高性能コンピューティング特論講義メモ（4）

～ベクトル処理とベクトルプロセッサ～

１．データ供給と演算の融合

これまでに述べたように、HPC において最も重要なのは、バンド幅である。すなわち、プロセッサで処理されるデータをメモリ(あるいはネットワーク)からいかにして高速に供給するか、ということが処理の高速化のキーとなる。しかし、仮にメモリのデータ供給バンド幅が、アプリケーションの要求するそれに対して十分だったとしても、それだけでは高速化を達成することはできない。つまり、プロセッサのアーキテクチャとして、メモリの高いバンド幅を十分に活かすための工夫がなされていなければならない。

例えば、キャッシュベースのプロセッサを考えよう。通常にループ処理を行なっていると、ワーキングセットがキャッシュ容量に収まっていない限り、キャッシュミスヒットが生じる。ここでは、メモリのバンド幅が広いことは、キャッシュミスヒット時のデータの埋め込み速度を向上させるために役立つ。しかし、依然としてミスヒットは存在し、ループ処理中の演算を、常に連続的に行なうということは不可能である。(実際には、キャッシュベースのプロセッサで、メモリバンド幅を有効利用するためにいくつかの手法が存在する。これらに関しては別の機会に触れる。)

そこで、最も単純で効果的なデータ供給システムは何かを、ループ処理の形から考えてみよう。以下のようなループを考える。

```
for(i=0; i < N; i++)
    s += a[i];
```

このループでは、配列の各要素間にデータ依存性は存在しない。従って、N 回のループ処理は全て独立に行なえる。変数 s はスカラ変数なので、レジスタにデータを溜めることにより、メモリアクセスは生じないものとする。プロセッサが 1 クロックで 1 つの浮動小数点乗算を行えるとすると、このループを連続処理するためには、メモリ側は理論的に、1 クロック 1 当たり 1 つの浮動小数点データの読み出しが可能である必要がある。一般的な DRAM を主記憶に使ったとすると、このバンド幅を提供するには、一般的にバンクメモリによる擬似パイプラインメモリを用いる必要がある。ここでは、多数のバンクメモリに対し次々にアクセスをかけることにより、結果的に必要なバンド幅を得ることは可能である。また、この処理ではメモリ番地を連続アクセスするため、比較的容易にバンクに対するアクセスを発生できる。

ここまでは話は簡単であるが、ここで問題となるのは、擬似パイプラインメモリでは、

実際のデータ読み出しはそのバンクに対する読み出しアクセスから、メモリの読み出し時間を経てからでないと行なえないという点である。すなわち、バンド幅は確保できているが、個々のデータ読み出しのためのレイテンシ(遅延時間)は存在するため、これを何とかしなければならない。このための唯一の方法は、データがプロセッサで必要になるよりも前に、予めプロセッサの内部(あるいは手前)にメモリからデータを送り込んでおき、演算装置にスムーズに供給するということである。この処理を、全てのデータに対し常に行ない、データを連続処理する。この処理はバンクメモリの性質と相性が良く、結果として全ての処理を理想的に進めることができる。すなわち、キーワードは「データの前倒しアクセス」である。

２．ベクトル処理とベクトルプロセッサ

上述のような例はかなり理想的に見えるが、実際の科学技術計算にはこのような単純なループ、あるいはそれに近いループ処理はかなり多く見られる。すなわち、配列要素間のデータ依存性がなく、メモリアドレス上のデータアクセスのパターンを予め予測可能な(つまり定型的な)ループがよく現れる。もちろん、大前提となるのは基本的なデータ構造が配列、すなわちベクトル型のデータであるということである。そこで、このような処理を一般的にベクトル処理(vector processing)と呼ぶ。

バンクメモリを用いた擬似パイプラインにより、メモリのバンド幅を物量によってカバーすることは、ベクトル処理の基本である。これに加え、先述のレイテンシ隠蔽(latency hiding または latency tolerating)を行うためには、ベクトル処理に特化された特別なアーキテクチャが有効である。このようなプロセッサアーキテクチャを、ベクトルプロセッサ(vector processor)と呼ぶ。ベクトルプロセッサを構成する要素は原理的に次の 3 つである。

- **多数のレジスタ(ベクトルレジスタ)**
- **メモリからレジスタへの(あるいは逆方向の)十分なバンド幅**
- **バンクメモリに対するアクセスアドレス先行発行機構**

処理のおおまかな流れは以下のようになる。なお、以下は 1 つのベクトル要素に対する処理を順番に示したものである。

1. 後で使用されるデータの番地を予めバンクメモリに与え、データ読み出しを行なう。
2. バンクメモリから読み出されたデータをベクトルレジスタの 1 つに格納する。
3. ベクトルレジスタのデータを用いて演算を行ない、結果を別のベクトルレジスタに格納する。
4. 結果データを持つベクトルレジスタから、バンクメモリにデータを書き込む。

もちろん、これらの処理をベクトルデータ１つ１つに対して順番に行なうのではなく、実際にはこれらの処理をパイプライン的に次々に実行する。このため、ベクトル要素間のデータ非依存性が重要になるわけである。先に示したループの例では、生成されたデータの格納先がスカラ変数であったため、実際には上述の最後のステップは実行されないが、一般的には最後のデータ書き戻しのフェーズも含め、パイプライン処理される。

