第 1 回 並列ゼミ

ゼミ担当者　：　輪湖純也，澤田淳二，降幡建太郎
指導院生　：　児玉憲造，下坂久司
開催日　：　2002 年 4 月 12 日

ゼミ内容: 本ゼミでは，並列処理，すなわちひとまとまりの処理を複数の CPU を用いて行う方法について学ぶ．ここでは，並列処理を対象問題，アルゴリズム，ソフトウェア，ハードウェアの各レベルに分け，問題領域からハードウェアに至るまで並列処理の概観を説明する．

1　はじめに

並列処理（Parallel Processing）とは,「コンピュータで、一連の処理を複数台の処理装置で同時に並行して行うこと」（大辞林）である．並列処理というものを, 日常の仕事に探すとすれば, 実は至る所に存在している．　例えば, 生協購買部のレジ, 高速道路のレーン, 興戸駅の自動改札である．　これらの処理は, 本質的には, これから説明する並列処理の大前提となる考えをはらんでいる．

2　並列処理のイメージ

並列処理を，次の 4 つのレベルに分けて考える．

- **対象問題**
- **アルゴリズム**
- **ソフトウェア**
- **ハードウェア**

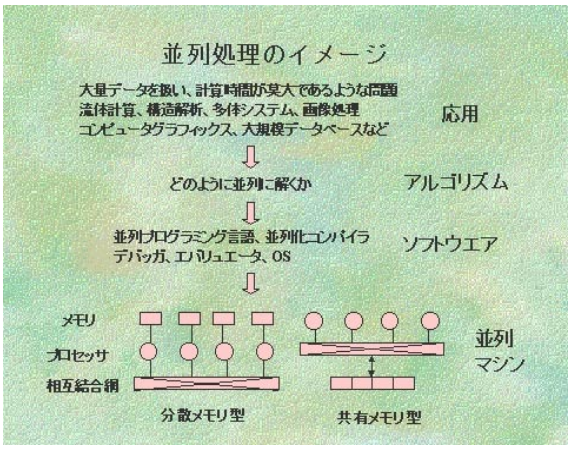

Fig. 1　並列処理のイメージ

3　並列処理で解くべき対象問題とは何か？

なぜ並列処理を行う必要があるのだろうか？それは，逐次処理では（実時間内で）解けなくて，並列処理を行えば実時間内に解ける問題が存在しているからである．

例えば,「3 次元 CG 処理」というものがこれにあたる．3 次元 CG は，レイ・トレーシングと呼ばれる手法を用いて計算できるが，これは一般に膨大な計算量を要し，逐次処理では限界がある．このレイ・トレーシングを並行処理する並行レイトレーシングと呼ばれる手法を用いることにより，3 次元 CG を描くことができるのある．（Fig.2 参照）

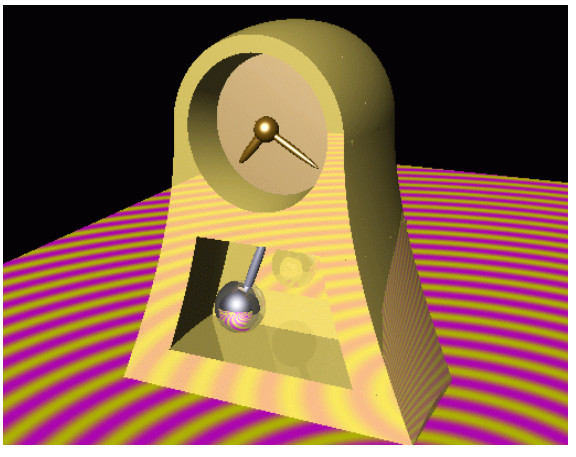

Fig. 2　3 次元 CG 処理の例

4　並列処理の必要性

ここでひとつ疑問が生じる．わざわざ複数のプロセッサを用いて並列処理を行うよりも，プロセッサ自身の計算速度を上げれば，問題は解決するのではないか？

結論から言うと，答えは否である．なぜならば，プロセッサ単独の計算速度は，今や限界に近づいてきているからである．

同一種類のプロセッサの計算速度は，クロック周波数に依存している．このクロック周波数が 1GHz になると周期は 1ns となり，電気信号はその間に 30cm しか進まず，集積回路の大きさや配線の基盤の寸法を考えると，これ以上クロック周波数を大幅に上げることは原理的に困難になってくる．

