講義 6

並列処理入門

同志社大学 工学部 知識工学科

三木 光範

mmiki@mail.doshisha.ac.jp

■ はじめに

並列処理（Parallel Processing）とは，一般的な意味では「仕事を並列に進めること」である.「並列」だけの意味は「並び連なること」（大辞林）であるが，これでは直列との違いがわからない．一方,「並列処理」は専門用語で,「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）という意味である．並列処理と同じような言葉に **並行処理** がある．複数の CPU を使って複数のプログラムを（真に）同時に実行することを並列（Parallel）実行，一つの CPU で擬似的に実行することを疑似並列（Pseudo-parallel）とか並行（Concurrent）実行とよぶ．また，並行という言葉は，真の並列と疑似並列の両方を総括して示す場合にも使われる [1]．

一方，マルチプロセスとは複数のプログラムが同一のコンピュータを同時に活用できる機能のことである．通常のコンピュータには CPU は一つしかないので，二つ以上のプログラムを同時に実行することはできない．マルチプロセスを提供するコンピュータでは 1/60 秒程度の短い間隔で複数のプログラムを順に実行し，同時に実行されているかのような錯覚をユーザに与えているのである．ただし，こうした環境下でもあるプログラムが入力待ちなどがあるので，システム全体の処理効率が期待できる [1]．

ここでは，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述べる．並列処理の背景には並列計算機やＰＣクラスタなどのハードウエアの発達と並列化コンパイラやプロセッサ間通信のためのドライバなどソフトウエアの発達がある．このため，並列処理を考える場合には,「そもそも仕事を並列に進めるとはどういうことか」という観点と「並列処理のためのハードウエアとソフトウエアはどうなっているのか」という観点が不可欠である．本講習会では特にＰＣクラスタに焦点を当て，こうしたことが初心者にも分かるように構成されている．本稿ではその最初の導入として，並列処理のための概要を述べるとともに,「そもそも仕事を並列に進めるとはどういうことか」という点について考える．

■ 超並列計算機の発達

1997 年 5 月 11 日，ロシア人のチェス世界チャンピオン，ガルリ・カスパロフは IBM 社のコンピュータ ”Deep Blue ”とシリーズ第６戦を行い，２勝１敗３引き分けで Deep Blue が勝った [2]．これは，チェスという「頭を使う」ゲームにおいてコンピュータが人間のチャンピオンを負かしたという画期

Rank	Manufacturer	Computer	Rmax	Installation Site	Country	Year	Area of Installation	# Proc	Rpeak	Nmax	N1/2
1	Intel	ASCI Red	2379.6	Sandia National Labs Albuquerque	USA	1999	Research	9632	3207	362880	75400
2	IBM	ASCI Blue-Pacific SST, IBM SP 604e	2144	Lawrence Livermore National Laboratory Livermore	USA	1999	Research Energy	5808	3868	431344	.
3	SGI	ASCI Blue Mountain	1608	Los Alamos National Laboratory Los Alamos	USA	1998	Research	6144	3072	374400	138000
4	IBM	SP Power3 375 MHz	1417	IBM/Naval Oceanographic Office (NAVOCEANO) Poughkeepsie	USA	2000	Vendor Aerospace	1336	2004	374000	.
5	Hitachi	SR8000-F1/112	1035	Leibniz Rechenzentrum Muenchen	Germany	2000	Academic	112	1344	120000	15160
6	Hitachi	SR8000-F1/100	917.2	High Energy Accelerator Research Organization /KEK Tsukuba	Japan	2000	Research	100	1200	115000	15000
7	Cray Inc.	T3E1200	891.5	Government	USA	1998	Classified	1084	1300.8	259200	26400
8	Cray Inc.	T3E1200	891.5	US Army HPC Research Center at NCS Minneapolis	USA	2000	Research	1084	1300.8	259200	26400
9	Hitachi	SR8000/128	873.6	University of Tokyo Tokyo	Japan	1999	Academic	128	1024	120000	16000
10	Cray Inc.	T3E900	815.1	Government	USA	1997	Classified	1324	1191.6	134400	26880
11	IBM	SP Power3 375 MHz	723.4	Oak Ridge National Laboratory Oak Ridge	USA	2000	Research	704	1056	187000	37500
12	SGI	ORIGIN 2000 250 MHz	690.9	Los Alamos National Laboratory /ACL Los Alamos	USA	1999	Research	2048	1024	229248	80640
13	Cray Inc.	T3E900	675.7	Naval Oceanographic Office (NAVOCEANO) Bay Saint Louis	USA	1999	Research Weather	1084	975.6	.	.
14	Cray Inc.	T3E1200	671.2	CSAR at the University of Manchester Manchester	UK	2000	Academic	812	974.4	.	.
15	Cray Inc.	T3E1200	671.2	Deutscher Wetterdienst Offenbach	Germany	1999	Research Weather	812	974.4	.	.

