
  - **HEP**[35]: Denelcor 社の Smith 等
  - **Horizon**[37]/**Tera Computer**[13]: Tera Computer 社の Smith 等
  - **transputer**[27]: Inmos 社
  - **MDP**[19][21]: MIT の Dally 等および Intel 社
  - **CPC**[33]: 東大の後藤 等
  - **MASA**[23]: MIT の Halstead 等
  - **APRIL**[12]/**Sparcle**[28]: MIT の Agarwal 等
  - Stanford 大の Gupta 等 [39]
  - 東大の森下 等 [11]
- *dataflow/von Neumann hybrid*[17][26]:
  - **P-RISC**[30]: MIT の Arvind 等
  - **Monsoon**[31]: MIT の Arvind 等および Motorola 社
  - **EMPIRE**[26][7]: IBM の Iannucci 等
  - **CODA**[5]: 電総研の戸田 等

[5] この性質を "*parallel slackness*" と言う [38].

上記のいずれの分類においても，マルチスレッド処理をハードウェアで支援するに当たっては，以下の項目を検討する必要がある．

- 粒度 (スレッド切替間隔)[34]:
  - 細粒度マルチスレッド処理 (*fine-grain multithreading*): 毎クロック・サイクル，スレッドを切り替える (*cycle-by-cycle interleaving*)．たとえば，HEP, Horizon, Tera, CPC, MASA, P-RISC, Monsoon, [11].
  - 粗粒度マルチスレッド処理 (*coarse-grain multithreading*): スレッドの実行をブロックする要因が発生するまで，当該スレッドを実行する．その要因には，以下のものがある．
    * データ依存関係による同期待ち: APRIL, EMPIRE, CODA
    * リモートメモリ・アクセス要求: APRIL
    * キャッシュ・ミス: [39]
    * 共有データへのライト・ヒット: [39]
    * 同期式メッセージ伝達における同期待ち: transputer
    * 優先度の高いメッセージの受信: MDP
- スレッド切替制御 〔方針〕:
  - ハードウェア 〔round-robin〕: CPC
  - ハードウェア 〔FIFO〕: transputer, MDP, P-RISC, Monsoon, EMPIRE
  - ハードウェア 〔戦略的〕: HEP, Horizon, Tera
  - ソフトウェア: APRIL, CODA
- 演算結果の主な格納場所:
  - レジスタ 〔組数〕: HEP 〔128〕, Horizon 〔128〕, Tera 〔128〕, MASA, APRIL 〔4〕, CODA 〔8〕
  - メモリ: transputer, MDP, P-RISC, Monsoon, EMPIRE
- 実行待ちスレッド状態の存在場所:
  - レジスタ 〔スレッド数〕+メモリ: HEP 〔128〕, Horizon 〔128〕, Tera 〔128〕, MASA 〔8 または 16〕, APRIL 〔4〕, CODA 〔8〕
  - トークン・キュー: P-RISC, Monsoon, EMPIRE
  - メモリ: transputer, MDP
- 同期をとる場所 〔同期機構〕:
  - 共有メモリ 〔full/empty ビット+busy-wait〕: HEP, Horizon, Tera
  - 共有メモリ 〔full/empty ビット+suspend=I-structure〕: MASA, P-RISC, Monsoon, EMPIRE
  - 共有メモリ 〔full/empty ビット+trap〕: APRIL, CODA
  - レジスタ/フレーム・メモリ 〔full/empty ビット〕: HEP, Monsoon
  - フレーム・メモリ 〔full/empty ビット+thread-switch〕: EMPIRE
  - レジスタ 〔full/empty ビット+trap〕: CODA

## 6 おわりに

汎用超並列マルチプロセッサ・システムの構築に適したプラットフォーム要素プロセッサ・アーキテクチャを検討するに当たって，その検討姿勢および方針，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を述べた．

まず，以下の3つの性質をプラットフォーム要素プロセッサに求めた．

- P: ポータビリティ(*Portability*)
- M: モジュラリティ(*Modularity*)
- S: スケーラビリティ(*Scalability*)

その方向において，transputer を始めとする iWarp, MDP 等のトランスピュータ型要素プロセッサは，検討のための良いたたき台を提供している．これらトランスピュータ型要素プロセッサの仕様をまとめ，共通する機能を示した．

