オープンスパコンのＯＳとアーキテクチャの基礎

松 葉 浩 也

東京大学情報基盤センター

1. はじめに

東京大学の情報基盤センターには 2007 年現在 SR11000/J2 が導入されており，2008 年度には「オープンスーパーコンピュータ」と呼ばれる新たなスーパーコンピュータが導入される。オープンスーパーコンピュータはその名の通り，一般的に仕様が公開されているハードウェアおよびオープンソースのソフトウェアを中心に構成されたスーパーコンピュータである。このオープンな性質により，ユーザーはより深くスーパーコンピュータを理解しその知識を最適化に役立てることができる。

本稿はスーパーコンピュータのアーキテクチャを理解する助けとなるよう，プロセッサ単体から並列計算機，通信の仕組みに至るまで，スーパーコンピュータに欠かせない技術を広く紹介するものである。本稿で扱える範囲は非常に限られているが，それでもスーパーコンピュータの動作原理を少しでも理解していただき，プログラムの最適化などに役立てていただければ幸いである。

2. プロセッサアーキテクチャの基礎

最初にスーパーコンピュータの最も基本的な構成要素であるプロセッサについて述べる。近年，スーパーコンピュータと一般のコンピュータとの違いは主にプロセッサの数であり，単体のプロセッサはスーパーコンピュータのものであってもパーソナルコンピュータのものであっても大きな差はない。したがってここで述べる内容はスーパーコンピュータのみならず，パーソナルコンピュータも含め，現在入手できるほとんどのコンピュータに当てはまるものである。

2.1 計算機の構成と動作原理

計算機(コンピュータ)はプロセッサ，メモリ，入出力機器，およびそれらを接続するためのチップセットから成る。これらは図 1 のように接続されている。

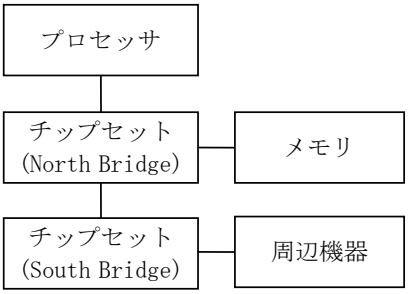

図 1　計算機の基本構成

まずプロセッサと直接接続されているのが North Bridge と呼ばれるチップセットであり，ここにはメモリコントローラと呼ばれるメモリアクセスのための機構が実装されている。North Bridge は伝統的に

はプロセッサとは別のチップであったが，近年では高速化のためにこの機能をプロセッサの中に内蔵しているものもある。North Bridge の先には South Bridge と呼ばれるチップセットが接続されている。South Bridge の主な役割は周辺機器との通信の窓口となることである。ディスク，ネットワーク，USB，拡張スロットに搭載する様々なボードはすべて South Bridge に接続されている。このように計算機は高速なプロセッサとメモリが一番近くに配置され，周りに低速な周辺機器が接続される形で構成されている。

計算機の動作をごく簡単に述べると，プロセッサは周辺機器であるディスクから読み込んだプログラムをメモリに書き，そのプログラムを順番に読み込み，実行することで動作している。また，計算対象のデータもディスクに書かれており，それをプロセッサがメモリにコピーし，さらにプロセッサ内部に読み込み，計算し，結果をまたメモリに書き出すことでデータが処理されている。つまり計算機の動作は大部分が「プロセッサ内での演算」または「メモリおよび周辺機器とのデータ交換」のどちらかである。したがってプロセッサの性能を最大限引き出すためには，演算効率とメモリアクセスの効率を上げることが重要となる。次節以降では演算性能に関わる事項として「パイプライン」，メモリ性能に関わる事項として「キャッシュメモリ」および「TLB」について解説する。

2.2 パイプライン

前述のように計算機はプロセッサがメモリに書かれた命令列を読み込み，指示された操作（演算やメモリ入出力）を繰り返すことで動作している。これをもう少し細かく見るとプロセッサは主に以下の手順で命令を実行している。

1. フェッチ（F）
   メモリから命令を読み込む
2. デコード（D）
   読み込まれた命令を解釈する
3. 読み込み（R）
   命令実行に必要な被演算数を読み込む
4. 実行（X）
   命令を実行する
5. 書き込み（W）
   実行結果をレジスタに書き込む

これらのステップはそれぞれ独立した回路で行われるため，ある命令がこれらのステップを完全に完了しなくても，次の命令の実行を始めることができる。したがって，プロセッサは下の図 2 のように複数の命令を同時に処理している。

