第 1 回 並列ゼミ

ゼミ担当者　：　真武信和，山本啓二，坂田大輔
指導院生　　：　下坂久司，輪湖純也，冨岡弘志
開催日　　：　2003 年 4 月 15 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，通信ネットワーク，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．

ここでは並列処理を対象問題の例，その必要性，効率，問題点の順に述べ，最後に並列処理を次の 4 つのレベルに分けて並列処理の概観を捉える．

- アルゴリズム
- ソフトウェア
- 通信ネットワーク
- ハードウェア

2　並列処理の対象問題例

なぜ並列処理を行う必要があるのであろうか？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．このような問題を，Top500 Computer Sites [1)] において現在世界一の速度を誇る（Fig.1 参照）並列計算機である**地球シミュレータ**を例にして考えて行く．

地球シミュレータとは，気象庁が全世界の気候や地殻などの変動をシミュレートするために作り上げた超並列計算機である．この計算機によるシミュレーションにより，気候変動予測・水循環予測・地球温暖化予測・大気組成変動予測・生態系変動予測・地球内部変動メカニズムなどを解明することが目標とされている．Fig.2 は，地球シミュレータでの計算機シミュレーションの地球の格子イメージであるが，例えば大気・海洋シミュレーション[1]ではこのように大気・海洋を格子に区切り，各格子

Rank	Manufacturer Computer/Procs	R_{max} R_{peak}	Installation Site Country/Year
1	**NEC** Earth-Simulator/ 5120	**35860.00** 40960.00	**Earth Simulator Center** Japan/2002
2	**Hewlett-Packard** ASCI Q - AlphaServer SC ES45/1.25 GHz/ 4096	**7727.00** 10240.00	**Los Alamos National Laboratory** USA/2002
3	**Hewlett-Packard** ASCI Q - AlphaServer SC ES45/1.25 GHz/ 4096	**7727.00** 10240.00	**Los Alamos National Laboratory** USA/2002
4	**IBM** ASCI White, SP Power3 375 MHz/ 8192	**7226.00** 12288.00	**Lawrence Livermore National Laboratory** USA/2000
5	**Linux NetworX** MCR Linux Cluster Xeon 2.4 GHz - Quadrics/ 2304	**5694.00** 11060.00	**Lawrence Livermore National Laboratory** USA/2002

Fig. 1　Top500 (Nov. 2002)

について温度・流れの向き・速度などの物理量を計算する．この際に格子を細かくすればするほど精度の高いシミュレーションが可能である．

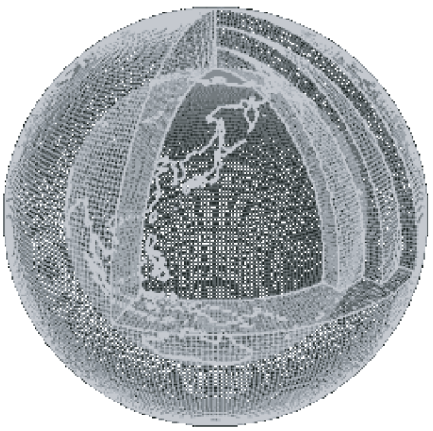

Fig. 2　計算機シミュレータの格子イメージ

地球シミュレータによる気候変動シミュレーションなどでは，地球全体の現象を一手に計算するため，その計算負荷は莫大なものになる．このように，計算に莫大な時間が掛かり，実時間内に解が得られない場合，これは逐次処理では解決不可能な問題となる．これを解決するのが 35TFLOPS[2] という驚異的な計算力をもつ地球シ

[1]大気・海流の変動をシミュレートするもの

[2]FLOPS とは浮動小数点演算に関する計算速度の単位．1TFLOPS

ミュレータである．地球シミュレータは，8 プロセッサからなるスーパーコンピュータを，高速のネットワークで 640 台つないだ並列計算機で，総計 5120 プロセッサで構成されている．

3　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．今や 1 つの CPU の高速化による計算機の高速化は限界にきている．クロック周波数が 1GHz の場合，周期は 1ns となり，電気信号はその間に 30cm しか進まない．集積回路の大きさや配線の基盤の寸法などを考えても，これ以上クロック周波数を飛躍的に上げることは原理的に困難である．

