第1回並列ゼミ

指導者 佐野正樹
チーフ 下坂久司
サブチーフ 田村隆一 松山靖彦

第I部
並列処理概論

1　並列処理とは？

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. 簡単に言えば, このゼミで学ぶ並列処理とは,「一つの目的を持った処理を，いくつかの処理に分割し, これらを同時に行うこと」である. 各処理は, 複数のマシンで実行される. この複数のマシンが同時期に動いていれば, 並列処理をしているといえる.

一般に並列処理の目的は主に，次の 3 点である.

- **処理時間の短縮**…1 つの問題に対する処理手順を並列に実行することにより処理が終了するまでの時間を短縮する．これにより，逐次処理では一ヶ月かかっていた処理が半月ですむ事も可能となる.
- **アルゴリズムの簡明化**…本質的に並列性をもっている問題も，これまでは逐次型アルゴリズムで解決してきたが，そのときの表現の複雑さ・困難性を並列処理により解消する．例として生協購買部のレジの問題を解決するプログラムを考えたとき, 逐次型ではなく並列的に考えた方が表現が簡単である.
- **信頼性と稼働性の向上 (計算速度向上ではない)**…信頼性向上のためのシステムである. 多重化とも呼ばれ, システムの一方の誤動作を検出し, もう一方がそれを訂正することでシステムの信頼性を向上することができる.

図 1: 並列処理の例

2　並列処理の問題点

2.1　並列処理における一般的問題点

並列処理には, 上で挙げたような利点もあれば, これから述べるような問題点も当然存在している. 問題点とは以下の 2 点が存在する.

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. そのため, 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, そのアーキテクチャのバックグラウンドまで考える必要が生まれる.「通信時間 ≪ 計算時間」という関係が, 少なくとも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.

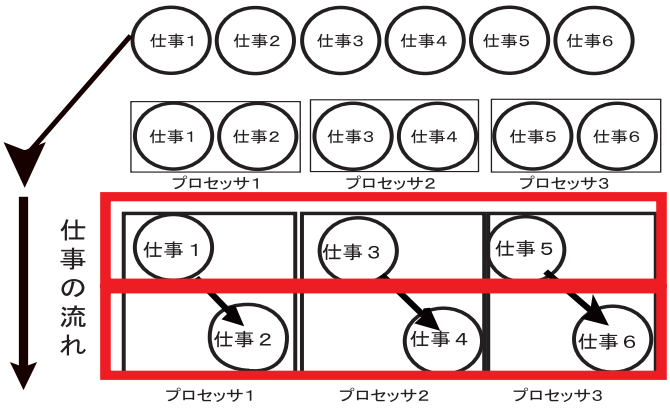

図 2: 仕事の割り振り

2.2 並列処理における他の諸問題

並列処理のパフォーマンスに関して, 根本的に性能を左右するのが, アムダールの法則である. アムダールの法則を説明する前に並列化効率について述べる.

並列化効率：まとまりのある一つの仕事を n 分割して，n 個のプロセッサで並列に実行した場合を考える．この仕事を逐次処理した場合の実行時間を T_1 とし，n 個のプロセッサで並列処理した場合の実行時間を T_n とすると，$S_p = T_1/T_n$ で定義される S_p を速度向上比，あるいは実行プロセッサ数と呼ぶ．また，速度向上比 S_p とプロセッサ数 n の比 $e = S_p/n$ で定義される e を並列化効率と呼ぶ．理想的には $e = 1(S_p = n)$ 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 (CPU 時間のオーバーヘッドなど：図.3) のために通常 $e < 1(S_p < n)$ 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.