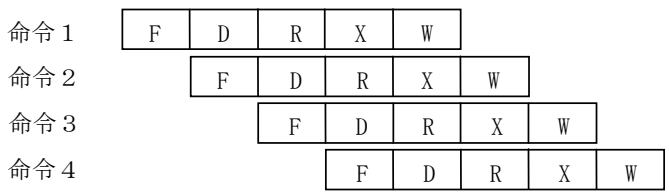

図 2 パイプライン

各命令が完全に独立ならばパイプライン実行は効率的であるが，実際にはプログラムは前の命令の実

行結果を使って次の計算を行うことが多い。このような場合，前の命令の実行が終わるまで次の命令を実行できないため，下図 3 のような待ちが発生する。このような実行待ちを「パイプラインハザード」と呼ぶ。

図 3　パイプラインハザード

パイプラインハザードは一般的に以下の場合に生じる。

1. 演算の被演算数（オペランド）に依存関係がある場合
   前の命令の結果がないと次の命令が実行できない
2. 条件分岐の場合
   前の命令が終わらないと，次にどの命令をフェッチするのかがわからない
3. 資源に競合がある場合
   例えば，メモリアクセス命令を連続して発行しても，メモリ操作には時間がかかるため待ちが発生してしまう

ここでは解説のため命令実行は 5 ステップとしたが，近年実際に使用されているプロセッサはこれよりはるかに複雑であり，20ステージ近くまで細かく分割しているプロセッサもある。このようにパイプラインの深いプロセッサではハザードが起きたときの性能ロスも大きくなる。このようなロスを避けるため，プロセッサには依存関係のない命令を探し出して順番を入れ替えて実行するなどの仕組みが実装されている。しかし効率的な実行のためにはソフトウェアの最適化も必要不可欠である。

スパコンプログラミングのためのヒント

プロセッサ性能を極限まで引き出すにはプロセッサメーカーが提供する最適化マニュアルなどを参照しながらアセンブリレベルでプログラムをチューニングする。しかしこれには専門的知識と長い時間を要するため，多くのスパコンユーザーにとって現実的な作業ではない。したがって，最適化はコンパイラとチューニングされた専用ライブラリに任せ，自ら記述する部分に関してはごく基本的なことのみ注意するのが現実的である。具体的な留意点としては以下の3点が挙げられる。

- 条件分岐やジャンプは高コストであることを意識する
  ループの最内周など，実行回数が非常に多い部分に細かい条件分岐や短い関数の呼び出しを入れるのはなるだけ避ける
- 基本的な計算は最適化されたライブラリに任せる
  後述のキャッシュ効率の観点からも重要である
- コンパイルの際には適切な最適化オプションを付けることを忘れない
  多くのシステムではデフォルトでは最適化オプションは付いていない

2.3 キャッシュメモリ

メモリに代表される記憶装置には，性能と容量にトレードオフの関係がある。つまり，大容量の記憶装置は低速で，高速な装置は容量が小さくなる。一方，一般的にアプリケーションプログラムのメモリアクセスには局所性があり，一度アクセスされたメモリは近い将来再びアクセスされるか付近をアクセスされ

る可能性が高いとされている。キャッシュメモリはこの性質を利用して，一度アクセスされたメモリ領域（通常 64 バイト程度のブロック単位で管理される）を高速メモリに格納することにより，続くアクセスを高速化する技術である。計算機の記憶装置は以下のような階層を持つ。

1. L1 キャッシュメモリ
   2,3 クロックでアクセスできる超高速メモリであり，プロセッサ内部に 64KB 程度存在する。
2. L2 キャッシュメモリ
   L1 よりも低速で 10 クロック程度の遅延があるが，プロセッサ内部に 512KB 程度存在する。
3. L3 キャッシュメモリ
   プロセッサの種類によって，存在しないもの，プロセッサ内部に持つもの，外部に持つもの様々であり，存在する場合も 1MB から 32MB 程度と幅がある。50 クロック程度の遅延のものが一般的だが，状況によって異なるものもあり，プロセッサによるばらつきも大きい。
4. メインメモリ
   プロセッサあたり数 GB の容量を持つ主記憶である。数百から千クロック程度のレイテンシがある。
5. スワップ
   メインメモリが足りないときにハードディスクなどにデータを書き出す。これはプロセッサの機能ではなくオペレーティングシステムの機能であるが，あまりにも低速でスーパーコンピュータでの使用には耐えないので，スーパーコンピュータではこの機能は OFF にされることが多い。