実際のベクトルプロセッサでは、これらのパイプライン処理の１つ１つを命令によって遂一指示することはない。ベクトルプロセッサには、ベクトルロード、ベクトル演算、ベクトルストアなどと呼ばれる一群の特殊な**ベクトル命令**が存在する。例えば、ベクトルロードは通常のスカラのロードと異なり、ロード開始番地・データサイズ・ストライド・終了番地等の情報を特殊なレジスタ(ベクトル制御レジスタ)にセットした上で発行される。これにより、各要素に対するロードが順次実行される。また、ベクトルロード命令と組になり、ベクトル演算命令が存在し、これはベクトルレジスタに対して順番に演算を行なう。ベクトルストアはベクトルロードの逆である。

ベクトルプロセッサのプログラムでは、一連のベクトル命令が塊になって配置され、相互に同期を取りながら上述の処理が実行される。このため、ベクトルプロセッサには特殊なコンパイラ(ベクトルコンパイラ: vector compiler)が用意され、これらのベクトル命令を構造的に配置し、効率的なベクトル処理を行なう。従って、ベクトルコンパイラはループ内のデータアクセスや演算の順序、データ間の依存性といったあらゆるループ特性を解析する必要がある。

３．ベクトルプロセッサの性能とその高速化

ベクトルプロセッサにより、擬似パイプラインメモリを有効に活用し、円滑なベクトル処理を行うことが可能となる。しかし、それだけでは本当の高速性を得るには十分ではない。

ベクトル処理の最大の特徴は、データ読み出し・演算・書き込みという全てのフェーズをパイプライン化している点である。今、メモリのバンド幅はプロセッサ内のデータ演算を支えるだけの十分な量があると仮定すると、N 個のデータの総処理時間 T(N)は以下のようになる。

$$T(N) = T_s + T_p \times N + T_e$$

ここで、T_s は最初のデータにアクセスをかけてからそれがレジスタに格納されるまでに要する時間、T_p は 1 つのデータの演算に要する時間、T_e は最後のデータが算出されてからそれをレジスタからメモリに書き戻すのに要する時間である。パイプライン演算の中身だけを考えるならば、上式の中央の項だけで良いが、実際には最初のデータを準備するための時間と、最後のデータの始末をするための時間が加わるわけである。ここで、T_s のことをパイプラインのスタートアップオーバヘッド(start-up overhead)と呼び、T_e のことをパ

イプラインのクリーンアップオーバヘッド(clean-up overhead)と呼ぶ。これらのオーバヘッドはパイプライン処理において避けられないものである。これらが一定だとすると、1データ当たりの平均処理時間を縮めるための最も簡単な手段は、データ数 N をなるべく大きくすることである。全体のバンド幅が確保されているならば、N をいくら大きくしても処理は破綻しない。よって、なるべく大きな問題を対象にするということは、ベクトル処理の効率化の上で重要な要件となる。ただし、これには実際に許される総演算時間や、主記憶のメモリサイズ等の物理的制約による上限がある。

以上がベクトルプロセッサの基本性能指標である。スタートアップとクリーンアップのオーバヘッドを一定だとすると、全体の処理をより高速化するためには、どうしても T_p を小さくする必要がある。例えば最初に示したループ処理の例では、浮動小数点加算という演算を 1 クロックで行なうとすると、システムのクロック周波数の上限はハードウェアの制約から決まっているため、T_p にはどうしても上限が存在する。これをさらに向上させるためにはクロック周波数をさらに上げる必要がある。そのための最も有効な手段は、1クロックで済ませていた加算処理の内部をさらに細かいパイプラインに分け、数クロックを要するようにする代わりに、プロセッサの周波数自体を上げるという方法である。

つまり、演算のステージをより細かいパイプラインで構成し、その分各ステージの回路の内容を簡単化させ、より高い周波数で動作させるようにするわけである。ベクトルプロセッサでは全ての処理がパイプライン化されているため、この改良によってスタートアップのオーバヘッドは若干増加するが、T_p の部分を短縮することができ、結果として処理性能は向上する。

一般的に、パイプライン構造の 1 ステージ分の大きさをピッチ(pitch)と呼ぶ。上記の改良は、パイプラインのピッチを短かくし、その分動作周波数を高くすることにより、スループットを向上させているわけである。ピッチを短くすればするほど、パイプラインの全ステージ長は一般的に長くなる。このような構造のパイプラインを「深いパイプライン(deep pipeline)」と呼ぶ。現在のベクトルプロセッサは非常に細かいパイプラインピッチを持っており、最高速のマシンではそのクロック周期は数百～数十 psec まで短縮されている。これを支えるために、メモリのバンク数は大幅に増大され、バンク数が数千に達するものまである。