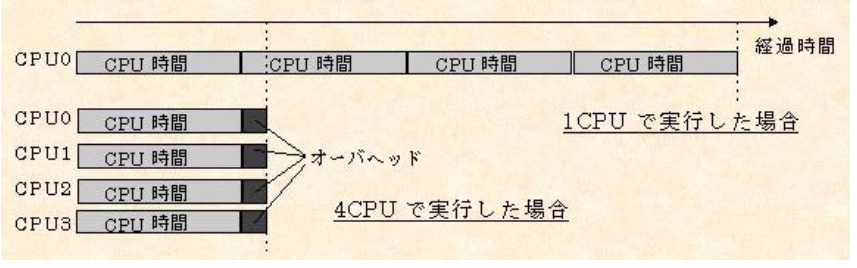

図 3: 並列処理に置ける CPU 時間のオーバヘッド

アムダールの法則：逐次処理部分の割合が全体の s $(0 < s < 1)$ 存在すれば n 台のプロセッサを用いて得られる速

[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

度向上は，$\frac{1}{s+\frac{1-s}{n}}$ になる (図.4)．つまり並列処理による処理を妨げる大きな要因の一つとして, プログラム全体の内並列に実行できる部分と, できない部分の割合が並列処理の効率を左右するということである. 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並列処理は実行されるべきである. 以下の図.4 は, アムダールの法則を表す数式である.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. パラメータ s としては, プログラム全体における逐次処理される部分の割合である. 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」.

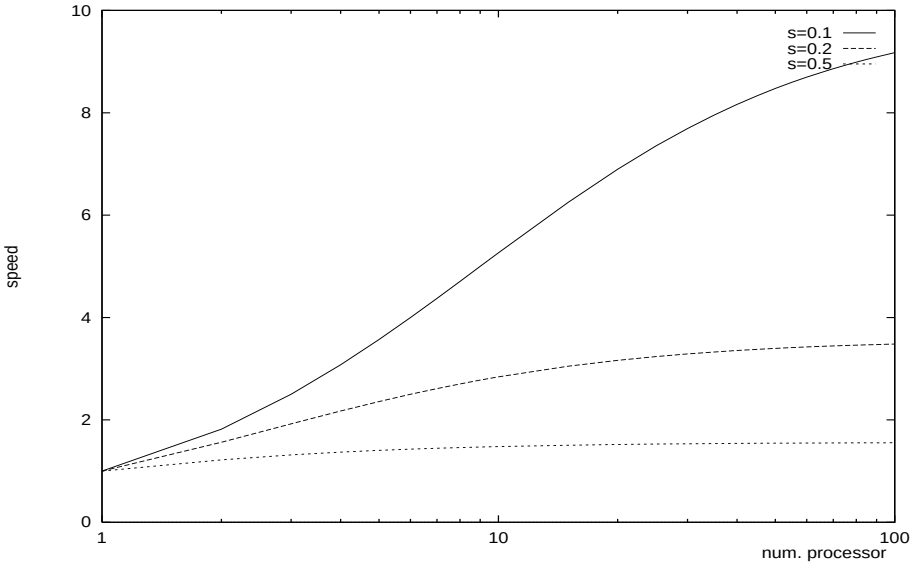

図 4: アムダールの法則

3 並列コンピュータのアーキテクチャ

並列コンピュータのアーキテクチャには, 大きく分けて分散メモリ型と共有メモリ型がある. また専用アーキテクチャと汎用アーキテクチャが存在する. ここではそれらの特長について述べる.

3.1 並列コンピュータの種類

分散メモリ型並列コンピュータ：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 (図.5)．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.

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（ヘテロな環境）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.

共有メモリ型並列コンピュータ：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 (図.6)．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.

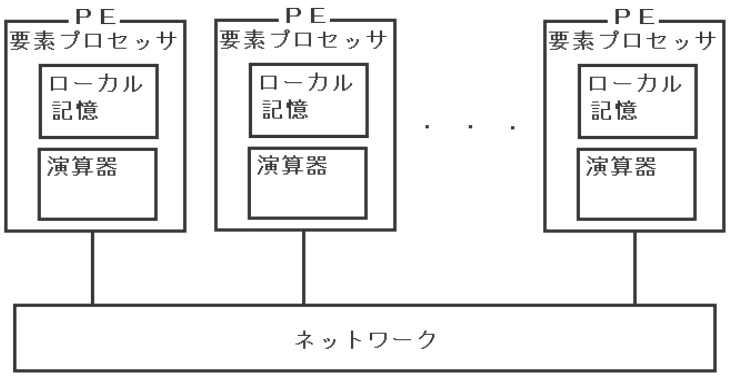

図 5: 分散メモリ型

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.