このように，どのメモリを使用するかによって性能は大きく異なる。効率的なプログラム実行のためには，なるだけキャッシュ上のメモリを使って計算を行うことが重要である。

スパコンプログラミングのためのヒント

キャッシュメモリに格納するデータの取捨選択やキャッシュメモリとメインメモリの一貫性の維持など，キャッシュメモリの管理はすべてハードウェアによって制御されている。そのため，プログラム上はキャッシュの存在を無視しても動作の正しさには影響しない。これは便利な性質ではあるが，プログラムによる制御ができないため最適化にはキャッシュの振る舞いを予測しながらの高度なプログラミングが求められることにもなる。キャッシュを効率的に使用する手法には様々なものがあり，本稿が扱う範囲を超えてしまうので本特集号の別記事を参照いただきたい。ただし，少なくとも以下の 2 点は意識してプログラミングを行うべきである。

- メモリはなるだけ連続アクセスする
  ランダムにアクセスすると，キャッシュされていないメモリをアクセスする可能性が高いので性能が低下する。計算アルゴリズムの工夫が重要であるが，それ以前の問題として有名なミスは多次元配列のアクセス順序を間違えることである。特に FORTRAN と C 言語では多次元配列のメモリの配置が異なるので注意が必要である。またプログラム中のジャンプ命令（特に関数ポインタを用いたジャンプ）は，パイプラインの効率にも影響を与えるばかりでなく，命令メモリのキャッシュミスにつながることがあるという点でも注意が必要である。
- 基本的な計算はなるだけライブラリに任せる
  数値演算ライブラリはキャッシュ効率も考慮した上で最高の性能で計算できるよう最適化されている。これらは高級言語で記述した数学的に等価なプログラムよりも高速であることが多い。

2.4 TLB

アプリケーションプログラムがメモリアクセスの際に使用するメモリアドレスは，アプリケーション固有のメモリ空間におけるアドレス（仮想アドレス）であり，物理的なメモリ内の位置を示すアドレス（物理アドレス）とは異なる。この仮想アドレスと物理アドレスの対応表はオペレーティングシステムがメインメモリ上に作成する。アプリケーションがメモリアクセス命令を発行した際には，プロセッサが自動的にその変換表を参照して物理アドレスに変換し実際のメモリにアクセスする。このアドレス変換は「ページ」と呼ばれる一定量の連続領域を単位にして行われる。例えば x86 系のアーキテクチャでは通常，ページサイズは 4KB である。キャッシュの項でも述べたようにメインメモリへのアクセスには時間がかかるため，メインメモリ上の変換表をメモリアクセスの度に参照するのは非効率である。そこでこの変換表にも TLB と呼ばれるキャッシュに似た機構が準備されている。メモリアクセスの際にアドレスが TLB に存在していれば遅延の短いアクセスが可能となる。

スパコンプログラミングのためのヒント

キャッシュとは異なり TLB は1エントリで 4KB もの領域をカバーできるため，普通にプログラムを書けば TLB はほぼヒットすると考えてよい。ただしあまりにもアドレスの離れたランダムアクセスや，多次元配列のアクセス順序の間違いを犯すと TLB ミスが多発する可能性がある。また，プログラム開始直後はすべてのページで TLB ミスが起きる。ただしプログラム開始直後に関しては TLB ミスのペナルティーよりもオペレーティングシステムのデマンドページングの処理に要する時間の方が問題であるため，詳細はオペレーティングシステムの章で述べる。

3. スーパーコンピュータの構成方式

並列計算機は多数のプロセッサを搭載した計算機であり，近年のスーパーコンピュータは例外なく並列計算機である。並列計算機はプロセッサを接続する方式の違いにより「共有メモリ型並列計算機」と「分散メモリ型並列計算機」に分類される。本章では両方式を紹介した上で，近年のスーパーコンピュータの構成方式について述べる。

3.1 共有メモリ型並列計算機

共有メモリ型並列計算機とは，その名の通りすべてのプロセッサが同一のメモリを共有する並列計算機である。この構成の並列計算機には主に SMP と ccNUMA が存在する。

3.1.1 SMP 方式

SMP は Symmetric Multi Processing の頭文字である。これはすべてのプロセッサが対等な並列計算機であり，理想的な SMP ではどのプロセッサからどのメモリをアクセスしても同じ性能であり，周辺機器との通信もすべてのプロセッサが均等に処理を行う。図 4 に SMP 構成の概略を示す。

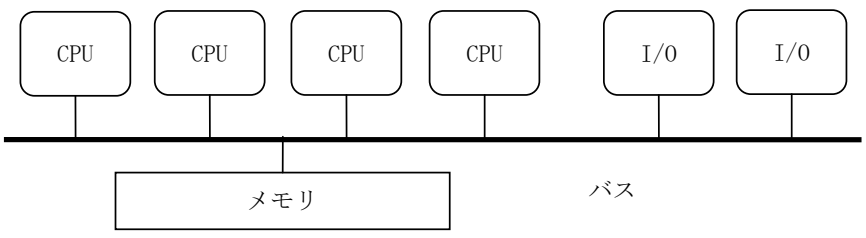

図 4 SMP

すべてのプロセッサや I/O(周辺機器)は1本のバスに接続されており，メモリアクセスはこのバスを通じて行われる。すべてのプロセッサと I/O が1本のバスにつながっているため，どのプロセッサからどのメモリをアクセスしても性能は均一であり，すべてのプロセッサが I/O に対しても対等の関係にある。