また，CPU の半導体材料にスイッチング速度が 1ns より速いものを使用すれば，まだ CPU の高速化の可能性は残されているが，そのような材料についても極めて限られてくる．よってこの面から考えても，CPU の高速化は限界にきているといえる．

さらにたとえ CPU が高速化されたとしても，メモリからのデータ転送がそれに追いつかなければ意味がない．よってメモリのアクセス速度向上も困難になってきた現状では，単独の CPU の高速化に多くは望めない．

このような理由から，計算機の高速化は並列化の方向へと移っていき，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が超並列計算機となるのも必然的な流れである．地球シミュレータについても，Fig.3 に示すように，世界中で同様の計画が実施されてきているが，1990 年のデータと 2002 年の地球シミュレータのデータを比べると，計算速度が 10,000 倍になっている．さらに 90 年代に入って計算機の数が急激に増えてきていることも分かるだろう．これは並列処理の研究が進んで，計算機の性能が大幅に向上し，実用的なシミュレーションが可能になって来たからである．

4　並列処理の効率

並列処理においてもっとも重要な観点は，複数のプロセッサを効率的に動作させることである．一般に n 個のプロセッサを利用しても，実際の速度は n 倍にはならない．これには分割された仕事の一つ一つが同じ負荷でないため，ある計算機が他の計算機の仕事の終了を待たないといけない状況が起こったり，計算機間の通信の遅延なども起こるためである．

並列処理を行うためには，まず対象問題・アルゴリズム・ソフトウェア・ハードウェアの各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

なら，1 秒間に浮動小数点演算を 1 兆回行える．

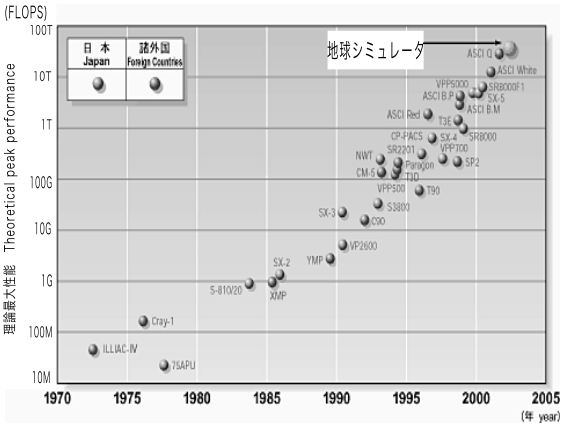

Fig. 3　世界の中の地球シミュレータ計画

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．

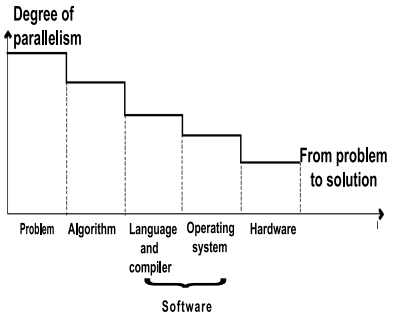

Fig. 4　各レベルにおける並列性

並列処理は，並列化効率と使用するプロセッサ数の数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，次の式 (1) で表される．すなわち，p 個のプロセッサを用いたときの並列化効率は，プロセッサ 1 個で逐次処理を行った場合の処理時間 T_1 を，並行処理したときの処理時間 T_p と p との積で除したものである．

$$E_p = \frac{T_1}{pT_p} \tag{1}$$

E_p：並列化効率

T_1：プロセッサ 1 個で逐次処理を行った場合の処理時間

T_p：p 個のプロセッサで並列処理したときの処理時間

p：プロセッサ数

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のために，通常 $E_p < 1 (T_1/T_p < p)$ となる．その主な要因を次に示す．

1. 並列化出来ない部分の存在
2. CPU のオーバーヘッド
3. 通信の遅延（レイテンシ）
4. ロードバランス

4.1 並列できない部分の存在

この割合が多いと，並列処理の効率は大幅に下がってしまう．「Amdahl の法則」はこれを数学的に示している．Amdahl の法則とは，全体の処理速度を並列処理の部分とそうでない部分の比率の関数として表したものである．s $(0 < s < 1)$ を全処理中逐次処理しなければならない部分の割合とし，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．

$$R(s) = \frac{1}{s + \frac{1-s}{p}} \tag{2}$$

$R(s)$：総合的な速度向上率

$s(0 < s < 1)$：全処理中逐次処理しなければならない部分の割合

p：プロセッサ数

これを，グラフにすると Fig.5 のようになる．横軸がプロセッサ数であり，縦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，$s = 0.5$（つまり逐次処理の部分が全体の処理のうちの 50% を占める場合）では 100 台のプロセッサを使用したとしても，速度向上はわずか 2 倍に及ばない．