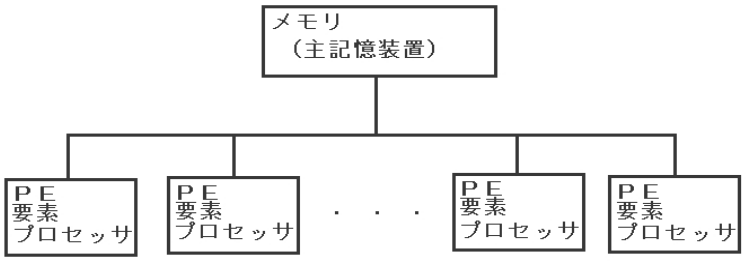

図 6: 共有メモリ型

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.

NUMA：NUMA(Non-Uniform Memory Access model : 非均等メモリアクセス) は SMP から派生したもので，分散メモリマシンの複数のメモリを一つの共有メモリとしてハードウェアでエミュレートしたものである．アクセス遅延, 転送容量に差のある共有メモリを持つ (個々の CPU とメモリ同士 (ローカルメモリ) のアクセスは高速だが, 他の CPU のメモリ (非ローカルメモリ) へのアクセスは遅い). 従来の SMP では CPU の個数の増加やハイパフォーマンス化に伴い，メモリアクセスを制御することが非常に難しくなってきた. また構造上の問題からメモリの遅延を抑えながら CPU やメモリを増やすことが難しくなってきた. そこで，分散メモリ共有という形を採ることによって, 高いスケーラビリティと可用性を得ることができるのである.SMP のプログラムモデルをそのまま利用することができるのも大きな利点である.

3.2 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, 専用のアーキテクチャを備えた専用マシンと, 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 種類のシステムが存在する.

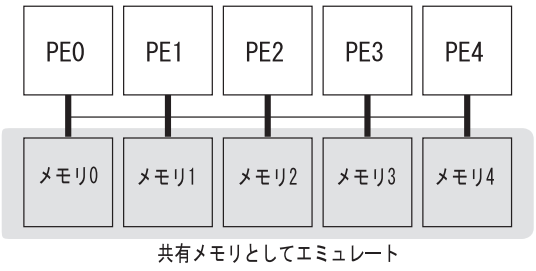

図 7: NUMA

3.2.1 専用アーキテクチャ

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. これらのシステムは, 専用の部品で組まれているため, 価格が非常に高いことが大きなネックとなっている.

3.2.2 クラスタ

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, 三木研究室で使用しているクラスタは, タスク分割による並列処理である. 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.

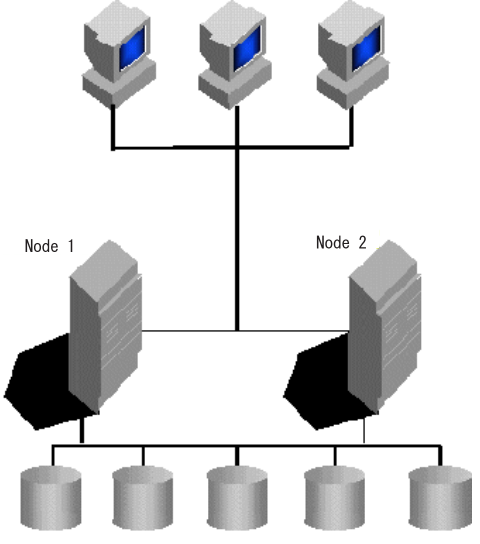

図 8: クラスタの例

3.3 並列コンピュータの分類

SISD(Single Instruction Stream Multiple Data Stream):一つの命令が一つのデータを対象とする方式である．通常の逐次型コンピュータがこれである．

SIMD(Single Instruction Stream Multiple Data Stream):複数の演算器やプロセッサを用いて，一つの命令を複数の異なるデータに対して実行する方式である．SIMD では演算装置が多数設置されており，それらが一つの制御部から発せられた演算命令を同時に実行する．各演算装置はパイプライン方式で一つの命令を実行することで高速化を実現している. 通常のスーパーコンピュータは SIMD 型である.SIMD の特徴を以下にあげる.

