目次

- 並列処理
- オペレーティングシステムの概要

★9　並列処理

★9.1　並列処理とは

以前説明した**パイプライン処理**は，単一の CPU の性能向上のための仕組みだった．ここでは，複数の CPU を互いに接続し同時に複数の処理を行う**並列処理**について説明する．例えば，100 個の数値の和を計算するとしたら，通常の CPU1 つの場合には一般に加算 100 回分の処理時間がかかる．しかし，2 つの CPU を同時に利用できるなら，それぞれに 50 個の数値の和を求めさせ，最後にどちらかのプロセッサにそれらの和を計算させればよいので，ほぼ半分の時間で済むことになる (☆ 1)．

並列処理: parallel processing

☆ 1) ただし，後でみるようにいつでもそううまくいくわけではない

並列処理の様式は，「命令およびデータの流れが単一か複数か」によって次のように分類できる（Flynn の分類）．

SISD(single instruction stream, single data stream)
1 つのデータに対して一度に 1 つの命令を実行（並列処理でない通常のコンピュータ）

SIMD(single instruction stream, multiple data stream)
複数のデータに対して同時に同じ命令を実行できる

例: $A+P, B+Q, C+R$ を同時に実行

MISD(multiple instruction stream, single data stream)
1 つのデータに対して同時に複数の異なる命令を実行できる

例: $A+P, A-P, A\times P$ を同時に実行

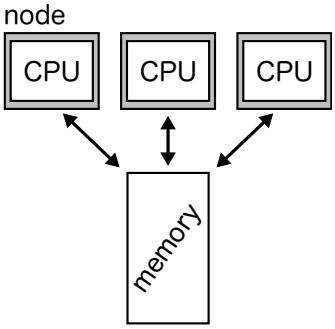
MIMD(multiple instruction stream, multiple data stream)
複数のデータに対して同時に複数の異なる命令を実行できる

例: $A+P, B-Q, C\times R$ を同時に実行

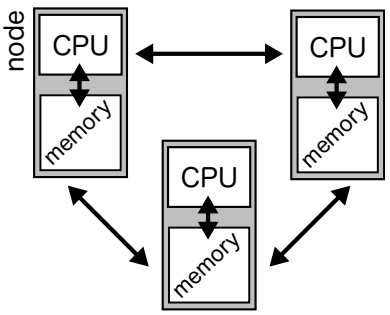
並列処理における処理単位（CPU など）のことを**ノード**または **PE**(Processor Element) という．並列処理の方式は，ノードとメモリとの接続方式によって**共有メモリ型**と**分散メモリ型**に二分することもある．共有メモリ型では，名前の通り全てのノードが単一のメモリ領域を共有するのに対して，分散メモリ型では個々のノードが独立したメモリ領域をもつ（下図参照，(☆ 2)）．並列処理の方式は，この他にもノード間を接続するネットワークのトポロジによって分類したりもできるが，この授業では省略する．

共有メモリ: shared memory, 分散メモリ: distributed –

☆ 2) 共有メモリ型ではノード間でのデータの受け渡しが容易な反面，メモリアクセスが競合して性能低下しやすい．分散メモリ型ではメモリアクセス速度を上げられる反面，ノード間でのデータの受け渡しがネックとなる．



共有メモリ型



分散メモリ型

★ 9.2 並列コンピュータの分類

並列処理を行うコンピュータの類型を以下に示す．これらは複合された形態となることも多い (☆ 3)．

☆3)「複数のマルチコア CPU から成る SMP 構成の PC をノードとしたクラスタ」など．

マルチコア ひとつの CPU パッケージのなかに複数の CPU コアを入れたもの．特に 2 つのコアを載せたものを**デュアルコア**，4 つ載せたものを**クアッドコア**という．近年では，PC に搭載される CPU の多くがマルチコアとなっている．それらの場合，主記憶を複数の CPU コアで共有する共有メモリ型の並列処理方式である（下図参照 (☆ 4)）．