一方，1ns より早いスイッチング速度を持つ半導体材料についても極めて限られてくる．材料の面からいっても限界は近い．

さらに，いくらプロセッサが高速に動作してもメモリーからのデータの供給がなければ必要な演算を行うことはできず，メモリーのアクセス速度の高速化が困難な現状では単独のプロセッサの演算速度はかなり限界に近づいたと言える．

以上のような理由から，大規模シミュレーションなどの力業を行うコンピュータは必然的に超並列計算機となり，並列処理技術の高度化が次の時代を開く．

5　並列処理の問題点

並列処理には, 以下の 2 点の問題点が存在している．

1. 仕事の分担の自動化は難しく, コスト (手間) がかかる．
2. 仕事には, 段取り, 順序が存在する．

1 に関しては, 実際に並列処理を実現するプログラミングモデルを考えれば, 容易にわかるであろう. 個々の問題には, その問題独自の性質が存在し，問題間で一般性を見つけ出すことは難しい. 人が, 毎回プログラムの際に, 仕事の分担を考えていては, そのコストは大きなものとなってしまう．

また, 仕事の分担に対しては, 通信のレイテンシ (遅延) を考えた分割法など, 仕事そのもののタスク[1]の振り分けのみならず, そのハードウェアのバックグラウンドまで考える必要が生まれる. 少なくとも「通信時間 $\ll$ 計算時間」という関係が, 成立する仕事でなければ, 仕事を分担した意味がなくなってしまう. 以上から, 仕事の分担は並列処理を考える上で, 第一に問題となるものである．

2 に関しては,1 で述べたことに付随する形で存在する. つまり仕事には順序が存在し, その順序の破壊は, 即ちシステムとしての破壊を生む. 次に述べることは, 逐次プログラムでもいえることであるが, 並列処理では通信を多用するため, タイミングによりデータのハザード[2]や, デッドロック[3]が起きる可能性がある. しかも, これらのミスは頻繁に起きやすい.(Fig. 3 参照)

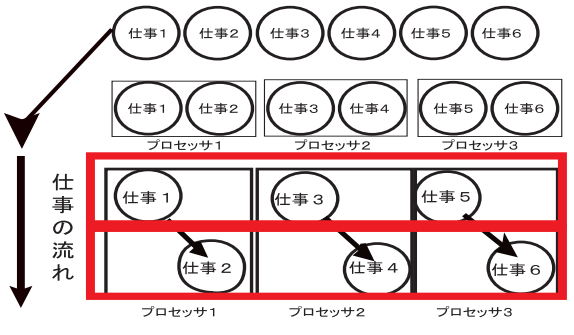

Fig. 3　仕事の割り振り

[1]オペレーティングシステム（OS）の処理の単位
[2]ある命令の入力が直前の命令の結果に依存すること
[3]2 つのプログラムが互いに相手の資源が解放されるのを待ち合うこと

6　並列処理の効率

並列処理において最も重要な観点は複数のプロセッサを効率的に動作させることであり，このためには対象問題，アルゴリズム，ソフトウェア，ハードウェアの種々の局面で内在する並列性を抽出しなければならない．

Fig.4 は，問題領域からハードウェアに至るまでにおける並列性の度合いを示したものである．縦軸は並列して行える独立なオペレーションの程度を示している．この図は，より対象問題に近い段階で並列化を行うほうが並列性の度合いが高い，つまりより並列化の効果が出せる（対象問題を速く解ける）可能性があるということを意味している．