図 6.1: 2000 年 6 月時点での高性能コンピュータの世界ランキング（15 位まで）[3)]

的な出来事である．この Deep Blue は過去 100 年の序盤戦と，数十億の終盤戦シナリオのデータベースを搭載し，256 個のプロセッサーを持つ超並列スーパーコンピュータであり，１秒間に数億通りの局面を読むことのできる能力を持っており，気の遠くなるような膨大な計算を行ってチャンピオンに対抗した．いわばしらみつぶしに近い力業で勝利した．

このようなコンピュータの力業は近年ますます重要になってきている．科学・工学におけるシミュレーションは取り扱う現象のレベルがミクロのスケールに近づくようになり，できるだけ普遍的で，近似の少ない，原理的な法則を基にコンピュータの力業で複雑な現象を解析するアプローチが注目を集めている．これは明らかにコンピュータの驚異的な発達に依存している．力業が行える驚異的な力が得られるようになったからであり，その驚異的な力が半導体技術の高度化とプロセッサの並列化という二つの技術によって支えられている．

プロセッサ単独の計算速度は次第に限界に近づいてきた．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．材料の面から言っても限界は近い．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．こうして，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．

■ 計算の高速化と並列処理

計算機の高速化の要求には終わりがない．この進歩を可能にする計算機技術が演算パイプライン方式であり，さらにそれを多重にした多重パイプラインであり，こうした方式の計算機は **スーパーコン**

ピュータ と呼ばれる．一方，CPU を極めて多数並列化した計算機が **パラレルコンピュータ** である．

現在の高性能計算機のランキングを図 6.1 に示す．これはドイツのマンハイム大学スーパーコンピュータセンターが毎年 2 回調査して発表しているもので，世界最大の能力を持つ 500 のコンピュータシステムのランキングであり，性能は LINPACK ベンチマーク [3] で比較している．この表で Rmax が達成された最高の LINPACK 性能であり，#Proc はプロセッサの数である．この表より，高性能のコンピュータシステムはすべて超並列システムであり，この流れは今後一層加速し，多くの高性能コンピュータは数千から数万のプロセッサを持たざるを得なくなることがわかる．

プロセッサの並列化はメモリーから考えても必然である．たとえば典型的な DRAM はメモリーアクセス時間が 80 ns であり，この時間で 1 word のデータを読むことができるから 1 秒間では 12.5 MW のデータが読める．この数字は典型的な SRAM になると **メモリーアクセス時間** が 10 ns であり，この場合は 100MW のデータが読める．ここで 1 word を 32bit，すなわち 4B（バイト）とすると，それぞれ 50 および 400 MB/s のデータ読みだし速度となる．これを **メモリーのバンド幅** という（図 6.2）．

図 6.2: メモリーのバンド幅

ここでたとえば単精度（4 B/W）の演算を 20 GFLOPS の速度で行おうとすると 40 GW/s のデータ読みだしが必要になり，必要なメモリーバンド幅は 160 GB/s となる．これだけのバンド幅を確保するためには高速の SRAM を用いても 400 個のメモリーバンクから並列に読みだしを行わなければならない．もし，低速の DRAM を用いるなら 3200 個のメモリーバンクから並列に読みだしを行わなければならない．結局，どれだけ高速のメモリーを用いてもメモリーの並列化は不可避であり，この並列化に適合するアーキテクチャはプロセッサの並列化であると言える．図 6.3 にバンド幅を上げる方法を示す．

こうして，超高速演算を可能にするコンピュータは，光の速度の有限性の点からも，半導体材料の面からも，そしてメモリーバンド幅の点からも超並列となることがわかる．

■ 並列計算機の分類

並列計算機の分類には **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図 6.4 の 4 種類に分類するものである [1]．

