第1回並列ゼミ

指導者 川崎高志 小掠真貴
チーフ 長谷佳明
サブチーフ 水田伯典 羽山徹彩

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解が得られ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. 例として再帰表現の多用はプログラムの難解さを高めるが, これらをできうる限り展開し並列に実行することで処理効率を上げるのみならず, 読みやすいプログラムを作成することができる.SMP 向け並列処理コンパイラ HPF[1]などでは, コンパイラレベルでこの処理を実行させることができる.
- **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.

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, 通信のレイテンシを考えた分割法など, 仕事そのもののタスク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や, デッドロック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.

[1] High Performance Fortran

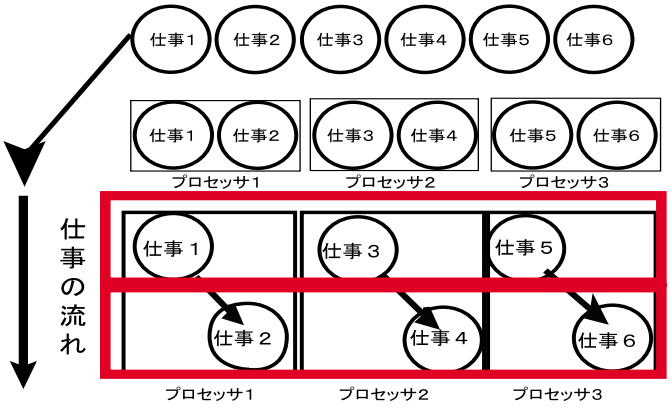

図 1: 仕事の割り振り

2.2 並列処理における他の諸問題

そして, 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, 以下のアムダールの法則である.

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度向上は，$\dfrac{1}{s + \frac{1-s}{n}}$ になる (Fig. 2)[2]．並列処理による処理を妨げる要因の一つ. つまり, 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こそが, 並列処理の効率を左右するということ. 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 Fig.2 は, アムダールの法則を表す数式の一例である.X 軸が, プロセッサ数であり,Y 軸が逐次処理した際との性能の向上率である. 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「効率化比で 2 倍に及ばない」.

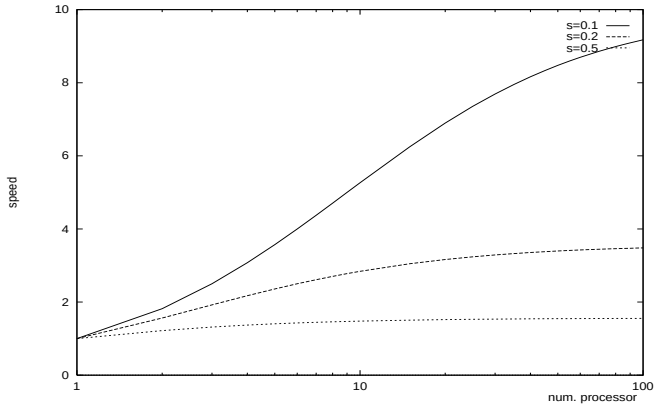

図 2: アムダールの法則

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：Fig. 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.

[2]ただし, 中には予想外に速度向上するいわゆる, 超線形 (super linear) なものも存在する.

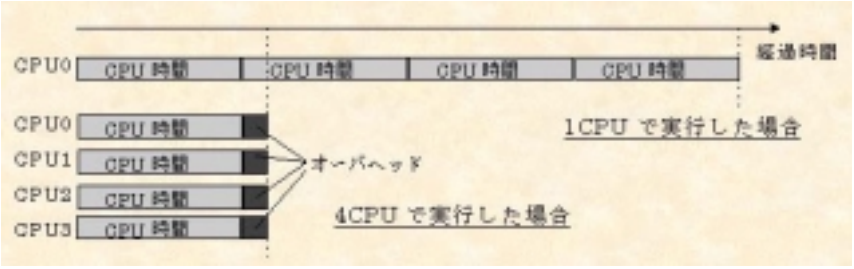

図 3: 並列処理に置ける CPU 時間のオーバヘッド

3 並列化処理を実現する上でのアーキテクチャ

3.1 並列コンピュータとは？

並列コンピュータとは, 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する.

前者は, 専用のバスを備えた SMP マシンが一般的である. 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[3]を用いたシステムもある. これらは, 分散メモリマシンを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. 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.

後者のシステムは, クラスタによって並列コンピュータを構成しようというものである. クラスタは, 複数台のコンピュータを高速のネットワークで繋いだものである. かつては, システムの信頼性向上のための方法として主に用いられていたが, 昨今は, このゼミのタイトルでもある並列処理を実現するための技術として急速に広まりつつある. 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.

3.2 各アーキテクチャの特徴

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 (Fig.4)．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コミュニケート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.