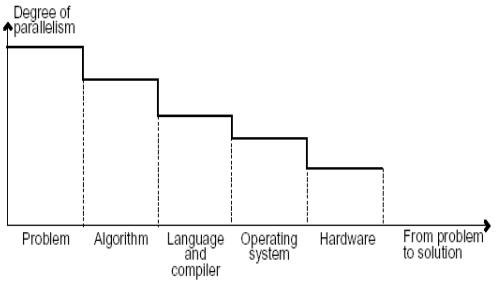

Fig. 4　各レベルにおける並列性

並列処理の数学的な効率としては，**Amdahl の法則**がある．これは，並列処理の部分とそうでない部分の比率の関数として全体の処理速度を表したものである．s $(0 < s < 1)$ を全体の処理における逐次処理可能な処理の割合とし，p をプロセッサ数，$R(s)$ を総合的な速度向上率とすると，$R(s)$ は次式で表せる．

$$R(s) = \frac{1}{s + \frac{1-s}{p}} \tag{1}$$

これを，グラフにすると以下の Fig. 5 のようになる. X 軸が, プロセッサ数であり,Y 軸が逐次処理した際との速度の向上率である. 上から順に,$s = 0.1, 0.2, 0.5$ である. パフォーマンスは,$s = 0.5$ では 100 台のプロセッサを使用したとしても, わずか「速度向上は 2 倍に及ばない」．

また，逐次処理部分の割合がたとえ 5%と少なくても（並列処理部分が 95%でも），1 万台のプロセッサで得られるスピードアップはわずか 20 倍にしかならない．いかに，逐次処理部分の割合が全体の効率を下げているかが分かる．

並列処理は, 並列化効率と使用するプロセッサの数とのコストパフォーマンスが最高となるところで実行されるべきである．

並列化効率 E_p は，簡単には次の式 (2) で表される. すなわち，p 個のプロセッサを用いたときの効率は 1 個のプロセッサで行ったときの逐次処理計算時間 T_1 を並

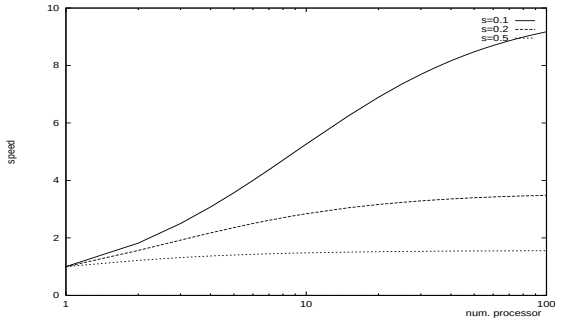

Fig. 5　Amdahl の法則

行処理したときの計算時間 T_p と p との積で除したものである．

$$E_p = \frac{T_1}{pT_p} \tag{2}$$

理想的には $E_p = 1(T_1/T_p = p)$ となるが，現実には並列処理に伴う種々のペナルティ要因 (並列化できない部分があること，CPU 時間のオーバーヘッドなど：Fig. 6) のために通常 $E_p < 1(T_1/T_p < p)$ となる．オーバーヘッドの主な原因は, タスクを分けたことによる通信処理や, タスク自体の制御によるところが多い.

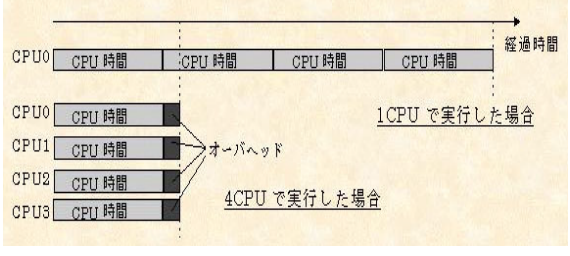

Fig. 6　並列処理に置ける CPU 時間のオーバヘッド

並列処理における効率に重大な影響を及ぼすもう一つのファクターは，**プロセッサ間通信の速度**である．この速度が遅く，しかもプロセッサ間通信が多い場合には並列処理の効率は著しく悪くなる．