1. 定期的な並列処理の高速化を達成でき，ベクトルや配列といった集合に対する演算を必要とする分野に適している．
2. 各演算装置には順序制御装置が必要でなく，小型化をはかることができ，VLSI 化に適している．
3. 制御装置は 1 台であり，データによって演算内容が各演算装置ごとに異なるような場合には制御が煩雑になる．

MISD(Multiple Instruction Stream Single Data Stream):複数の命令がと一つのデータを対象とする方式である．分類学上は存在するが実用的なコンピュータとしてはまだない.

MIMD(Multiple Instruction Stream Multiple Data Stream):それぞれの異なった処理を行う複数の命令を複数のプロセッサ上で，それぞれ異なった複数の命令に対して実行する方式である．アレーコンピュータ[4]がこの分類に入る．SIMD 方式のコンピュータと比較して，並列処理の柔軟性が高く適用範囲が広いのが特長である．

4 並列プログラミング概要

4.1 プロセスとスレッド

プログラムの実行単位をプロセスと言う．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は，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．プロセス内のメモリ空間が保護されていることは安全ではあるが，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.

そこで，スレッドという実行単位が登場する．プロセス内には複数のスレッドを生成することができる．あるプロセス内の任意のスレッドでは，そのプロセスの持つメモリ空間に自由にアクセスすることができる．つまり，1 つのプロセス内に限定して言えば，スレッド間でデータを共有できることになる．これはスレッドの便利な点であるが，危険な点でもある．同時にデータの書き込みが行われるという事態になった場合の動作は保証できない．スレッドプログラムを作成する場合，プログラマはデータアクセスを制御しなくてはならない．並列プログラムとはプロセスやスレッドを複数生成して，それぞれプロセッサに割り当てるプログラムであると言える．特にプロセスは分散メモリ環境，スレッドは共有メモリ環境に適していると言える.

4.2 マルチプロセスプログラミング

複数のプロセスを発生させて, ある目的を達成させるようにしたものをマルチプロセスという. 異なるプロセス同士は直接, プロセスがもつメモリ領域にはアクセスできない. このため, 異なるプロセスのメモリ領域にアクセスしたい場合は, 外部的・内部的な通信を行わなくてはならない.

分散メモリ型やクラスタにおいて, 複数プロセスでの処理・通信を行うには MPI(Message Passing Interfase) や PVM(Parallel Virtual Machine) といったライブラリを用いることができる。その他にも TCP/IP ベースのプログラミングも広い範疇ではこれに属しているといえる．PVM に関しても,MPI に関してもそのローレベルでの処理は, インターネットの標準プロトコルである TCP/IP[5]である. つまりは, プログラマーが通信に関してローレベルで操作する部分を隠蔽する役目をこれら並列プログラム環境が担っているといえる.

並列・分散環境におけるマルチプロセスプログラミングにおいて, オーバヘッドの要因となるのは, 大きく分けて負荷バランスと通信のオーバーヘッドである. 負荷バランスに関しては, 並列プログラムで解決するというよりは, アルゴリ