このアーキテクチャに即した並列プログラミング環境としては，PVM[4]および MPI[5]が有名であるが，その他 TCP/IP ベースのプログラミングも広い範疇ではこのアーキテクチャ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[6]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.

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.
- 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.

[3]Non-Uniform Memory Access：不均等メモリーアクセス
[4]Parallel Virtual Machine
[5]Message Passing Interface
[6]Transmission Control Protocol/Internet Protocol

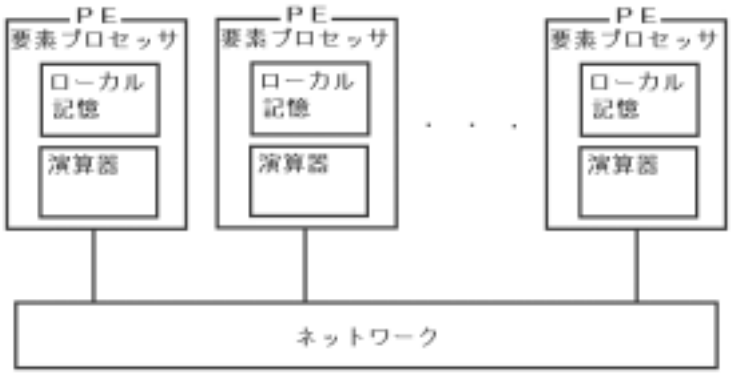

図 4: 分散メモリ型

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 (FIg.5)．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.

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.

図 5: 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.

クラスタ：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.

3.3 ソフトウェアアプローチ

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. そして,

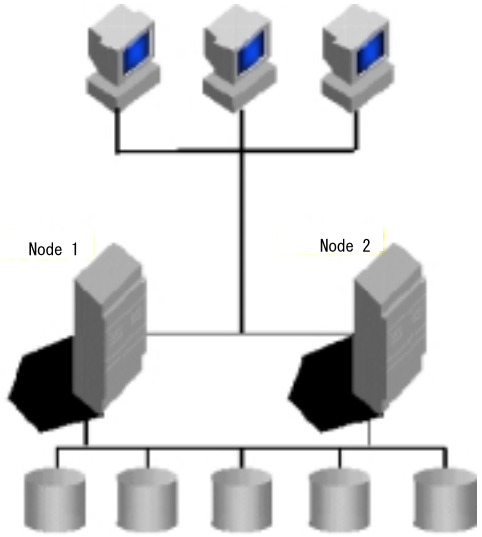

図 6: クラスタの例

分散メモリ型ならば,MPI などの並列ライブラリ規格に準じたシステムが提供される.

並列処理計算を目的としたクラスタにより並列コンピュータを構成する際には, 先の節でも述べた PVM や MPI といったものが使われる.

4 並列処理における重要単語

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 (Fig. 7). 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 (Fig. 8).

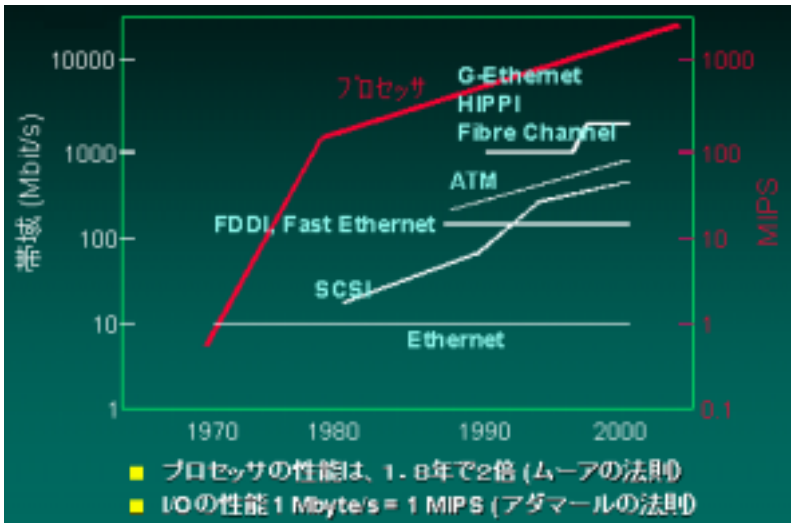

図 7: ネットワークの進歩

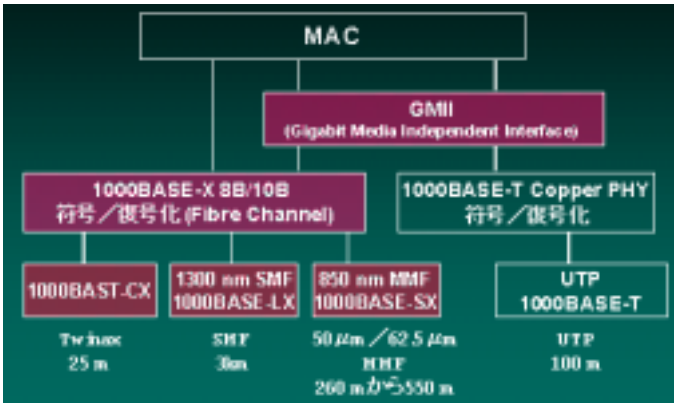

図 8: 物理層でのインタフェース

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.