また，シングルプロセッサ・システム環境とは異なり，マルチプロセッサ・システム環境ではレイテンシ・トレランスが大きな技術的課題となる．そのための諸技術を，以下の側面に関して示した．

- 低レイテンシ化
- レイテンシ隠蔽:
  - ノンブロッキング化
  - マルチスレッド処理

このうちマルチスレッド処理プロセッサに注目し，その要検討項目を述べた．マルチスレッド処理を始めとして，ここで挙げた諸技術の各々については，今後より詳細に検討を行なっていく予定である．

実用ないし研究用の汎用超並列マルチプロセッサ・システムを優れた価格対性能比で構築可能とするため，国内からプラットフォーム要素プロセッサ[6]が一刻も早く登場することを切に希望する．

## 謝辞

富田眞治 京都大学工学部教授 (九州大学大学院総合理工学研究科教授 併任)，ならびに，日頃ご討論頂く九州大学富田研究室の諸氏に感謝致します．

## 参考文献

[1] 小柳, 田辺, "超並列マシンの実現技術," 情報処理, vol.32, no.4, pp.365–376, 1991年4月.

[2] 小池誠彦, "超並列マシン," 情報処理, vol.28, no.1, pp.94–105, 1987年1月.

[3] 佐藤康朗, "Transputer T9000を正式発表，ピーク性能は400MIPS," 日経エレクトロニクス, no.527, pp.120–121, 1991年5月.

[4] 高木, 有田, 曽和, "細粒度並列実行を支援する種々の静的順序制御方式の定量的評価," 並列処理シンポジウム *JSPP'91* 論文集, pp.269–276, 1991年5月.

[5] 戸田, 西田, 内堀, 島田, "マクロデータフロー計算機CODA —アーキテクチャ —," 並列処理シンポジウム *JSPP'90* 論文集, pp.185–192, 1990年5月.

[6] 馬場敬信, "超並列マシンへの道," 情報処理, vol.32, no.4, pp.348–364, 1991年4月.

[7] 平木 敬, "プロトタイプハイブリッド・データフロー計算機のアーキテクチャ," 並列処理シンポジウム *JSPP'90* 論文集, pp.177–183, 1990年5月.

[8] 藤井, 新實, 柴山, "超並列計算機システムにおける要素プロセッサ・アーキテクチャの検討," 信学技報, CPSY–90–45, 1990年7月.

[9] 村上和彰, "スーパースカラ・プロセサの性能を最大限に引き出すコンパイラ技術," 日経エレクトロニクス, no.521, pp.165–185, 1991年3月.

[10] 村上和彰, "ハイパースカラ・プロセッサ・アーキテクチャー 命令レベル並列処理への第5のアプローチ—," 並列処理シンポジウム *JSPP'91* 論文集, pp.133–140, 1991年5月.

[11] 森下 巖, "多段結合ネットワークを用いる超並列マシンのためのパイプライン化MIMDプロセッサ," 情報処理学会論文誌, vol.31, no.4, pp.523–531, 1990年4月.

[12] Agarwal, A., Lim, B.-H., Kranz, D., and Kubiatowicz, J., "APRIL: A Processor Architecture for Multiprocessing," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.104–114, June 1990.

[13] Alverson, R., Callahan, D., Cummings, D., Koblenz. B., Porterfield, A., and Smith, B., "The Tera Computer System," *Proc. 1990 Int'l. Conf. Supercomput.*, pp.1–6, June 1990.

[14] Athas, W. C. and Seitz, C. L., "Multicomputers: Message-Passing Concurrent Computers," *Computer*, vol.21, no.8, pp.9–24, Aug. 1988.

[15] Borkar, S. et al., "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proc. Supercomuting'88*, pp.330–339, Nov. 1988.

[16] Borkar, S. et al., "Supporting Systolic and Memory Communication in iWarp," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.70–81, June 1990.

[17] Buehrer, R. and Ekanadham, K., "Incorporating Data Flow Ideas into von Neumann Processors for Parallel Execution," *IEEE Trans. Comput.*, vol.C-36, no.12, pp.1515–1522, Dec. 1987.

[6] すなわち，和製トランスピュータ第1号.