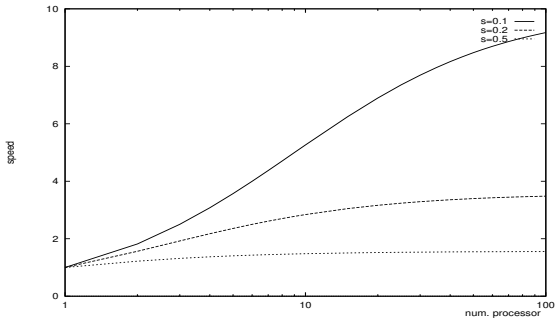

Fig. 5　Amdahl の法則

また，逐次処理部分の割合がたとえ 5% と少なくても（並列処理部分の割合が 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．一般に並列処理を行う場合には，並列処理可能な部分の割合（$= 1 - s$）が 0.98 (98%) 以上ないと，その有効性が出ないとされる．

4.2 CPU のオーバーヘッド

（Fig.6）に示すように，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．

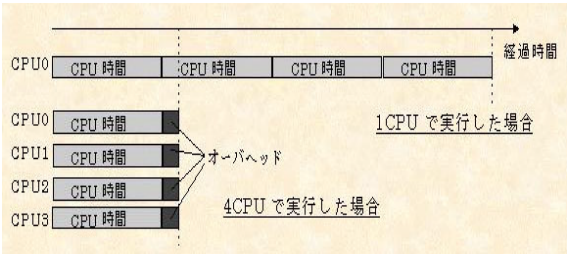

Fig. 6　並列処理における CPU 時間のオーバヘッド

4.3 通信の遅延

これも並列効率を下げる大きな要因である．プロセッサ間通信の速度が遅く，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．

4.4 ロードバランス

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，できるだけ計算負荷を均等に分けなければならない[3]．

各プロセッサに与えられる負荷が不均一であると，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，単純に考えると負荷が 2 倍異なれば計算時間が 2 倍異なり，負荷が小さいプロセッサは，同期をとるためにまだ処理中のプロセッサを待つことになる．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，全体の速度は一番処理の遅いプロセッサの速度に律速されてしまう．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

[3]各プロセッサの計算速度が均一でない場合は，それぞれの能力にあった計算負荷を分けてやる必要がある．クラスタなどではプロセッサの計算速度は均一な場合が多いが，Grid の場合はまず各プロセッサの計算速度はバラバラになるので，この場合は計算負荷の分割が難しくなる．

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．

5　並列処理の問題点

4 章でも述べたように，並列処理では，仕事の分担の仕方が並列化効率に大きく影響する．よってこのことは並列化を行う場合に必ず問題となる．この章では並列処理の問題点についてまとめる．

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．

まずは 1 について考えてみよう．個々の問題にはその問題独自の性質が存在し，各問題の間に一般性を見つけ出すことは難しい．しかし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，計算機での仕事の分担では，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[4]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 ≪ 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[5]や，デッドロック[6]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．

仕事の割り振りの仕方が並列化効率を左右する例を挙げよう．Fig.7 の様に仕事の順序が保たれていれば，各プロセッサは各自の仕事をタイムロス無く処理できるが，Fig.8 の様に仕事の順序が破壊されてしまった場合は，プロセッサ 2 はプロセッサ 1 が仕事 1 を終えるまで仕事 2 を開始できない．このため全体の効率は下がってしまう．