４．ベクトル型計算機

以上のようなベクトルプロセッサを基本とする科学技術専用計算機をベクトル計算機(vector machine)と呼ぶ。HPC における計算性能需要に伴ない、現在のベクトル計算機は、その内部にベクトルプロセッサを複数持つような構造になっている。すなわち、データ処理のパイプラインが複数本存在し、独立なデータ流に対して並列処理を行なう。従って、データ流は 1 つのパイプライン内で時間並列・多重化されるだけでなく、複数のパイプラインによって空間的にも並列化されるわけである。さらに、このような複雑なマルチパイ

プラインを持つベクトルプロセッサを複数台結合し、全体構成をも並列化するのが一般的になっている。このようなベクトルプロセッサの並列化を行なったシステムを特に並列ベクトル計算機(parallel vector machine または parallel vector processor)と呼ぶ。

ベクトル型計算機を初めて商用化・実用化したのは、Seymore Cray が創立した Cray 社であった。その初代マシンである Cray-I は、当時としては画期的な 100MFLOPS オーダの計算能力を持っており、その後も Cray 社は世界のベクトル計算機界をリードしていた。その後、日本の三大コンピュータメーカ(NEC、富士通、日立)はそれぞれ独自のベクトル計算機を実現し、NEC は SX シリーズ、富士通は VP シリーズ(後に VPP シリーズ)、日立は S シリーズを生み出した。NEC は SX-2 で世界初の GFLOPS 性能を実現した。その後、ベクトル計算機は並列ベクトルへと移行し、1990 年代半ばまで HPC 界の王道であった。NEC はその後も SX-8 までのシリーズを生み出している。SX シリーズの大きな飛躍は、国産スーパーコンピュータ作成の国家プロジェクトとしてこれまで最大規模の予算を投じて実現された、「地球シミュレータ（Earth Simulator)」の完成である。これが後に SX-6 として、ネットワーク部分を変更して商用化された。また、富士通は VPP-500 シリーズからガリウムヒ素を用いたプロセッサを実現し、動作周波数を大幅に引き上げた。

1990 年代初頭からスカラプロセッサを基本とする超並列計算機が現れ始め、現在では世界トップレベルの最高性能を持つマシンのほとんどは超並列計算機で占められている。この中にあって、日立は S-3800 シリーズを最後に純粋なハードウェアによるベクトルプロセッサの生産を中止し、筑波大学の CP-PACS プロジェクトと協力して、ソフトウェア制御とパイプラインメモリを組み合わせた擬似ベクトルプロセッサ機構により、SR-2201 を生み出し、現在はその後継機種である SR-8000、SR-11000 に技術が受け継がれている。しかし、SR-11000 からはプロセッサに IBM 製の Power プロセッサファミリーが使われ（現在は Power5）ている。

富士通も、VPP-500 の後に VPP-700、VPP-5000 までを製作した後でベクトル機路線から撤退した。現在は SPARC アーキテクチャを独自で高性能化した富士通製 SPARC をベースとし、100 プロセッサ程度までを共有メモリ結合する PrimePower シリーズに主力が移っている。このように、現在日本国産スーパーコンピュータとしてベクトル計算機路線を維持しているのは NEC のみである。

一方、元祖ベクトル計算機メーカである Cray は、Seymore Cray の死後、会社の方針を転換し、ベクトル機の生産を一時中止した。そして、グラフィックプロセッサと共有メモリ型超並列機を主力とする SGI に吸収合併され、さらにその後で、世界唯一の商用マルチスレッドアーキテクチャを提供する Tera Computer に買収されている。しかし、Cray 社は米国唯一のベクトル計算機メーカーとしてその後継機種も作り続けており、最近ではベクトルプロセッサ、スカラープロセッサ、マルチスレッドを混在させたハイブリッド型アーキテクチャを提案・　構築している。

５．GPU：新しいタイプのベクトル処理

GPU (Graphics Processing Unit) は、ノード PC を含む一般的な PC 全てに標準で備わっているプロセッサで、ディスプレイへの画像／テキスト表示やアニメーション出力を高速処理する。特に近年ではディスプレイの大型化・高解像度化や、PC ゲームの発展に伴い、極めて高い演算能力が要求されるようになってきた。GPU では高解像度での表示能力はもちろんであるが、これらの高度な処理に対応するために浮動小数点演算性能も飛躍的に高まってきている。そこで、この高い潜在的演算能力を HPC に流用し、GPU 上で単なるグラフィクス処理だけでなく、一般的な高性能演算を行わせるという GPGPU (General Purpose GPU)と呼ばれる技術が発展しつつある。

近年の GPGPU の発展は目覚ましく、2009 年には NVIDIA 社が Fermi と呼ばれるアーキテクチャを発表、それに基づく GPU 製品として C2050, C2070 が登場した。これらは理論ピーク演算性能 515 GFLOPS で、単純比較で言えば通常のマルチコア CPU とは 1 桁以上違う性能を持っている。GPU における高性能化の鍵は、GPU 本体のチップ上に数百個のコアプロセッサが搭載され、これを非常に高いメモリバンド幅が支えている。つまり、演算装置及びメモリバンド幅としては、従来のベクトル計算機が求めているものに匹敵する物量が備わっていると考える事ができる。このため、GPGPU テクノロジを「現代のベクトル技術である」とする見方もある。