[18] Dubois, M. and Thakkar, S. (Eds.), "Cache Architectures in Tightly Coupled Multiprocessors," *Computer*, vol.23, no.6, pp.9–83, June 1990.

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

[20] Dally, W. J. and Wills, D. S., "Universal Mechanisms for Concurrency," *Proc. PARLE'89 Parallel Architectures and Languages Europe*, pp.19–33, Lecture Notes in Computer Science 365, Springer-Verlag, June 1989.

[21] Dally, W. J. et al., "The J-Machine: A Fine-Grain Concurrent Computer," *Proc. IFIP 11th Computer Congress*, pp.1147–1153, Elsevier Science Publishers, Aug. 1989.

[22] Flynn, M. J., "Some Computer Organizations and Their Effectiveness," *IEEE Trans. Comput.*, vol.C-21, no.9, pp.948–960, Sept. 1972.

[23] Halstead, R. H., Jr. and Fujita, T., "MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing," *Proc. 15th Int'l. Symp. Comput. Architect.*, pp.443–451, May 1988.

[24] Hey, A. J. G., "Supercomputing with Transputers — Past, Present and Future," *Proc. 1990 Int'l. Conf. Supercomput.*, pp.479–487, June 1990.

[25] Hill, M. D., "What is Scalability?," *ACM SIGARCH Comput. Architect. News*, vol.18, no.4, pp.18–21, Dec. 1990.

[26] Iannucci, R. A., "Toward a Dataflow/von Neumann Hybrid Architecture," *Proc. 15th Int'l. Symp. Comput. Architect.*, pp.131–140, May 1988.

[27] INMOS Ltd., *The Transputer Databook*, INMOS Ltd., Nov. 1989.

[28] Kurihara, K., Chaiken, D., and Agarwal, A., "Latency Tolerance through Multithreading in Large-Scale Multiprocessors," *Proc. Int'l. Symp. Shared Memory Multiprocessing*, pp.91–101, Apr. 1991.

[29] Murakami, K., Mori, S., Fukuda, A., Sueyoshi, T., and Tomita, S., "The Kyushu University Reconfigurable Parallel Processor — Design Philosophy and Architecture —," *Proc. IFIP 11th Computer Congress*, pp.995–1000, Elsevier Science Publishers, Aug. 1989.

[30] Nikhil, R. S. and Arvind, "Can Dataflow Subsume von Neumann Computing?," *Proc. 16th Int'l. Symp. Comput. Architect.*, pp.262–272, May 1989.

[31] Papadopoulos, G. M. and Culler, D. E., "Monsoon: an Explicit Token-Store Architecture," *Proc. 17th Int'l. Symp. Comput. Architect.*, pp.82–91, May 1990.

[32] Scott, S. L., "A Cache Coherence Mechanism for Scalable, Shared-Memory Multiprocessors," *Proc. Int'l. Symp. Shared Memory Multiprocessing*, pp.49–59, Apr. 1991.

[33] Shimizu, K., Goto, E., and Ichikawa, S., "CPC (Cyclic Pipeline Computer) — An Architecture Suited for Josephson and Pipelined-Memory Machines," *IEEE Trans. Comput.*, vol.38, no.6, pp.825–832, June 1989.

[34] Skillicorn, D. B. and Woodside, W., "Grain Size in Multi-Threaded Systems," submitted to *The Mathematical Scientist*, 1991.

[35] Smith, B. J., "A Pipelined, Shared Resource MIMD Computer," *Proc. 1978 Int'l. Conf. Parallel Processing*, pp.6–8, Aug. 1978.

[36] Smith, B., "The End of Architecture," *Keynote Address presented at 17th Ann. Int'l. Symp. Comput. Architect.*, June 1990; *ACM SIGARCH Comput. Architect. News*, vol.18, no.4, pp.10–17, Dec. 1990.

[37] Thistle, M. R. and Smith, B. J., "A Processor Architecture for Horizon," *Proc. Supercomputing'88*, pp.35–41, Nov. 1988.

[38] Valiant, L. G., "General Purpose Parallel Architectures," *Handbook of Theoretical Computer Science*, Elsevier Science Publishers and MIT Press, 1990.

[39] Weber, W.-D. and Gupta, A., "Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results," *Proc. 16th Int'l. Symp. Comput. Architect.*, pp.273–280, May 1989.