キャッシュメモリの存在は共有メモリ型並列計算機にとっては大きな問題である。各プロセッサは個別にキャッシュメモリを持つため，複数のプロセッサが同じ領域をキャッシュする場合があり，これらが矛盾しないようにしなくてはならない(キャッシュの一貫性保持)。SMP 方式の場合，各プロセッサはバスに流れる制御信号を常に監視し，他のプロセッサや I/O によって自身がキャッシュしているメモリ領域が更新された際にはキャッシュの内容を破棄する処理を行っている。

SMP 方式はメモリの物理的な場所をソフトウェアが意識する必要がないため扱いやすい。しかし，バス性能の限界やキャッシュの一貫性保持のためのコストが高く，大規模な SMP 型計算機を作るのは困難である。

3.1.2 ccNUMA 方式

NUMA とは Non-Uniform Memory Access であり，cc は cache coherent を意味する。つまりプロセッサとメモリの位置関係によって性能や振る舞いが異なるが，キャッシュの一貫性は保たれている共有メモリ型並列計算機である。図 5 に ccNUMA 方式の概略図を示す。

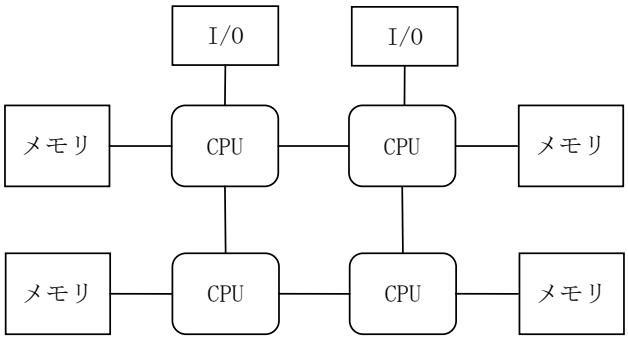

図 5 NUMA

図のような構成の場合，直結されているメモリには高速にアクセスできるが，他の CPU に接続されているメモリにアクセスする場合は遅延が大きくなる。また，周辺機器から見たメモリ性能も均一ではない。ソフトウェアの立場から見るとccNUMA構成の並列計算機はSMPよりは扱いづらい。しかしハードウェア作成における制限が SMP よりは緩和されるため，大規模な計算機の作成が可能であり，最適化されたプログラムでは SMP よりも高性能であることが多い。

ccNUMA は機能的には SMP と同じである。つまり，すべてのプロセッサからすべてのメモリのアクセスが可能でありキャッシュの一貫性も保持される。したがって基本的にはSMP用のソフトウェアがそのまま使用できる。ただし，最高の性能を得るためにはなるだけ近くのメモリにアクセスすることが必要である。近年では多くのオペレーティングシステムが NUMA の特性を考慮したメモリ配置を行っている。

3.2 分散メモリ型並列計算機

分散メモリ型並列計算機は複数の計算機をネットワークで接続した並列計算機である。構成要素となる個々の計算機は「ノード」と呼ばれ，非常に限定された機能のみを持つ小型計算機である場合や，前述したような共有メモリ型の並列計算機である場合などがある。また，ノード間のネットワークはインタ

ーネットなどで一般的に使用されているネットワークとは違い，高速な専用ネットワークであることが多い。図 6 に分散メモリ型並列計算機の概略を示す。

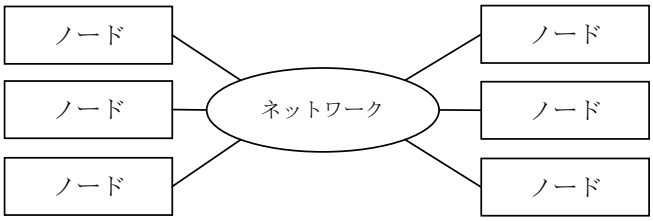

図 6　分散メモリ型並列計算機

分散メモリ型並列計算機はそれぞれのノードのメモリが独立しているため，プロセッサから他のノードのメモリを直接アクセスすることはできない。したがってノードにまたがる協調動作のためにはアプリケーションプログラムが明示的な通信を行う必要があり，共有メモリ型並列計算機のプログラムをそのまま使用することはできない。このように分散メモリ型並列計算機はアプリケーションに通信を記述しなくてはならないという欠点を持つが，共有メモリ型よりも安価であり，また共有メモリ型では不可能な超大型計算機が作成可能であるという利点を持つ。