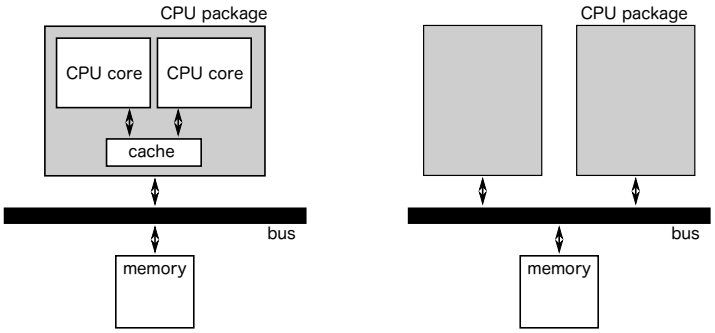
マルチコア: multi core．デュアル–: dual –．クアッド–: quad –．

☆4) 下図では 2 つの CPU コアが 1 つのキャッシュメモリを共有しているように描いているが，コア毎に独立したキャッシュを備えることもある (1 次キャッシュはコア毎，2 次キャッシュは共有，といった形態もある)．

対称型マルチプロセッシング (SMP) 同じ型の CPU を複数単一のバスに接続した共有メモリ型の構成．プロセッサの中に特別な役割のものがなく，同一のものを用いることから「対称」型といわれる (☆ 5)．プロセッサ数は 2 から数十程度．サーバ等に用いられることが多い．

SMP: Symmetric Multi Processing

☆ 5) PlayStation3 などに搭載されている “Cell” は，機能の異なる複数の CPU コアを組み合わせているので「非対称」．



マルチコア　　マルチプロセッサ

超並列コンピュータ／クラスタ ．多数のノードを接続した形態．一般にノード毎に独立したメモリを持つ（分散メモリ型である）ことが多い．ノード間の接続方式によっていくつかに分類できる．**超並列コンピュータ** は，非常に多数のノードを超高速の通信路で密に接続した大規模な並列計算機である．それに対して，ノード間の接続にそれほど高速ではない，一般的な LAN と同程度のネットワークを用いるものを，**クラスタ**という．現代の**スーパーコンピュータ**のほとんどは，超並列コンピュータやクラスタの一種である．これらのノードに組み込まれる CPU は，PC に搭載されるものと性能的に大差ないものである場合も多いが，多数のノードを接続した並列処理によって超高速な演算能力を実現している (☆ 6)．

超並列–: massively parallel computer, クラスタ: cluster

☆6) ノード数が数十万にも達するものもある．

☆ 7) 電波望遠鏡の観測信号を解析して地球外生命体からの信号を探す SETI@Home や，たんぱく質の折りたたみ構造を解析して病気の治療に役立てようとする Folding@Home，素数の探索など，様々なものがある．

分散コンピューティング／グリッドコンピューティング インターネットなど一般のコンピュータネットワークを介して複数のコンピュータをつなぐことで仮想的な高性能並列計算機を構成するもの．不特定多数の参加者が自分の PC やゲーム機の計算時間を提供するプロジェクトとなっているものもある (☆ 7)．

分散コンピューティング: distributed computing
グリッド: grid, 格子，網目

GPGPU 近年では，PC のグラフィックスカードなどに搭載されている画像処理用のプロセッサ (**GPU**) の性能向上がめざましく，GPU を汎用の情報処理に利用することも行われるようになってきている．このような処理を GPGPU という．GPU はもともと画像処理に特化しているため，並列処理に向いている．

GPU: Graphics Processing Unit
GPGPU: General Purpose computing on GPU

★ 9.3 並列処理の問題点 − n ノードで並列処理したら n 倍高速か？

ここまでの話からすると，「n ノードで並列処理したら 1 ノードの場合の n 倍の性能になる」と思うかもしれない．しかし，実際にはそこまでの性能向上は見込めない．その理由には，主に次のようなものがある．