並列計算機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．たとえば Thinking Machines 社の CM-2 では 2048 個の 32 ビット浮動小数点演算プロセッサが演算対象データを分割して同じ処理を行う負荷分散に用いられる．この方式は同期をとるためのオーバーヘッドがないこと，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こと，負荷分散に対する要求は多いので利用価値が高いこと，などの利点があるが，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

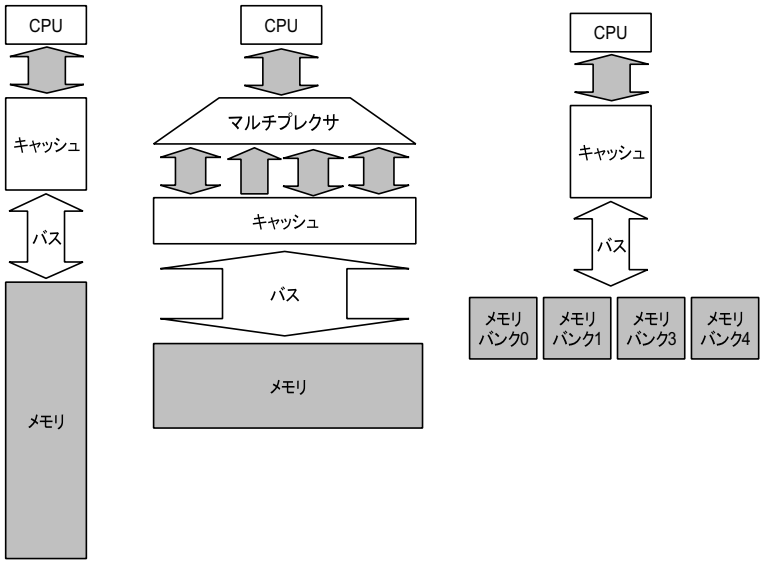

図 6.3: バンド幅を上げる方法 [4)]

結果の集約を効率化する仕組みが重要になること，そして多くの応用においては負荷を一様に分散させることが困難で，プロセッサのロードバランスが悪くなることが避けられないことなどの欠点がある．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．

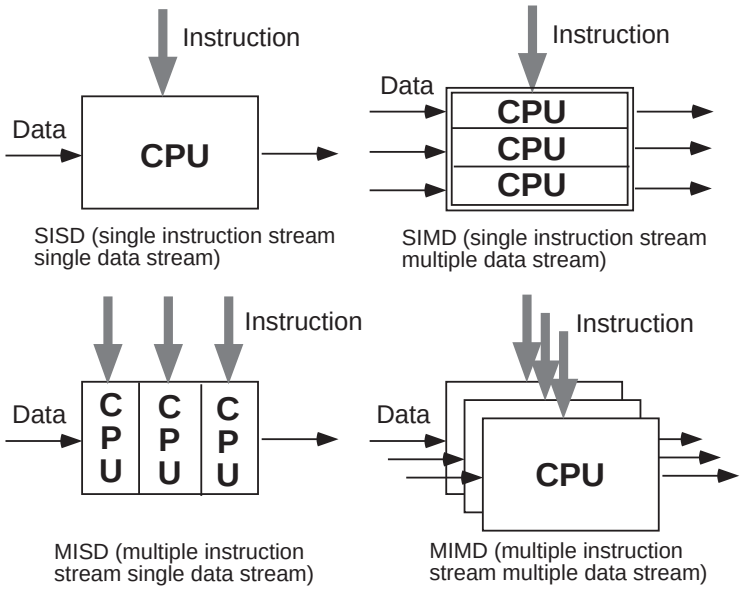

図 6.4: 並列計算機の分類

■ 並列計算機の構成方式

並列計算機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．

1) **共有メモリモデル** ：論理的にメモリを共有する．複数のプロセッサが通常のアドレス指定で直接的に読み書きができる共有メモリが存在する．

   (a) **集中共有メモリ** （centralized shared memory）：プロセッサとメモリが対称的になることから **対称型マルチプロセッサ** （ **SMP** : symmetric multiprocessor）とよぶ．