プロセッサ間通信の頻度は処理の**粒度**に関係している．粒度とは，複数のプロセッサに与えられるタスクの大きさのことである．一つのステートメントあるいはそれ以下のタスクを細粒度，ループまたはサブルーチンレベルのタスクを粗粒度，それらの中間的な大きさのタスクを中粒度とよぶ．粒度が小さくなればプロセッサ間通信が多くなり，反対に粒度が大きくなればプロセッサ間通信は少なくなる．このため，プロセッサ間通信の速度が遅く，細粒度の場合は並列処理の効率が著しく悪くなる．

さらに，効率に重大な影響を及ぼすもう一つのファクターは，**ロードバランス**である．ロードバランスとは，並列計算機の各プロセッサに与えられる計算負荷の均等性のことであり，並列処理の効率を高めるためには，できるだけロードをバランスさせなければならない．

各プロセッサに与えられる負荷がが不均一であると，処理に不可避な同期待ちが多くなり，処理効率は著しく悪化する．例えば，負荷が 2 倍異なれば計算時間が 2 倍異なり，同期をとるために遅いプロセッサを待つことになり，いくらほかのすべてのプロセッサが早く処理を済ませても一番遅いプロセッサに律速される．このため，処理効率はこのことだけで 50%程度に悪化する．

こうした問題を克服するため，各プロセッサに与える負荷はできるだけ均一になるようにしなければならない．しかしながら，これは難しい問題である．なぜなら，画像処理などのようにデータの量に応じて負荷が決まる場合はデータを分散させればロードバランスは取れるが，連立一次方程式の Gauss-Seidel 法に基づく解法や傾斜法に基づく最適化計算など，多くの繰り返し計算手法ではデータの質（内容）によって収束までの時間が異なるからである．

7　並列処理のためのアルゴリズム

処理の並列化を行うためには，どのように並列化を行えばよいか，ということを考える必要がある．つまり，問題のどの部分が逐次的ではなく，複数のコンピュータで同時に処理が可能かを考える必要がある．

並列処理のためのアルゴリズムの効果的な利用例として, ここでは GA を挙げる.GA に並列処理を導入することにより, 逐次的に処理を行っていれば, 膨大な計算時間が必要とされる, 何世代にもわたる遺伝操作が, はるかに短時間に行えるようになる. さらに,GA は並列処理を行える構成要素が多いため, 容易に並列化が行える点でも並列処理に適している.

GA の高速計算のための並列化は, おおまかに, 適応度評価を並列処理する単純並列アプローチ, 個体集合を分割し, 独立して遺伝的操作を行う分解型アプローチの二つに分けることができる. ここでは, 単純並列アプローチについて説明する.GA では, 各個体の適合度は他の個体の適応度とは独立して計算可能であるから, この点に着目して,Fig. 7 のように, 適応度関数の評価を並列処理することにより高速化を図ろうとするのが, この手法である. ただし, 適応度評価が高速に行える問題の場合, 各プロセッサ間の通信速度がネックになって, 操作全体でみると, 全く高速化されないどころか, 逐次処理の方が速いこともある.

このように, 並列処理のためのアルゴリズムは, 問題の性質を分析し, 並列処理に向いている処理部分に応じて, 適切なものを選択しなければならない.

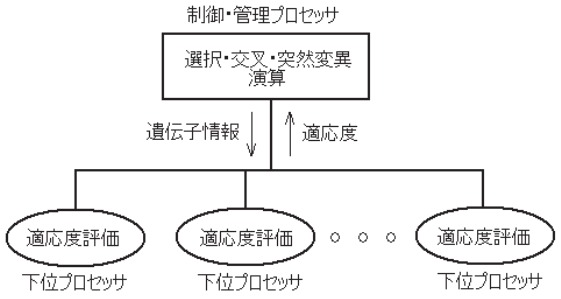

Fig. 7　GA の単純並列アプローチ

8　並列処理のためのソフトウェア機構

並列化を実現するためには，アルゴリズムを考えるだけではなく，並列化をサポートするためのソフトウェア機構を用いることも重要である．

各ソフトウェア機構は利点もあれば欠点も存在する．よって，どの方法が並列処理にもっとも向いているかは決定できない．対象となる問題によってどのソフトウェア機構を用いるかが変わってくることになる．