3.3 近年の並列計算機

共有メモリ型並列計算機はメモリアクセスの度にキャッシュの一貫性保持のための通信が必要である。プロセッサ数が増加するにつれて通信量も増加するため，共有メモリ型並列計算機では規模に比例した性能向上は見込めない。そのため大規模化には限界があり，実際，市場に存在する最大の共有メモリ型並列計算機は，SMP で 100 プロセッサ，ccNUMA で 1000 プロセッサ程度である。このような共有メモリ型の限界のため，現在の大型計算機の主流は分散メモリ型である。

一方，分散メモリ型並列計算機を構成するノードに目を向けると，プロセッサのマルチコア化の流れを受けて，ノードが小規模な共有メモリ型並列計算機を構成していることが多くなってきている。現在のスーパーコンピュータでは，マルチコアのプロセッサを数個搭載した 4 から16プロセッサ程度の共有メモリ型並列計算機をノードとして使用するのが主流である。

スパコンプログラミングのためのヒント

現在の主流である分散メモリ型並列計算機を使用するためには，必要な通信をアプリケーション内に明示的に記述する必要がある。この通信の記述には MPI と呼ばれる通信ライブラリを使用するのが一般的である。MPI でアプリケーションを記述すれば，並列計算機が使用するネットワークに依存しないインタフェースで通信が記述できるため，構成の異なる並列計算機に移行する際も，再コンパイルするのみで動く可能性が高い。また，MPI は分散メモリ型並列計算機向けの通信規格であるが，共有メモリ型並列計算機でもメモリを介して通信を行うことで MPI を使用できる。

一方，各ノードは共有メモリ型並列計算機であるので，ノード内では共有メモリを使用した並列化が可能である。ノード内での並列化には自動並列化または OpenMP が使用されることが多い。自動並列化はコンパイラが自動的に依存関係のない複数の計算を検知して，それぞれの計算を個別のプロセッサに割り付ける方法である。プログラマの負担は一切ないため最も簡単な方法であるが，コンパイラによる並列性の自動検出には限界があり，性能がよくないことがある。OpenMP は並列化して計算できる部分をユーザーが明示的に指定するための規格である。並列化可能なループに OpenMP 指示文を

追加すると，コンパイラはそれにしたがって並列化したコードを生成する。自動並列化や OpenMP は共有メモリ型並列計算機向けの仕組みであり，原則として分散メモリ環境では使用できない。

以上をまとめると，並列アプリケーションの作成方法は以下の2通りとなる。

1．ノード内を自動並列化または OpenMP，ノード間を MPI で並列化する

自動並列化の場合は MPI プログラムを自動並列化オプション付きでコンパイルするのみである。OpenMP の場合は MPI による通信を書いてはならない場所があるので注意が必要である。

2．全体を MPI で並列化する

共有メモリ型並列計算機内でも MPI は使用できるので，全体を MPI で並列化してもよい。

残念ながら，いずれにしても MPI による通信の記述は避けられないのが現実である。したがって新たにアプリケーションを作成する際は，まず MPI で通信を記述することによる並列化を行うべきである。それをそのまま実行すれば2の方式となり，OpenMP 指示文などを追加してノード内並列化を最適化すれば1の方式となる。一般的に1の方が高速だとされているが，アプリケーションによっては2の方が高速なこともある。

3.4 メモリバンド幅

ここまでスーパーコンピュータの構成方式を述べてきたが，最後にまとめとして，スーパーコンピュータの構成を考える上で非常に重要なメモリバンド幅について考えてみる。

スーパーコンピュータを数値演算に使用する場合，メモリの性能（メモリバンド幅）はアプリケーション性能を左右する重要な要素である。例えば代表的な数値計算ライブラリである BLAS の中には $\vec{y} = \vec{y} + a * \vec{x}$ を行う DAXPY と呼ばれるサブルーチンがある。この演算では2回の浮動小数点演算に対して3回のメモリアクセスが発生する。DAXPY では浮動小数点数は 8 byte なので，一回の浮動小数点演算で発生するメモリアクセスは 3 x 8 / 2 = 12 Byte である。これを 12 Byte/Flop と言う。一方，実際のプロセッサの例として最新の Quad Core Opteron プロセッサを例に挙げると，このプロセッサは1クロックで 4 回の浮動小数点演算ができるため，2.3GHz 動作の Quad Core プロセッサでは 1 秒あたり 2.3G x 4 x 4 = 36.8G 回の計算ができる。このプロセッサと共に一般的に使われるメモリの性能は 10.6GB/s なので 10.6 / 36.8 = 0.28 Byte/Flop ということになる。これは DAXPY で求められる 2.3% でしかない。つまり，キャッシュの効かない巨大なベクトルでこの演算を行った場合，メモリバンド幅が律速となって CPU の理論性能の 2.3%の性能しか得られないこととなる。これは非常に極端な例であり，行列積計算のようにキャッシュの利用効率が高く，理論性能の 90% 近くが得られる計算も存在するが，DAXPY はメモリバンド幅の重要性，プロセッサの周波数やコアの数をのみ求める無意味さを示す好例である。