   (b) **分散共有メモリ** （distributed shared memory）：ローカルメモリとして分散配置する．

2) **メッセージ交換モデル** ：論理的にせよメモリは共有しない． **メッセージの交換** という形でデータ授受を行う． **NORA** （no remote access）モデルともいう．

   (a) メッセージの送信と受信は同期するか，非同期か．

   (b) メッセージの送受信においてブロック（封鎖：完了するまで処理を止める）するかどうか．

これらの分類を図示したものが図 6.5 である．なお，送受信のブロッキングはソフトウエアの問題であり，ハードウエアには現れない．このほかに，キャッシュの配置，コヒーレンスなどの観点から分類が可能であるが，専門的であるのでここでは割愛する．

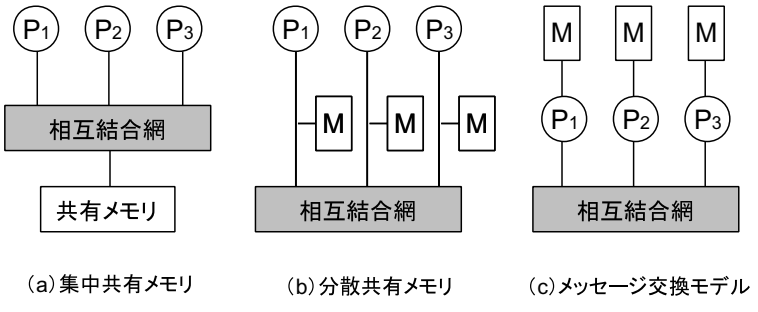

図 6.5: 並列計算機の構成法式

メッセージ交換モデルではネットワークのアーキテクチャや性能を除けば，慣用的なコンピュータをネットワークで結合したものと本質的には同じである．一方，共有型では対象問題を一括して一つのメモリーに展開することが容易で使いやすいが，その反面，共有メモリーのバンド幅は一定であるから，プロセッサ数はそれほど多くできない．普通は 4〜20 ぐらいである．

1990 年前後には TCP/IP ベースのネットワークで接続された複数のコンピュータの集合体を，仮想的に 1 台の大きな並列計算機として見せるためのソフトウエアである **PVM** (Parallel Virtual Machine) が開発された．これにより「計算機クラスタ」と呼ぶべき物理的な計算機群が登場した．1995 年前後には，イーサネットスイッチや 100Mbit Ethernet などの技術が一般にも普及し，比較的安価に高性能なネットワークの構築が可能となった．さらには計算機クラスタを指向した専用の高速ネットワークが登場し，一方ではＰＣの高性能化と低廉化により，コストパフォーマンスの優れた計算機クラスタの実現が可能となった．図 6.6 は計算機クラスタへの道程を示したものである．

■ プロセッサ間ネットワーク（相互結合網）

相互結合網 （interconnection network）は並列計算機を構成する複数のプロセッサやメモリを相互に結合し，その間における交信路を提供する．目標は **通信遅延** （ **レイテンシ** ：latency）を最小にし，スループット（throughput：処理能力）を最大にすることである．

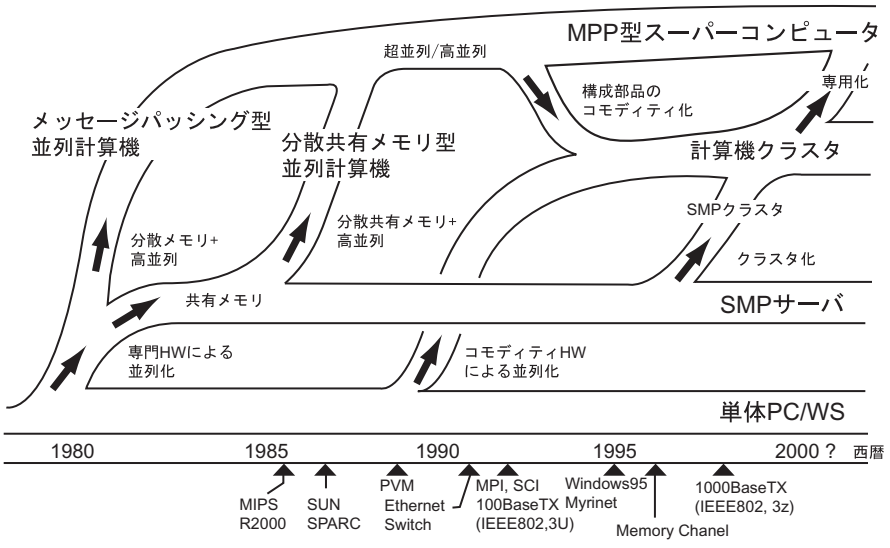

図 6.6: 計算機クラスタへの道程 [5)]