8.1　プロセスとスレッド

8.1.1　プロセス

プログラムの実行単位をプロセスという．各プロセスにはそれぞれ独立したメモリ空間が割り当てられる．どのプロセスもほかのプロセスの持っているメモリ空間にアクセスすることはできない．このため，プロセス間のデータのやりとりをしたい場合は，Fig. 8 のように外部的・内部的な何らかの通信手段を介して行うことになる．つまり，プログラム中にデータの受け渡しを明示的に示す必要がある．

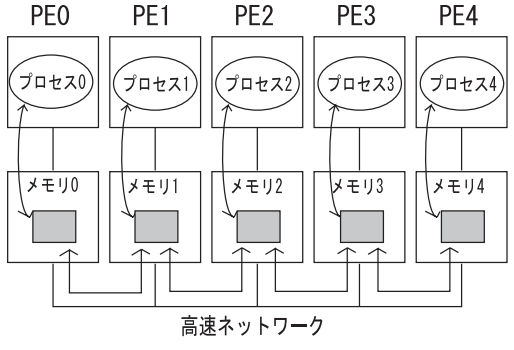

Fig. 8　プロセスの構造

プロセスを用いた方法では，プロセス内のメモリ空間が保護されているという点では安全であるが，プロセス間の通信量が多い場合には非常に非効率である．

8.1.2　スレッド

一般に 1 プロセスが 1 プログラムに対応する．しかしながら，プロセスの中にはさらに分けることができる部分が存在する．それをスレッドとして扱うことができる．あるプロセス内の複数のスレッドは Fig. 9 のように同じメモリ空間を共有するため，プロセスと比べて，ダイレクトにメモリをアクセスできるため，パフォーマンスがよい．

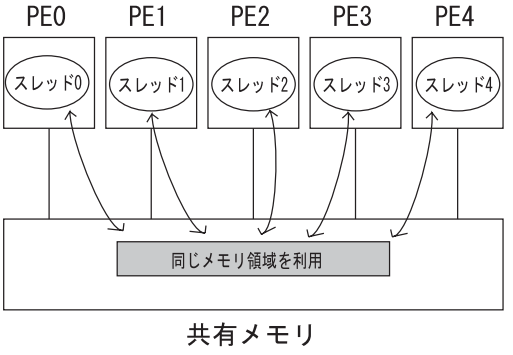

Fig. 9　スレッドの構造

メモリに関して，プログラマが管理する例として，データを読み込んで再び書き込むまでの排他制御があげられる．たとえば，複数のスレッドが同時に同じデータ領域に書き込むことがないように，片方のデータの書き込み (読み込み) を制御する必要がある．これを排他制御という．

8.2　MPI

MPI(Message Passing Interface) とは，分散メモリ環境における並列プログラミングの標準的な実装である．Message Passing とは，プロセッサ間で互いに通信してメッセージ（データ）交換を行ったり，同期を取ったりすることを指している．しかし，通信といっても様々な問題が存在する．たとえば，同じ並列処理マシンでの通信や TCP/IP によるネットワーク越しの通信もある．並列処理マシンのアーキテクチャにより，通信方法が異なるために，その実装が全く異なったものになってしまう．この部分を代行してくれるものが MPI である．