[4]パイプラインを複数設けたコンピュータ
[5]Transmission Control Protocol/Internet Protocol

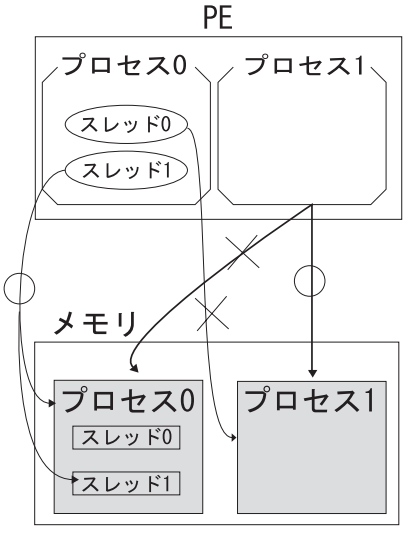

図 9: スレッド・プロセスとメモリアクセス

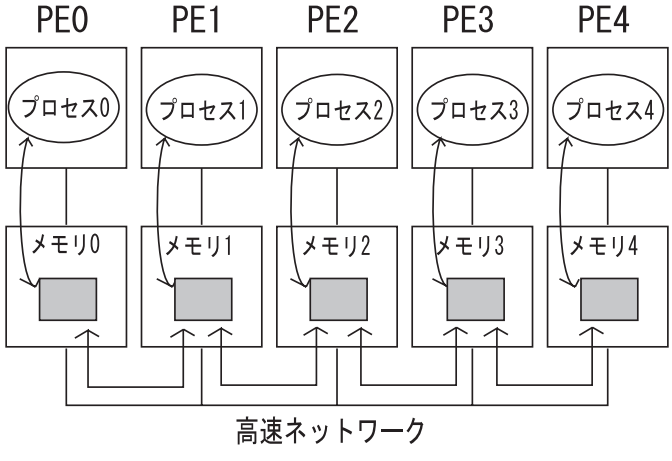

図 10: プロセスと分散メモリ型並列コンピュータ

ズムなどで解決する問題である. 通信は並列プロセスが相互に影響を与えながら計算を進める上で必要なものであるが, この通信は単にデータの移動であり, 純粋なオーバヘッドとして現れる. この通信のオーバヘッドを減らすためには, もちろんアルゴリズムを修正・改良し, 通信の削減を行うのが一番効果的であるが, プログラミングのテクニックとして

- 不要なオーバヘッド部分の削除
- メッセージベクトリゼーション

を行うことがある. メッセージベクトリゼーションとは (同じあて先の) メッセージを幾つかまとめて一度に送る手法である. 通信にはレイテンシ (遅延) が必ず起こるので, なるべくたくさんのメッセージを一度に送った方が, この通信のオーバヘッドを削減することができる.

4.3 マルチスレッドプログラミング

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位が存在する. 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, 同じメモリ空間を共有するため. 各アーキテクチャに対して, 以下のような利点が存在する.

- マルチスレッドそのものの利点として, 複数のプロセスを立ち上げるには, 多くのリソースを消費することになる. それに対しスレッドはあくまで, プロセスとしては一つであり, リソースは必要に応じてスレッド起動時に与えられる. つまり, スレッドは, プロセスよりも “軽い” といえる. 軽いということは, 同じ目的のシステムを構築する際にも, マルチスレッド化で実現した方が, 圧倒的にパフォーマンスは良いということになる.
- マルチスレッドが一般的利用される場面としては, クライアントサーバシステムであり. 例えば, 一つのプログラムに対して同時に複数の要求があった際に, 個々の要求に対して, 一つのスレッドを当てることで, スループットをあげることができる.
- マルチプロセッサシステム (I/O 動作と計算処理が集中するシステム) のパフォーマンスが改善される．各プロセッサに対して, スレッドを独立して走らせることで, 並列に処理できる.
- ユニプロセッサシステム (I/O 動作が集中するシステム) のパフォーマンスが改善される．スレッドは, プロセスと異なりリソースをスレッドの実行中であっても, 別のスレッドに渡すことができる. ただし, これには, メモリに関してはプログラマが管理する必要がある.

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ないように, 片方のデータの書き込み (読み込み) を制限する必要がある.