本章で述べてきた並列計算機の構成方式もメモリバンド幅と深い関係がある。各方式の説明でも触れたが，SMP 方式はキャッシュの一貫性保持のためのコストやバスのバンド幅が律速となりやすくメモリバンド幅を確保するのが困難である。ccNUMA であればバスが律速となることがなくなるため，SMP よりはメモリバンド幅は確保しやすくなるが，キャッシュ一貫性保持のためのコストは依然として残るため，プロセッサ数に比例してメモリバンド幅が増えるわけではない。その点，分散メモリ方式であれば計算機の数を増やせばそれに比例して総合的なメモリバンド幅も増加する。なお本稿では触れないが，高いメモリバンド幅を持つ計算機の代表はベクトル型スーパーコンピュータである。近年では並列化された演算器を作るのは難しいことではないため，ベクトル型の最大の利点はベクトルであることよりもスカラー型を圧倒的に上回るメモリバンド幅であると言える。

4. オペレーティングシステム

本章ではアプリケーションの実行をサポートするオペレーティングシステムについて述べる。オペレーティングシステムとは，アプリケーションに対する資源の割り当てを決定したり，ディスク入出力をサポートするためのソフトウェアであり，近年の計算機ではほぼ例外なく使用される基本ソフトウェアである。共有メモリ型並列計算機では計算機一台につき一個のオペレーティングシステムが，分散メモリ型並列計算機では各ノードで一個のオペレーティングシステムが動作している。

本章では動作中のアプリケーションを表現するための概念であるプロセスとスレッドについて紹介した後，アプリケーションがオペレーティングシステムの機能を使用する際に使用するシステムコールについて述べる。そして最後にメモリ管理の仕組みとして重要な概念であるページングについて紹介する。

4.1 プロセス，スレッド

プロセスとは実行中のプログラムのことである。同一のプログラムであっても複数走らせれば複数プロセスになる。プロセスは独立したメモリ空間を持つため，他のプロセスのメモリを直接アクセスすることはできない。スレッドはプログラム実行の最小単位であり，プロセッサの割り当てはスレッド単位で行われる。一個のプロセスは複数のスレッドを持つことができ，同じプロセスに属すスレッドはすべて同一のメモリ空間で動く。下図 7は同一のプログラムを2個走らせた状況を示しており，それぞれのプロセスが3個のスレッドを持つ。図ではそれぞれのスレッドが別の関数を実行しているが，これは同じであっても構わない。

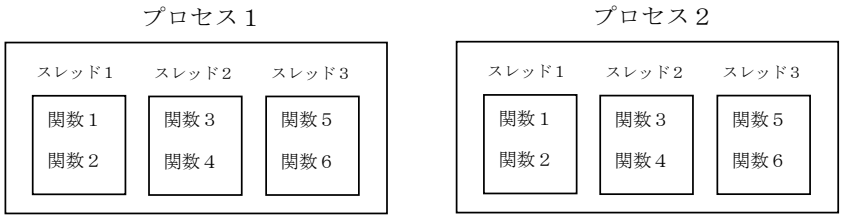

図 7　プロセスとスレッド

共有メモリ型並列計算機上では，複数のスレッドを同時実行することでも複数のプロセスを同時実行することでも並列計算は可能である。ただし，複数プロセスによる並列化の場合は，明示的にデータ共有の仕組みを作らなくてはならない。これはほとんど分散メモリ型並列計算機を使用しているのと同じ状態であり，実際，一台の共有メモリ型並列計算機の中で複数のプロセスを使った並列化を行う際には，MPI で通信を記述するのが一般的である。共有メモリ型並列計算機の利点は，メモリを共有しながら複数のプロセッサを同時に使用できる点であり，この利点を生かすためにはスレッドを用いるのが最適である。前述した OpenMP による並列化を行った場合，または自動並列化を使用した場合は自動的にスレッドが使用される。