8.3　OpenMP

OpenMP とは，共有メモリ型による並列化を記述するための API(Application Program Interface) の仕様のことである．

9　並列処理のためのハードウェア

9.1　並列コンピュータの分類

並列コンピュータの分類には，Flynn の分類が広く用いられている．これは，命令の流れとデータの流れが単

一か複数かで計算機を Fig. 10 の 4 種類に分類するものである．

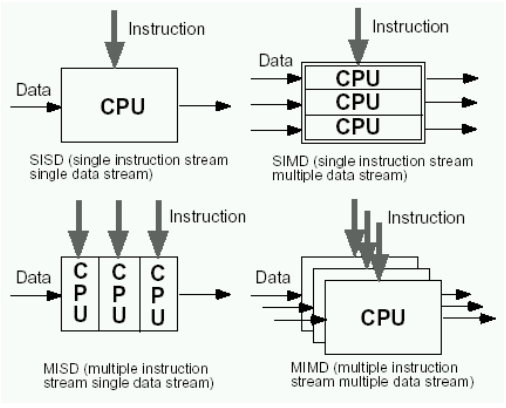

Fig. 10　並列コンピュータの分類

並列コンピュータのアーキテクチャはまず次の二つに大別される．それらは，SIMD(Single Instruction Multiple Data) と MIMD(Multiple Instruction Multiple Data) である．

前者では多くのプロセッサが局所的なメモリーを有し，すべてのプロセッサは完全な同期の下でコントローラーから送られる単一の命令を同時に実行する．そしてその結果が集約されて一つの結果となる．この方式の利点は次の通りである．

- 同期をとるためのオーバーヘッドがない
- 単純なプロセッサで良いのでプロセッサ数を多くできる
- 負荷分散に対する要求は多いので利用価値が高い

逆に欠点も存在する．

- 同期をとるためのハードウエアとブロードキャストおよび演算結果の集約を効率化する仕組みが重要になる
- 負荷を一様に分散させることが困難

一方，後者では各プロセッサは異なった処理を行うことができる．すなわち，プロセッサごとにコントローラーが独立して配置されており，SIMD アーキテクチャよりも柔軟な利用が可能となる．

9.2　並列コンピュータの構成方法

並列コンピュータ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をどのように行うか，および，メモリーをどのように構成するかである． プロセッサ間通信は以下の観点から分類する．

並列コンピュータの構成方法として，共有メモリ型と分散メモリ型がある．ここでは，それぞれについて説明する．

9.2.1　共有メモリ型

複数のプロセッサがメモリバス／スィッチ経由で，主記憶に接続される形態である (Fig. 11)．このアーキテクチャを有するシステムのことを SMP と呼ぶ．このシステムは，いうならば巨大なマザーボード上に複数個のプロセッサを取り付けた形である． 最近はやりのデュアルマザーはこの手のシステムに相当する．これらを有効に使用するためには OS でのプロセッサ間の通信制御，プロセッサの割り当てが必要となる．

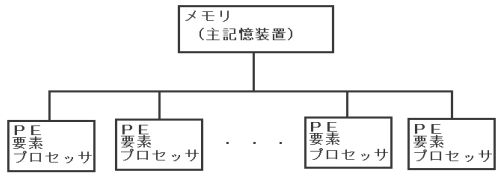

Fig. 11　共有メモリ型の構造

この形態の特徴としては，複数のプロセス (後述) による並列化のみならず，スレッド (後述) という概念を用いたスレッド単位の並列処理が行われる．これらを用いる理由としては, 同じプロセス中のスレッド間では同じメモリ領域にアクセスすることができる. つまり, マルチプロセスモデルでは, データの受け渡しは通信を介して行われたが, マルチスレッドでは, 得たいデータ領域に対して直接そのアドレスを参照することができる. 逆に欠点は，他タスク・他ジョブとのメモリアクセス競合による性能低下が発生することである．

共有メモリ型は分散メモリ型と比較して次の点で優れている．

- プログラミング時にデータ分割を考慮する必要が無く，容易に自動並列化が可能．
- メモリ間通信が無いため，プロセッサ数が一桁程度の範囲では実効性能の理論最大性能に対する落ち込みが少ない．

9.2.2　分散メモリ型

プロセッサと主記憶から構成されるシステムが複数個互いに接続された形態である (Fig. 12)．つまり，一連のコンピュータが内部ネットワークまたは，外部ネットワークを介して通信することであたかも一つのコンピュータのように動作する．大規模なメモリを得るシステムが組みやすいが，その反面，メモリが分散しているためコンパイラによる自動並列化が困難であるという問題が存在する．