マルチスレッド：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位が存在する. それをスレッドとして扱うことができる. 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.

Fig. 9 に例を示す．このシステムでは，スレッドをあらかじめ生成しておきそれらをプールしておき，実際にトランザクションの要求がくるとリカバリサーバ内で自動的にそれらのスレッドを利用する仕組みになっている.

マルチプロセス：マルチプロセスとは，同時に複数のプロセッサを使用し，並列処理を行なうことである．並列処理の実行方式として，非対称型多重プロセッシング (ASMP) と対称型多重プロセッシング (SMP) という種類がある. 2 種類のモデルを Fig. 10 に示す．マルチプロセスのポイントは，一度により多くの仕事をさせて，ユーザーのコンピュータを高速化することにある.

ASMP(非対称型多重マルチプロセス) とは，異なるプロセッサは異なる処理を行う機能である．すなわち，OS は必ずどれか特定の CPU で動作し，それ以外のプロセスまたはスレッドは他の CPU で動作する．しばしば，あるプロセッサは別のプロセッサをコントロールするようにデザインされている．ASMP は特殊な作業と大規模なマルチプロセス (プロセッサ 8 個以上) に向いている.

SMP(対称型マルチプロセス) とは，並列処理によりシステム資源を有効に活用し，データベース処理のパフォーマンスを大幅に向上させる機能である．SMP を利用すると，システム内の複数のプロセッサーが一つの仕事（タスク）を分担し，主記憶域を共有・活用する．このテクノロジーは特に大量データの検索処理において効果を発揮する．SMP

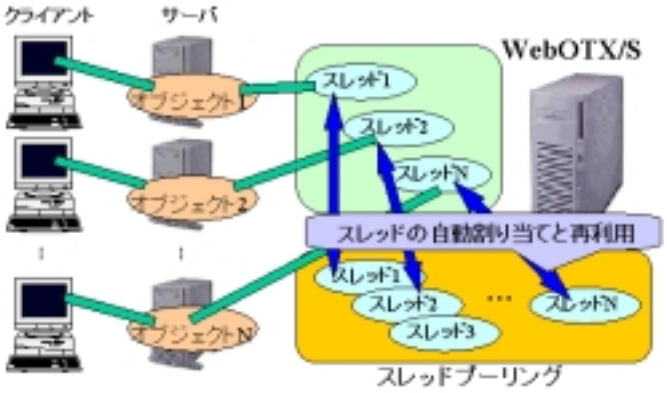

図 9: マルチスレッドを利用するシステムの例

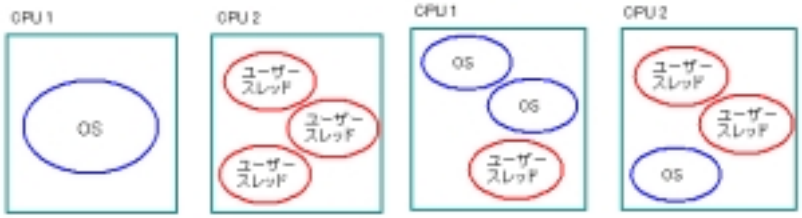

図 10: ASMP(左) と SMP(右) のモデル

はだいたいにおいて，比較的小規模な処理 (プロセッサ 4 個から 8 個) と最も一般的なアプリケーションで優れている．簡単なシステムの例を Fig. 11 に示す．

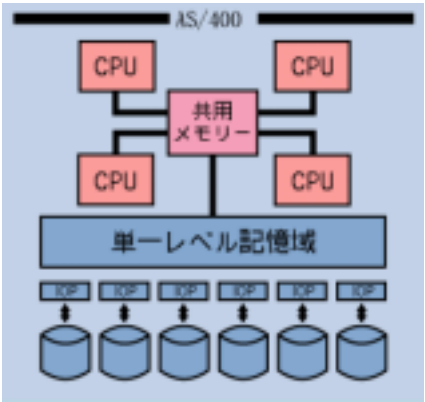

図 11: SMP の例

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

[3] 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

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

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

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

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

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

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

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

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

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

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

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

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

[16] マルチプロセス：http://search.zdnet.co.jp/macweek/9912/14/c_igeek.html