スパコンプログラミングのためのヒント

スレッドとプロセスの使い分けに関しては，メモリ共有の有無の他，スレッドはプロセスよりも生成，消滅に関わるオーバヘッドが低いというような性質も考慮して適切な方を選択する。これはサーバアプリケーションなどを作成するときには重要なポイントであるが，並列計算にあいては，MPI を使用すればプロセス，OpenMP や自動並列化を使えばスレッドが自動的に選択される。このためスパコンユーザー

がこれらの性質を考慮する必要はない。

スパコンユーザーが注意するべき点は，スレッドを使った際の資源競合である。スレッドの場合はすべてのスレッドがメモリ空間を共有するため，グローバル変数や動的に確保したメモリ領域は複数のスレッドから同時にアクセスされる可能性がある。複数スレッドが同じメモリ領域を同時にアクセスすると，普通では起こりえない矛盾した状態のデータが書き込まれる危険性がある。例えばグローバル変数の値に1を加える処理を2スレッドが同時に行う場合を考えると，それぞれのスレッドが同じ値をメモリから読み込み，1を加算してメモリに書き戻すため，結果として1しか加算されない（同時実行でなければ2増加したはずである）。OpenMP による並列化を行う場合，このような矛盾した状況を引き起こさずに並列化できるかどうかを判断するのはユーザーの責任である。例えばグローバル変数への書き込みがあるようなループを並列化する際には安全性の検討を十分に行う必要がある。

4.2 システムコール

プロセスやスレッドの生成やディスク入出力など，アプリケーションがオペレーティングシステムの機能を使用する場合はシステムコールと呼ばれる方法を使ってオペレーティングシステムに処理の要求を出す。システムコールはソフトウェアによって明示的に発生させる例外であり，例外が発生するとプロセッサは特権モードと呼ばれる権限の高いモードに移行した上で，オペレーティングシステムの決められた部分（例外ハンドラ）へと処理を移行する。この例外ハンドラにおいてオペレーティングシステムはユーザープログラムの要求を受け取り，必要な処理を行った上で例外ハンドラの終了をプロセッサに伝える。例外ハンドラが終了するとプロセッサは権限をユーザープログラムの権限に戻した上で，例外を発生させた次の命令より実行を再開する。

スパコンにおけるプログラムはほとんどが数値計算であるため，このようなシステムコールが発行される頻度は低い。しかし，ファイル入出力など，一部のシステムコールはスパコンでも頻繁に使用されるので，システムコールの正しい使用方法を知ることは重要である。

スパコンプログラミングのためのヒント

C 言語から使用する場合，システムコールは通常のライブラリ関数同じように使用できる。例えばファイルにデータを書き出すシステムコールは write 関数であり，この関数は特別な準備をすることなく C 言語のプログラム中で使用することができる。しかし，見た目は単なるライブラリ関数であっても，前述のようにシステムコールは例外を発生させる。例外が発生するとプロセッサはプログラム実行の強制中断に伴う状態の保存や特権モードの切り替えなど，様々な作業を行わなければならない。したがってこれには単なる関数呼び出しとは比較にならないほどの長い時間を要する。

一般にスパコンプログラムの中でシステムコールを直接呼び出すことはなるだけ避けた方がよい。例えばファイル入出力の場合，システムコールを効率的に使用するライブラリ群が準備されているので（通常使用する fread，fwrite がそのライブラリである）そちらを使う方がよい。特殊なファイルシステムなどでライブラリ関数が効率的に動かない場合はシステムコールを直接使用することになるが，その場合はシステムコールのコストやファイルシステムの特性などを十分理解した上でプログラムを作成する必要がある。

4.3 ページング

プロセッサアーキテクチャの節で述べたように，アプリケーションプログラムがメモリアクセスの際に使用するアドレスは仮想アドレスであり，物理的なメモリの位置を示すアドレス（物理アドレス）とは異なる。この仮想アドレスと物理アドレスの変換は4KB程度の「ページ」と呼ばれる単位で行われており，この変換表を「ページテーブル」と呼ぶ。ページテーブルの作成はオペレーティングシステムの役割である。

実は，C 言語のグローバル変数や FORTRAN の配列はプログラム開始時には物理アドレスが割り当てられていない。また C 言語の malloc のように動的に割り当てられるメモリ領域も，malloc 関数が終了した時点ではまだ物理的なメモリは割り当てられていない。これらは実際にアクセスされて初めて物理的なメモリが割り当てられる。このメモリ割り当ての仕組みについて簡単に解説する。