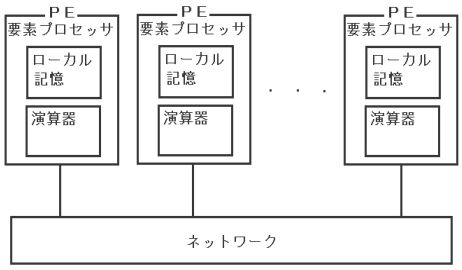

Fig. 12　分散メモリ型の構造

分散メモリ型コンピュータは以下の点で共有メモリ型コンピュータよりも優れている．

- 複数のマザーボードにメモリを点在できるので，巨大なメモリ空間を得るシステムが構築できる．
- 一般的に，同程度のシステムを組んだ場合，分散メモリ型，特に PC クラスタ (後述) では安価にシステムを構築できる．

しかし，程度の異なる PE(プロセッサーエレメント) を組む場合，個々の処理時間に差が生じるため，ネットワークを介してデータ交換を行う際，最も処理能力の低いコンピュータに依存してしまう．結果として処理効率が悪くなるという欠点も存在する．そのため計算負荷を調節するなどの処理が必要である．

9.3　専用アーキテクチャと汎用アーキテクチャ

並列コンピュータには，並列処理専用のアーキテクチャを備えた専用マシンと，特別には並列処理の機能を持っていない汎用アーキテクチャの組み合わせによって構成された 2 つの分類がある．

9.3.1　専用アーキテクチャ

専用アーキテクチャとは，並列処理専用のアーキテクチャを備えているコンピュータのことである．専用アーキテクチャでは，並列化を行うには非常に便利ではあるが，反面，専用の部品を使うために価格が非常に高価になってしまうという欠点もある．

専用アーキテクチャの例として, 専用のバスを備えた SMP マシンが挙げられる. 例えば Sun のエンタープライズサーバの一部は，これら専用ハードウェアによる共有メモリ型の並列コンピュータである．他にも，NUMA などが挙げられる．

9.3.2　汎用アーキテクチャ

汎用アーキテクチャとは，並列処理のための特別な機能を持っていないコンピュータのことである．このアーキテクチャでは，並列化のための機能を各マシンが持っていないために，その部分は別途用意する必要がある．しかし，安価にシステムが構築できるという利点がある．

汎用アーキテクチャの例としてクラスタが挙げられる．クラスタとは複数の独立したサーバからなるが，あたかも 1 つのサーバシステムのように機能するサーバグループを指す．クラスタを構築する最大の目的は信頼性が求められるシステムにおいて，万一何らかの問題が発生した場合でも，問題を起こしたサーバに代わってクラスタ内の他のサーバ (ノード) で処理を続行しミッションに対してクリティカルに対応できる環境を構築することを目指す．もう一方の目的として，タスク分割による並列処理というものもある．これは，より高速な計算環境を目指したものであるといえる．このシステムは分散メモリ型の並列コンピュータに分類される．これらシステムが重宝される理由として以下の 3 点が存在する．

1. 各マシンは PC レベルで十分である．
2. 昨今の PC の高性能化
3. PC 自体の価格の下落

1 にも現れているように, クラスタといってもコストを優先させるならば各マシンは PC で十分である. このことにより, 次の 2,3 の理由と相まって高速な計算環境を比較的安価に構築することができる．

10　並列処理における重要単語

10.1　伝送方式

クラスタの伝送方式には, イーサネットや MYRINET という規格がよく用いられる. クラスタにおいては, 通信速度が全体の処理速度に与える影響が非常に大きい. 逆に言えば, 高速通信, 効率の良い通信によって大きく全体の処理が大きく向上する．