相互結合網のトポロジ（topology：幾何学的位相形状）の分類を図 6.7 に示す．ここで直接網とは入出力ノード（相互結合網に結合されるプロセッサやメモリなど）間の結合がはじめから固定されているものであり，間接網では結合が接続要求に応じて決まる．代表的な直接網を図 6.8 に示す．

上で述べた相互結合網は主として専用並列計算機のためのものである．すなわち，各相互結合網にはそれぞれの長所と短所があり，その専用並列計算機の目的に応じて相互結合網を設計することになる．一方，ＰＣクラスタではコストパフォーマンスをあげるために汎用のネットワークハードウエアを用いるか，もしくはクラスタ用に設計された汎用のネットワークハードウエアを用いることになる．このため，相互結合網の種類は用いるネットワークハードウエアに依存する．

たとえば，TCP/IP ベースの通常のハブを用いた場合では相互結合網はバス型となり，二つのプロセッサが通信しているときには他のプロセッサは通信が行えない．一方，レベルの高いスイッチングハブを用いれば独立したプロセッサ間通信が同時に行えることになる．

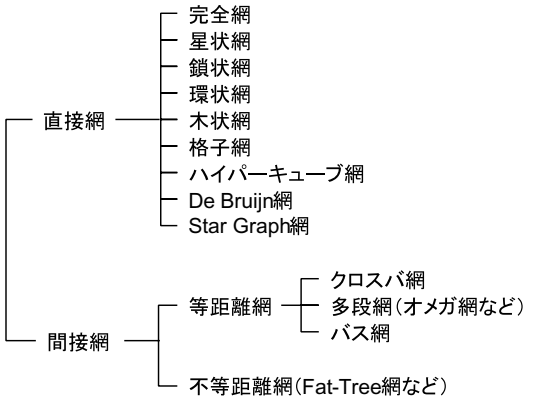

図 6.7: 相互結合網の分類 [1)]

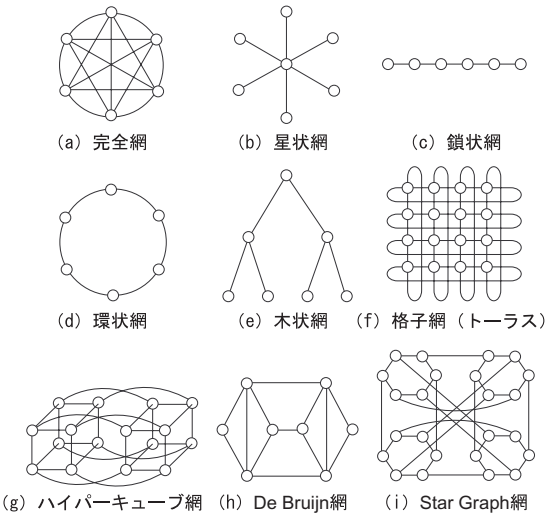

図 6.8: 代表的な直接網 [1)]

■ 並列処理のプログラム

並列計算機を用いるには並列処理のためのプログラムを作成しなければならない．これは用いる並列計算機の環境によって異なったアプローチとなる．たとえば，専用並列計算機でもっとも一般的な方法は **HPF** (High-Performance Fortran) などの並列プログラミング言語を用いて配列に関わる演算を中心に並列化を行う．こうした言語を用いればコンパイラが自動的に並列処理のコードを生成するため，複数のプロセッサを使っているという意識なしに並列計算が行える．

しかしながら，こうした言語では大規模な配列やマトリックスに関わる演算では極めて有効であるが，それ以外の利用では柔軟性が少ない．そうした場合には通常の逐次処理プログラミング言語とプロセッサ間通信用の関数を用いてメッセージパッシング型のプログラムを利用者が作成しなければならない．これには PVM (Parallel Virtual Machine) や **MPI** (Message Passing Interface) を用いる方法，あるいは各社仕様の C 言語や FORTRAN 言語を用いる．ＰＣクラスタでは PVM や MPI を用いてプログラム開発を行う．