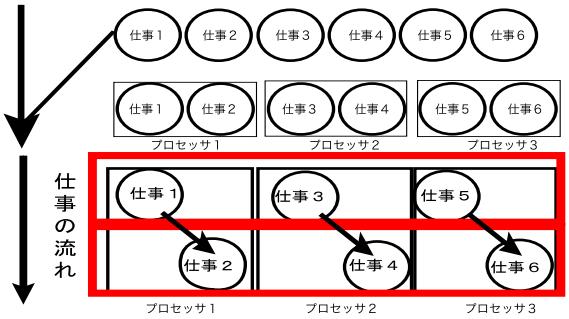

Fig. 7　仕事の割り振り（成功例）

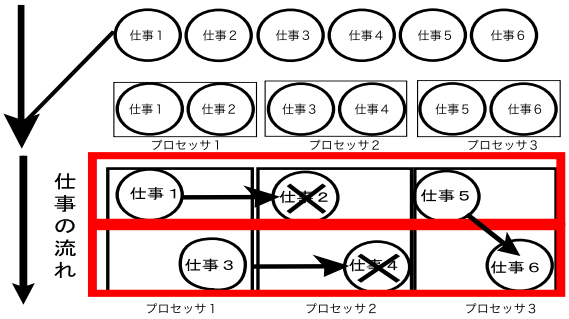

Fig. 8　仕事の割り振り（失敗例）

6　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．

6.1　GA

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ような遺伝的操作が，はるかに短時間に行えるようになる．さらに，GA は並列処理が可能な操作が多いため，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，大まかに，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合度とは独立して計算可能である．この点に着目して，Fig. 9 のように，適合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，適合

[4]オペレーティングシステム（OS）の処理の単位
[5]ある命令の入力が直前の命令の結果に依存すること
[6]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

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，操作全体で見ると全く高速化されないどころか，逐次処理のほうが早いこともある．

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．

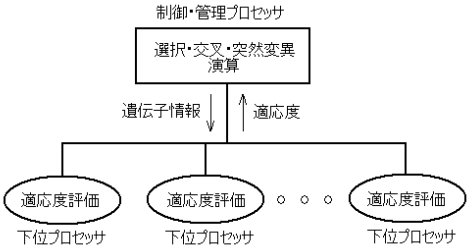

Fig. 9　GA の単純並列アプローチ

6.2　行列計算

行列計算では並列計算できる部分が多いため，効果的に並列処理を行うことができる．例えば，2 つの行列 A, B の掛け算について考える．$A \times B$ の場合，それぞれのプロセッサに A の行列と B の列を送信する．例えば，c というプロセッサに A の行列と B の第一列，d というプロセッサに A の行列と B の第二列となるように送信する．それぞれの計算は影響を及ぼさないため，並列計算可能である．最後にそれらの計算結果をまとめることで行列の計算が終了する．

7　ソフトウェア

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最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．

7.1　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．どのプロセスも他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取りをしたい場合は，Fig. 10 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．

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．

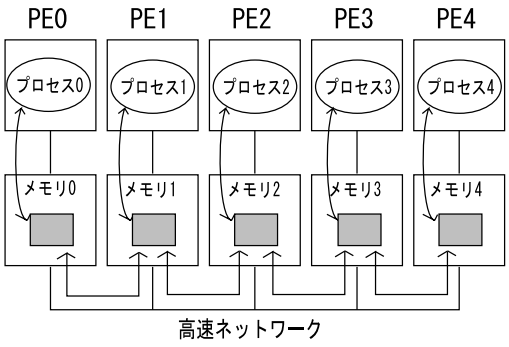

Fig. 10　プロセスの構造

7.2　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出来る部分が存在する．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. 11 のように同じメモリ空間を共用するため，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，パフォーマンスが良い．

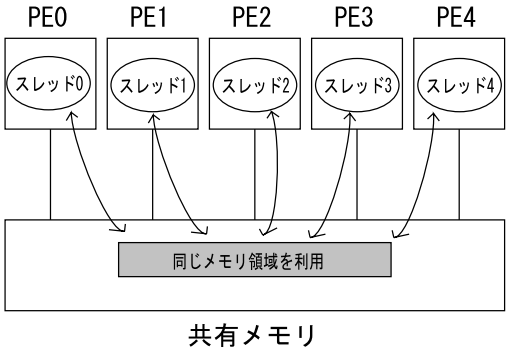

Fig. 11　スレッドの構造

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挙げられる．例えば，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無いように，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．

7.3　メッセージ通信ライブラリ

2 つのプロセスの間でのメッセージのやり取りは，比較的分かりやすい概念である．メッセージ通信は，あるプロセスのメモリ空間から，もう一方のプロセスのメモリ空間へのデータの転送と見ることができる．もちろん一般には，メッセージをやり取りするのは 2 つのプロセスとは限らず，もっと多くのプロセスが関与し得る．