イーサネット：イーサネットは，OSI 参照モデルにおけるデータリンク層の媒体アクセス制御副層 (MAC) において，CSMA/CD 方式 (Carrier Sense Multiple Access with Collision Detection：衝突検出型搬送波検知多重アクセス方式) を用いた，バス型もしくはスター型の LAN の規格である (図.13)．このイーサネットを標準化したものとして，IEEE802.3 標準 LAN があり，伝送速度や伝送媒体などの違いにより，10BASE5 や 10BASE‐T，100BASE‐TX といった規格が存在する．現在では 10BASE‐T や 100BASE‐TX といった，非シールドツイストペアケーブルを用いたイーサネットがよく用いられている．10BASE‐T は伝送速度が 10Mbps，100BASE‐TX は伝送速度が 100Mbps である．

ギガビット・イーサネット：ギガビット・イーサネットは，ネットワーク接続規格において現在広く使われている 10Mbps (10BASE-T) や 100Mbps (100BASE-T) ファースト イーサネットの拡張版にあたり，1Gbps のネットワーク帯域幅をサポートすることができる (図.14)．しかしながら, かつてのファースト イーサネットと同じく, 現在価格は 20 万円ほどであり, 汎用的に用いるには

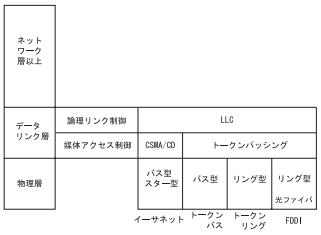

Fig. 13　イーサネットと OSI 参照モデル

厳しい状態である．

- イーサネットまたはファーストイーサネットと同じ伝送手順とフレーム・フォーマットを使うので，複雑で速度低下の要因となるエミュレーションや変換処理は不要である．一般的な通信プロトコルである TCP/IP をサポートしている．
- 既存のイーサネット，またはファーストイーサネットで利用している管理ツールをそのまま使用できる (図.15)．

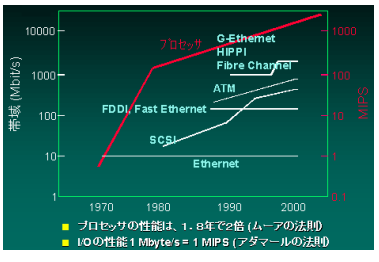

Fig. 14　ネットワークの進歩

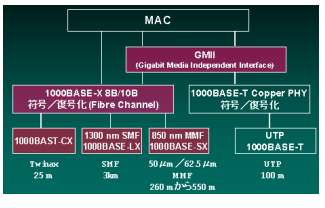

Fig. 15　物理層でのインタフェース

MYRINET：MYRINET は高性能なパケット通信及びスイッチングテクノロジーを持ち，PC クラスタのインターコネクトとして広く利用されている．MYRINET は次のような特徴を持つ．

- 全 2 重 2Gbps(第 3 世代 Lanai9 システム使用時) でリンクされるスイッチ及びインターフェイス
- フロー制御，エラー制御，リンクのモニタ
- 低レンテンシ，カットスルー型のクロスバースイッチ
- 障害発生時の代替経路の探索
- 様々なネットワークトポロジーを構築可能

Myrinet スイッチでは MTBF は 100 万時間を超え，Myrinet インターフェイスの MTBF は数 100 万時間にも及ぶ．Myrinet は非常に低いビット誤り率と 1 日あたり 1 ビット未満のエラーを何百ものホストのネットワークで実証し， ホスト， スイッチおよびケーブルの不具合に関しては非常に強固と言える．Myrinet はシステムフォルトが発生した時のため，これを回避するのに絶え間なく通信経路をトレースし，不具合が発生した時には代替ルートを使用する．ハードウェアは各リンク上でパケット落ちがないか CRC を計算しチェックする．最新の 64 ビットインターフェイスは，メモリと PCI バス上のパリティチェックを行う．これらのことから，MYRINET を使用したクラスタでは，ハイパフォーマンス・ハイアベラビリティを実現できる．

参考文献

1) 渡邊真也：MPI による並列プログラミングの基礎，http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/.

2) 三木光範：並列処理入門，http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/.

3) 三宮信夫，喜多一，玉置久，岩本貴司：遺伝的アルゴリズムと最適化，朝倉書店，1998.