■ 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，実装など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．図 6.9 は問題領域からハードウエ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数を示している．ＯＳとハードウエアでのギャップは，たとえばプロセッサの処理速度とプロセッサネットワークのバンド幅の相違による．

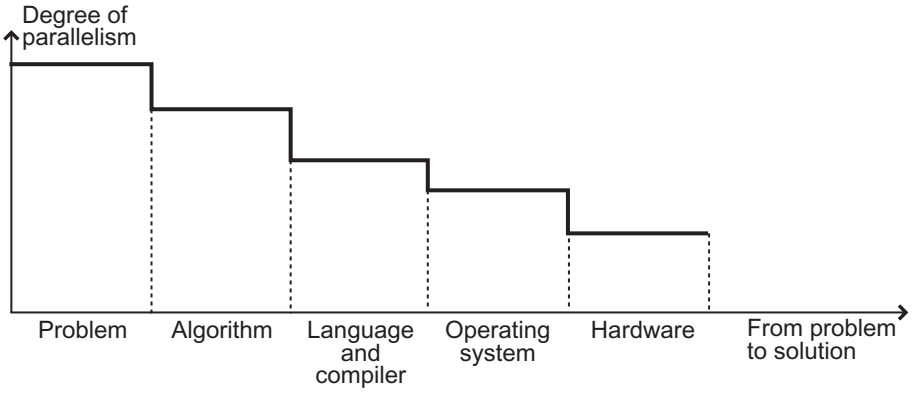

図 6.9: 各レベルにおける並列性 [5)]

マトリックス演算 $C = AB$ を考える．各マトリックスのサイズを $n \times n$ とすると，並列計算の複雑度は $O(\log n)$ である．すべての要素の乗算は並列で行うことができるが，n 回の加算に時間 $\log n$ が必要となる．しかし，n 回の加算を逐次的に行うとこのアルゴリズムの複雑度は $O(n)$ となる．ここに問題とアルゴリズムとの並列度のギャップが生じる．

種々の問題を並列計算機に写像し，プロセッサ間のロードバランスをとるのは極めて難しい問題である．さらに，これがうまくできてもどこかの段階に修正が生じると全体を見直す必要が生じる．すなわち，並列計算は図 6.9 のすべての段階が密に関連している．

$$E_p = \frac{T_1}{pT_p} \tag{6.1}$$

並列処理の効率 は簡単には式 6.1 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 を並列処理したときの計算時間 T_p と p の積で除したものである．ここで，逐次処理では最良のアルゴリズムを用いる必要がある．

$$R(f) = \frac{1}{(1-f) + f/R_p} \tag{6.2}$$

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としての処理速度を表したもので，全体の処理速度が低速部分に大きく影響されることをよく理解することができる．

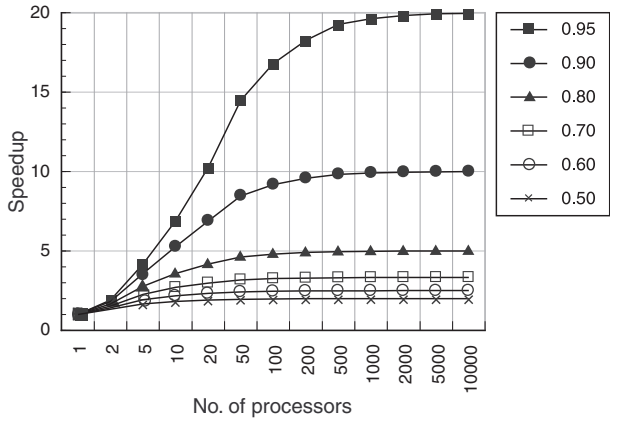

図 6.10: 並列化の速度向上

ここで，f は全体の処理における並列化可能な処理の割合であり，$R(f)$ は総合的な速度向上率，R_p は並列処理の **速度向上率** である．これを図示すると図 6.10 となる．たとえば逐次部分の割合がたとえ 5 %と少なくても，逆に言えば並列処理部分が 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**（grain size or granularity）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**（fine grain），ループまたはサブルーチンレベルのタスクを粗粒度（coarse grain），それらの中間的な大きさのタスクを **中粒度**（medium grain）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．特に，ＰＣクラスタの場合，一般にはプロセッサ間通信の速度が遅く，このため，細粒度での並列処理を行わせると複数台のプロセッサを用いているにも拘わらず，１台のプロセッサを用いたときより遅くなることもある．このため，特にＰＣクラスタを用いる場合には，できるだけ粒度を大きくしなければならない．