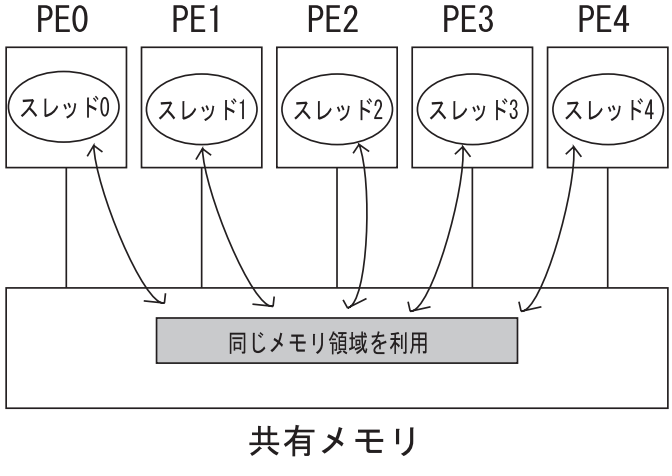

図 11: スレッドと SMP

アーキテクチャには, それぞれ特有のソフトウェアの実装が存在する． 専用ハードウェアの多くは, 専用の C や Fortran コンパイラによって自己のアーキテクチャに最適なプログラムを生成するシステムが大半である. これらについては, 機種既存が高く, 一概にはいえないが,SMP 型ならばマルチスレッドによるアプローチが多用される. 複数スレッドでメモリを共有するには，Pthread(POSIX thread) などのライブラリや OpenMP などのコンパイラを使用することができる.

5　並列処理における重要単語

5.1　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.12)．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．

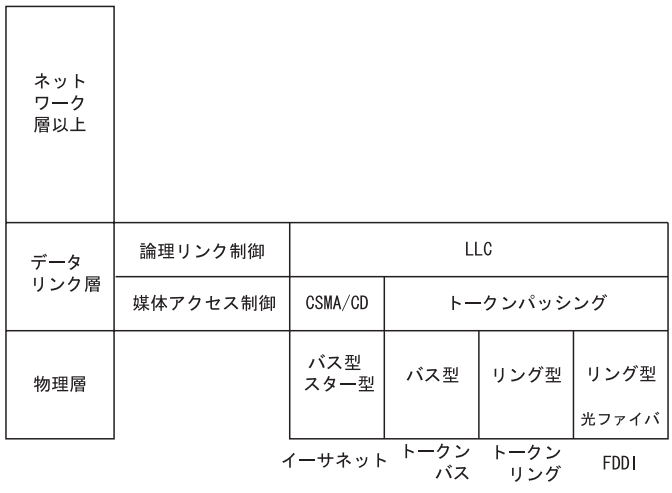

図 12: イーサネットと OSI 参照モデル

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.13)．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.14).

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.