まず第一に，与えられた作業がうまく n 等分できるとは限らない (☆ 8)，ということが上げられる．また，等分できたとしても，処理に依存関係があれば，あるノードの結果を利用するために別のノードが待たされて，無駄が発生してしまう．さらに，ノード数が多くなると，ノード内での処理時間と比べてノード間の通信に要する時間が無視できなくなってくる，という問題もある．

☆ 8) スーパーコンピュータを必要とするような科学技術計算の分野では，小さい部分に分割しやすくかつ部分同士の独立性の高い（「並列度」が高いという）問題が多い．そのため超並列計算機が威力を発揮しやすい．

以上からもわかるように，一般のプログラムでは全体が並列化可能というわけではない．では，プログラム中で並列化可能な部分の占める割合がわかっているときに，n ノードで並列処理したら全体として性能が何倍になるかを予測するにはどうしたらよいだろうか．このような問題を考える際には，**アムダールの法則**: Amdahl's law

$$(\text{改善後の実行時間}) = \frac{(\text{改善に関係する部分の実行時間})}{(\text{改善度})} + (\text{改善と無関係な部分の実行時間})$$

が役に立つ．これは，あるシステムの一部を改良したときに，全体としてどれくらい性能が向上するかを見積もるためのものである．

一方，ソフトウェアの観点からは，並列計算機の能力を活かしたプログラムをいかにして作成するか，ということが問題となる．ソフトウェア開発者が並列処理用の言語／ライブラリを駆使すれば効率的なプログラムを開発できるかもしれないが，それにはかなりの手間がかかる．普通のプログラムが自動的に並列処理されるのが理想であるが，そのような仕組みを実現するのは大変難しい．

Q1. A,B ふたつの部分から成るプログラムの実行時間を短縮することを考える．改良前は，A,B にそれぞれ 0.8 秒，0.2 秒かかっていたとする．このとき，B の処理を 10 倍高速にした場合と A の処理を 2 倍高速にした場合のそれぞれについて，改良後の実行時間を求めよ．

Q2. ほげお君は，ある並列計算機で動くプログラム P を作った．P の処理全体の 8 割は完全に並列化可能であるが，残りは並列化できないという．P を 1000 ノードから成る並列計算機で実行すると 1 ノードの場合の何倍高速か．

★10 オペレーティングシステムの概要

★10.1 オペレーティングシステムとは

現代のコンピュータシステムは，ハードウェアだけでなく**オペレーティングシステム** (**OS**) と呼ばれるソフトウェアがなければ動かない．OS は，コンピュータを使いやすくするために，かつ効率的に使えるようにするために存在しており，ユーザや**応用ソフトウェア**とハードウェアとの仲立ちをするものである．

OS の核となるのは，ハードウェアやユーザプログラムを管理する**制御プログラム**と呼ばれる部分である．狭義ではこの部分のみを OS と呼ぶが，広義では，プログラムの実行や作成に必要な道具（ソフトウェアライブラリやコンパイラ等）や，ファイル一覧ソフトやエディタ等の基本的なソフトウェアまで含めることもある (☆ 9)．

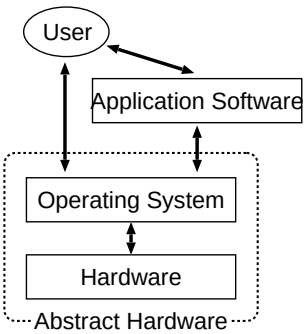
したがって，OS とそれ以外の部分との境界は曖昧である．例えば，PC の代表的な OS である Windows では，**GUI**(Graphical User Interface) の機能も OS と一体で提供されているので OS の機能と思いがちであるが，Linux 等では OS とは切り離されており，ユーザが好みの GUI をもつソフトェアを選択することができる．

応用ソフトウェアの実行プログラムは特定の OS 向けになっているので，通常は異なる OS のもとでは動作しない．

Operating System

応用ソフトウェア: Application Software, アプリケーションソフトウェアとも言う

☆ 9) 広義の OS を**基本ソフトウェア**と呼ぶこともある．また，基本—と応用—の中間的な位置づけのものを**ミドルウェア**と分類することもある (仮名漢字変換ソフト等)．