並列処理の効率に大きな影響を与える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．たとえば，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，いくら他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出来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とれるが，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．

そのために，最初から静的に負荷を分割するのではなく，その時々の各プロセッサの負荷を測定し，負荷を動的に均一化させる方法も有用である．しかし，この方法では負荷を管理するプロセッサが別に必要となり，しかも，負荷の測定のためにプロセッサ間通信が必要となり，別のオーバーヘッドが大きくなる．

■ 数値計算の構造

次に考えなければならない点は，数値計算に用いる概念モデルと計算モデルの組み合わせを数値計算の構造と呼ぶことにすると，数値計算の構造と並列処理の適合性はどうか，ということである．

図 6.11 はこの関係を示している．最初に特定のドメインでの問題があり，それを解くための概念モデルが作られる．このモデルの種類によってこの図でいう **数値ソルバー** を用いるか， **自然ソルバー** を用いるかが決まる．そしていずれも場合にも分割と写像が重要なポイントとなる．このブレークダウンができれば仮想的な計算機モデルを作ることができ，そのモデルに依存してどのようなアーキテクチャの計算機を用いるかが決まる．たとえば固体解析に有限要素法を用いるとして，それを並列化する場合には粗粒度の分割，すなわち領域分割により計算の並列化を行うこともでき，一方，細粒度の分割，すなわち内部でのベクトル計算のループを並列化することもできる．最近ではシミュレーテッドアニーリング，遺伝的アルゴリズム，あるいはニューラルネットワークなどの，この図でいうところの自然ソルバーを用いる場合も多く，この場合にも異なった並列化の計算モデルが考えられる．たとえば，遺伝的アルゴリズムで，全体的母集団における個体を複数プロセッサに割り当てて進化を進める方法や島モデルで個体群を分散させて独立して進化させ，ときどき移民させるというモデルもある．いずれにせよ，シミュレーションで重要な点は最初の概念モデルであり，そして次の計算モデルであり，これらの組み合わせで多くのタイプの並列処理が可能となる．

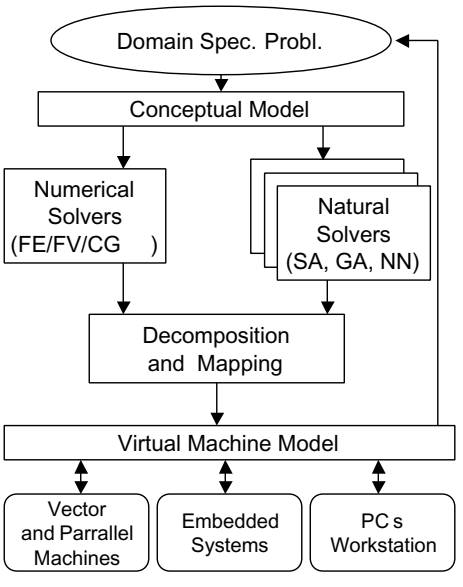

図 6.11: 数値計算の構造 [7)]

■ 数値計算の並列化

□ 差分法の並列化

ラプラスの方程式やポアソン方程式で表される種々の物理現象（重力場，静電場，静磁場，定常熱伝導，定常物質拡散，渦なし非圧縮性流れ，弾性体の平衡，結晶の成長など）の **シミュレーション** は差分法に基づいて行われることが多い．この場合，並列化は細粒度では Red-Black 法などによって色分けされた並列化可能な格子点をグループ化したものをプロセッサに割り当てるか，あるいは中粒度では複数の格子点を格子状にグループ化した領域を並列可能性に基づいてプロセッサに割り当てる．もしくは大粒度では全体をプロセッサ数だけの領域に分割して行う．いずれの方法でも Jacobi 型の緩和法を用いるのか Gauss-Seidel 型の緩和法を用いるのかによって計算モデルが異なり，計算速度と並列化効率のバランスが変わる．