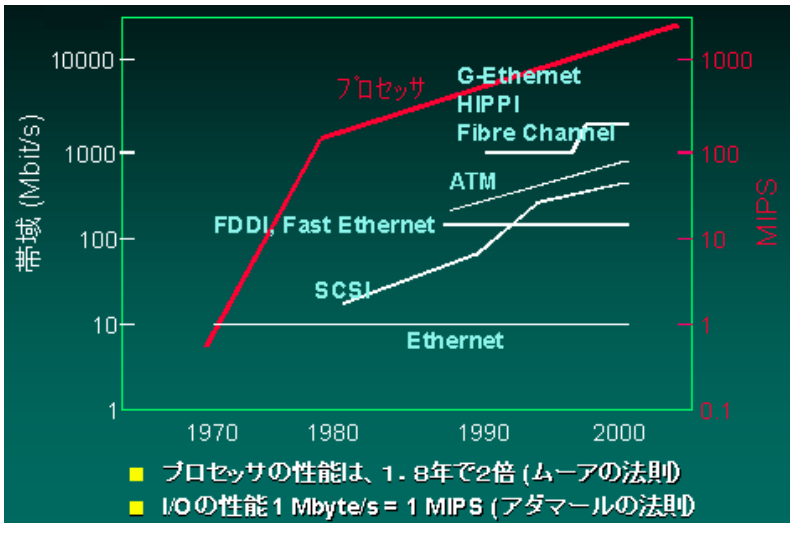

図 13: ネットワークの進歩

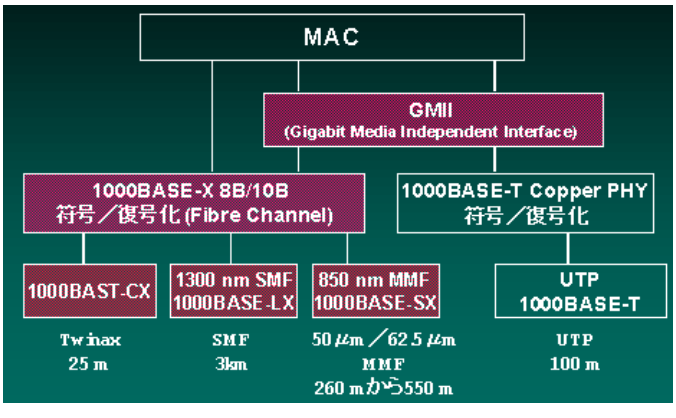

図 14: 物理層でのインタフェース

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．

5.2 粒度

並列処理を行うためにプログラムを各プロセッサへの割り当てる単位であるタスクに分けなくてはならない．そのタスクの大きさのことを粒度という．タスクの大きさと比重にかけられる物は, その通信コストであり, タスクは分ければ分けるほど良いというものではない. 粒度をどのように設定するかという問題は，並列処理効果を向上させる上で非常に重要な項目である．中には, ある時期を区切って大きな単位のタスクを集団通信させることで, パフォーマンスがあがることがある. 通信量に対して, 実際の内部の処理量が多いことを粒度が大きい, 逆に少ないことを粒度が小さいと言う. 一般に粒度が大きいと通信のオーバヘッドが少なくなる．

参考文献

[1] 湯淺太一，安村通晃，中田登志之著『はじめての並列プログラミング』(共立出版株式会社，1999)

[2] 小泉修著『図解でわかる LAN のすべて』(日本実業出版社，2000，第 15 版)

[3] 並列処理の目的：http://nsgw3.mse.kanagawa-it.ac.jp/pararell/para3.html

[4] 並列化によるオーバーヘッド：http://www.cc.tohoku.ac.jp/refer/super/PARA/auto-para2.html

[5] 並列計算機のモデル：http://www.tuat.ac.jp/ kiwasawa/para/Parallel.html

[6] 共有・分散メモリ型の特徴：http://www.alpha-act.co.jp/hpc/hp_4.htm

[7] 分散共有メモリ：http://www-hiraki.is.s.u-tokyo.ac.jp/members/tanaka/Ocha5/Intro/introduction.html

[8] マルチクラスタの特徴：http://www.compaq.co.jp/service/server_faq/faq03/1998100084.html

[9] クラスタのメリット：http://www.harapan.co.jp/Tech_Lib/cluster/sld002.htm

[10] ギガビットイーサネット：http://www.intel.com/jp/comm-net/network/overview/gigabit/gigabit1.htm

[11] スレッドとは：http://www.inprise.co.jp/tips/delphi/dh004/thread_1.html

[12] 物理層のインタフェース：http://www.intap.or.jp/INTAP/information/seminar/g-ether/sld016.htm

[13] ネットワークの進歩：http://www.intap.or.jp/INTAP/information/seminar/g-ether/sld003.htm

[14] SMP：http://www.ibm.co.jp/as400/0696.html

[15] マルチプロセッサの構成：http://hp.vector.co.jp/authors/VA004443/pcat/SMP.html

[16] マルチスレッドの例：http://www.sw.nec.co.jp/middle/WebOTX_S/overview2.html

[17] MYRINET：http://q.sse.co.jp/comid/engin/myrinet/no_frame/index.html