★10.2 オペレーティングシステムの役割

上述のように，OS の目的は主に使いやすさの向上とシステム性能の向上である．その役割には，次のようなものがある．

資源の管理: コンピュータシステムはプロセッサ，メモリ，外部記憶装置，プリンタ等のハードウェア資源と，プログラムやデータ等のソフトウェア資源から成っており，ユーザや応用ソフトはこれらの資源を要求する．OS は資源の状態を管理し，複数の要求が競合した場合でも適切な対応をする (☆ 10)

資源（リソース）: resource

☆ 10) 例えば, 他のプログラムを待たせる，エラーを返す等.

コンピュータの利用効率の向上: 複数の処理を同時に行う際に，資源の割当順序等をうまく制御することで，全体としての処理のスループットを向上させる．

ハードウェアの抽象化:ユーザや応用ソフトが多種多様なハードウェアを直接操作しようとすると，個々のハードウェアの違いに対応したプログラムを作成しなければならない．OS はハードウェアの操作を抽象化した統一的なインタフェースを提供し，ユーザの操作や応用ソフトの開発作業を容易にする (☆ 11)．

☆ 11) 例えば UNIX では，様々な周辺機器から／への入出力をファイルの読み書きとして行えるようになっている．

★ 10.3 オペレーティングシステムの発展過程

OS の発展過程を概観しよう．

黎明期である 1940 年代のコンピュータには，OS は存在しなかった．ユーザがひとかたまりの処理（**ジョブ**という）をコンピュータに与えたり処理結果をコンピュータから取り出す際には人手を介していたため，一つのジョブを終えてから次のジョブを開始するまでの間，コンピュータは無駄に停止していた．

ジョブ: job

1950 年代になると，複数のジョブをまとめて連続的に処理する，**バッチ処理**の機能を備えた OS が開発された．これにより，コンピュータをジョブとジョブの間で停止させることなく効率的に利用できるようになった．

バッチ処理: batch processing

1960 年代には，より効率的にコンピュータを使用するための仕組みとして，**マルチプログラミング**という方式が開発された．また，同時に複数のユーザがコンピュータを対話的に使用できる**タイムシェアリングシステム**が開発された．1964 年には，これらの技術を取り入れた実用レベルの OS として，OS/360(☆ 12) が発表されている．この他，**仮想記憶**（次回以降説明します）の概念や，後の UNIX に大きな影響を与えた OS である Multics が登場したのもこの頃である．

マルチプログラミング: multi-programming，マルチタスク (multi-tasking) とも．

タイムシェアリングシステム: time-sharing system

☆ 12) OS/360 は，IBM の System/360 という汎用コンピュータに搭載された OS. コンピュータの発展に大きな影響を与えたものとして有名．

Multics はいくつもの革新的な技術を取り入れた大規模な OS であったが，開発に時間を要したこともあって普及するには至らなかった．この Multics のプロジェクトにかかわっていた技術者たちによって 1970 年前後に作られたのが **UNIX** である．設計がシンプルで柔軟である，異なるアーキテクチャのコンピュータへの移植が容易である (☆ 13) 等の理由により，広く使われるようになった．

☆ 13) それまでの OS はアセンブリ言語で書かれるのが普通で, UNIX も最初はそうだったが，すぐに C 言語で書き直された（C 言語はそのために作られた）．ちなみに，現在では UNIX は OS の名称ではなく仕様の名称となっている．

1970 年代後半以降，PC の登場にともなって様々な PC 向け OS が開発された．1980 年代には，MS-DOS, Mac OS, Microsoft Windows などが誕生している．

1980 年代後半以降，PC の性能向上にともない，UNIX のような OS（UNIX 系 OS）を PC 上で動作させようという取り組みがあちこちで行われるようになった．Linux(☆ 14) もそこから生まれた．

☆ 14) 1991 年当時フィンランドの大学生だった Linus Torvalds が一人で一から Linux（のカーネル）を作った．