□ 有限要素法における並列化

有限要素法で定式化されたシミュレーションモデルでは細粒度ではベクトル計算の並列化となり，これは HPF などの並列プログラミング言語で比較的容易に並列化できる．大粒度での並列化では領域分割法（Domain Decomposition）が中心となっている． **領域分割法** は有限要素法におけるデータ分割あるいは負荷分散である．この方法では対象を幾何学的に複数の領域に分割し，それらを対応するプロセッサに割り当てる．そして，領域内の計算が終了すれば互いに境界条件を交換し，再度，領域内の計算を行う．これを反復することで全領域の計算を行う．領域分割法では領域の分割の方法によってプロセッサのロードバランスが大幅に異なるため，並列化効率を上げるにはこの点に注意する必要がある．

□ 粒子法による物質の挙動シミュレーション

多数の粒子の相互作用を基本として各粒子の位置や速度を求め，粒子全体からなるシステムの挙動を計算するアプローチを **粒子法**（particle method）[8] という．相互作用が重力のときは天体力学における N 体問題となる．一方，相互作用がクーロン力の場合には分子動力学などのアプローチを用いて原子あるいは分子の配置を決定する解析が行える．

N 体問題の場合，重力は長距離力であり，N 個の粒子の計算には $N(N-1)$ 回の相互作用を求める必要がある．これは N が大きくなると膨大な計算となる．これを回避するには空間を複数のセルに分割し，そのセルに含まれる粒子の平均的挙動をセルの状態として考え，ある一定の距離以上のセル内の粒子についてはセルとしての代表的な状態で近似する高速粒子法が必要となる．これによって計算回数は $N \log N$ ぐらいまで減少させることができる．ただし，この場合には空間に固定されたセル内の粒子の出入りを監視することが必要になる．

一方，分子動力学では原子間のクーロンポテンシャルが長距離で減衰が激しいことを利用し，ある一定の距離以上では他の粒子を考えなくても良い．これによって膨大な数の原子を取り扱うことができる．この場合に原子間距離の計算回数はやはり $N(N-1)$ 回となるため，たとえポテンシャルを計算しなくても距離計算の負荷が大きく，これを避けるには着目している原子の近傍粒子とそうでない粒子を分類し，その分類自体は時折行うことで毎回の距離計算自体を無くす **粒子登録法**（Bookkeeping method）が用いられる．

このような粒子法の並列化は大粒度では空間でのセルあるいはクラスター化した粒子群を一つのプロセッサに割り当てる方法であり，細粒度での並列化はベクトル計算における inner loop の並列化である．ただし，後者においてもプロセッサのロードバランスを良好にするために単一プロセッサの場合とは異なるアルゴリズムが必要となる．

■ おわりに

現在の自動並列化プログラミング言語は基本的に DO-LOOP の並列化が中心である．特に inner loop の並列化には効果的である．しかしながら，大規模シミュレーションの多くが最終的には膨大なベクトル計算に帰着するのはあくまでも図 6.9 でいうところの数値ソルバーを用いる場合である．しかしながら，同図でいう自然ソルバーにはまだまだ大きな発展の可能性があり，それらのアプローチでは種々の並列化の方法が可能となる．しかも，そうした自然ソルバーは本質的に並列処理を内在している場合が多い．このため，そうした自然な並列性を抽出して並列処理モデルを構築することはこれからの並列処理の可能性を大きく広げるものと考えられる．

■ 参考文献

1) 情報処理学会編. 情報処理ハンドブック. オーム社.

2) IBM. Kasparov vs. deep blue, 1999. http://www.research.ibm.com/deepblue/.

3) Mannheim University and Netlib. Top500 supercomputing sites, 1999. http://www.netlib.org/benchmark/top500.html.

4) パターソン, ヘネシー, 成田訳. コンピュータの構成と設計. 日経 BP 社.

5) 森真一郎, 富田真治. 並列計算機アーキテクトからみた計算機クラスタ. 情報処理, Vol. 39, No. 11, pp. 1073–, Nov. 1998.

6) D.I. Moldovan. *Parallel Processing from Applications to Systems.* Morgan Kaufmann.

7) P.M.A. Sloot. *Modelling for Parallel Simulation: Possibilities and Pitfalls.* EUROSIM.

8) E.F. Van de Velde. *Concurrent Scientific Computing.* Springer-Verlag.