分散メモリ型の並列計算機では，メッセージ通信によるプログラミングが最もハードウェアの構成に即したプ

ログラミングであり，最高の性能が期待できる．また，共有メモリ型の並列計算機でも，メッセージ通信で書いたプログラムを実行するのは容易である．

ここでは最も主要なメッセージ通信ライブラリである MPI(Message Passing Interface) と PVM(Parallel Virtual Machine) について述べる．

7.3.1 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メッセージ通信 API 仕様である．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．例えば，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，その実装がまった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．

MPI の実装ライブラリとしては，フリー，ベンダ問わず数多く存在する．その中でも代表的なフリーのライブラリとしては，LAM と MPICH が挙げられる．

7.3.2 PVM

MPI はインターフェイスの規定であるのに対して，PVM は実装パッケージそのものである．PVM は，TCP/IP ネットワークで接続された何台ものコンピュータを仮想的に 1 台のマシンととらえて，並列プログラムを走らせることのできるメッセージ・パッシングの環境とライブラリを提供する．

8 通信ネットワーク

分散メモリ型の並列計算機の一つである PC クラスタの通信ネットワークには，イーサネットや Myrinet という規格がよく用いられる．PC 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全体の処理が大きく向上する.

8.1 イーサネット (Ethernet)

イーサネットは，OSI 参照モデル (Fig. 12) 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．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．

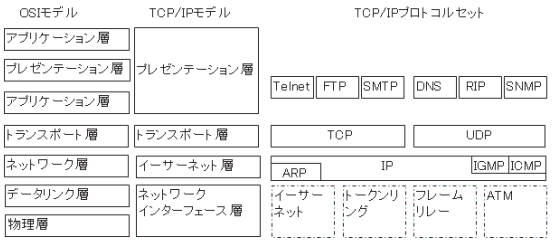

Fig. 12　OSI 参照モデル

現在では，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る 1Gbps のネットワーク帯域幅をサポートすることができるギガビット・イーサネットという規格も使われはじめている．

8.2 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 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
- 低レイテンシ，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 スイッチでは平均故障間隔 (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，ハイアベイラビリティを実現できる．

9　ハードウェア

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．

9.1　共有メモリ型

複数のプロセッサがメモリバス／スイッチ経由で，主記憶に接続される形態である (Fig. 13)．このアーキテクチャを有するシステムのことを SMP(Symmetric MultiProcessor) 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．

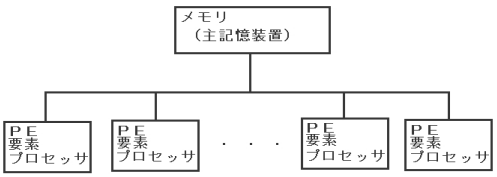

Fig. 13　共有メモリ型の構造

この形態の特徴としては，複数のプロセスによる並列化のみならず，スレッド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，コンパイラによる自動並列化が可能．
- 分散したメモリ間での通信が無いため，プロセッサ数が一桁程度の範囲では実行性能の理論最大性能に対する落ち込みが少ない．

9.2　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. 14 参照)．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を組みやすいが，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．

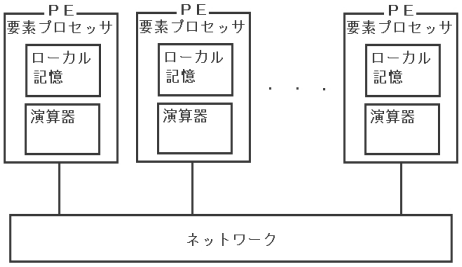

Fig. 14　分散メモリ型の構造

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，ロードバランスを考慮する必要がある．

参考文献

1) 地球シミュレータの概要，http://www.es.jamstec.go.jp/esc/jp/outline.html/.

2) 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/.

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

4) TOP500 List for November 2002 ，http://www.top500.org/list/2002/11/.

5) 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.

6) David A.Patterson & John L.Hennessy 著，成田光彰 訳：コンピュータの構成と設計¡上¿，日経 BP 社，1999.

7) David A.Patterson & John L.Hennessy 著，成田光彰 訳：コンピュータの構成と設計¡下¿，日経 BP 社，1999.

8) 超並列計算研究会 編：PC クラスタ超入門，超並列計算研究会，2000.