プロセスの項で解説したように，プロセス生成時にはそのプロセス固有のメモリ空間が割り当てられる。これは仮想アドレスの空間であり，この仮想アドレスに対応する物理アドレスは未知のまま実行が始まる。つまりこの段階ではこのプロセス用のページテーブルは空の状態である。プログラムの実行が始まりメモリへのアクセスが発生すると，プロセッサはそのメモリアドレスを物理アドレスに変換しようとページテーブルを参照する。しかし，ページテーブルは空であるため，アドレス変換は失敗する。するとこれは例外となり，プログラム実行は一時停止されオペレーティングシステムへと処理が移る（システムコールと似たような動作になる）。ここで初めてオペレーティングシステムが空いている物理メモリ領域を割り当て，ページテーブルに物理アドレスをセットする。この例外処理が終了すると，実行はアプリケーションプログラムに戻される。システムコールと異なり，メモリ関連の例外はハンドラから戻った直後に再実行されることになっているため，先ほど失敗したメモリアクセス命令は再び実行される。今度はプロセッサが正しいページテーブルを見つけることができるため，このメモリアクセスは成功する。malloc による動的割り当ての場合も動作は同様である。malloc は有効な仮想アドレスの割り当てをオペレーティングシステムに依頼しているのみであり，実際のメモリ割り当てを行っているわけではない。C の入門書などでは malloc に失敗するのはメモリ不足の場合であると解説されているが，これは仮想アドレス空間の不足を意味しており，近年の 64bit アーキテクチャで仮想アドレス空間の不足が発生する可能性はほぼゼロである。したがって，実際に物理的なメモリ不足に陥った場合は malloc が NULL で終了するのではなく，実行時エラーによる強制終了という形でユーザーに報告される。

このように，実際にアクセスされて初めて物理メモリを割り当てる方式を「デマンドページング」と呼ぶ。この方式は実際に使用されない領域にメモリを割り当ててしまう無駄を避けるため，今日ではほとんどのオペレーティングシステムに実装されている。

スパコンプログラミングのためのヒント

デマンドページングはその存在を知らずにプログラミングをしても動作の正しさには影響しない。気をつけるべき点は，本当のメモリ不足は malloc の返値が NULL でないことを見るだけでは判別できないことのみである。ただ，性能評価の際にはデマンドページングを意識する必要がある。特にプログラム開始直後はすべてのページにおいて例外が発生していることに気をつける必要がある。例えばループの一周目にかかる時間を測定して全体の性能を予測したとしてもおそらく正しい値は得られない。ループの一周目ではデマンドページングの処理が行われているため，二周目以降は大幅に高速化されるからである（もちろん TLB やキャッシュの影響もある）。また，性能測定の際にも注意が必要であり，性能測定のために取得した時刻をメモリに書く際に例外を起こすと，その分のペナルティーが計測結果に含まれてしまう。

5. 通信

本章では分散メモリ型並列計算機の重要な構成要素である通信機構について述べる。並列計算機における通信は通常はインターコネクトと呼ばれる専用ネットワークが使用される。本章の最初はこのインターコネクトについて特徴と性能を紹介する。続いて通信の仕組みについて述べた後，最後に通信性能がアプリケーション全体の性能に及ぼす影響について簡単なデータを紹介する。

5.1 インターコネクト

分散メモリ型並列計算機のノード間接続には様々な高速ネットワークが使用されている。これらは広義にはネットワークであるが，短距離を高バンド幅で接続する意味で「インターコネクト」と呼ぶ場合が多い。並列計算機メーカーはそれぞれ固有のインターコネクトを開発しており，10GB/s 以上のバンド幅を持つ非常に高速なものもある。一方で特定の並列計算機専用ではなく，一般的な PC アーキテクチャの計算機で使用できるインターコネクトも開発されており，Infiniband または Myrinet が広く使われている。これらは単体でそれぞれ 2.0GB/s，1.25GB/s であるが，複数リンクを同時に使用することで，5GB/s 程度の性能は実現可能である。さらに，現在発売されているほとんどの計算機に標準で搭載されている Ethernet もインターコネクトとして使用することがある。ただし，PC サーバに内蔵されているような一般的な Ethernet をインターコネクトとして使用する場合は，プロセッサ負荷，遅延，大規模ネットワークでのバンド幅の点で専用インターコネクトに劣る点が多い。したがって Ethernet をインターコネクトとして使用するのは主にコスト的メリットを求める場合である。

5.2 通信の基本性能

通信性能を示すために最もよく用いられるのは通信バンド幅である。バンド幅は bps(bits per second) または MB/s などで表され，1 秒間に転送できるデータ量を意味する。ここで注意が必要なのは，カタログスペックとして示されるバンド幅は設計上の限界値である点であり，実際に得られるバンド幅は転送するデータのサイズに依存する。

実際の通信性能の傾向を示すために，例として Myrinet-10G と呼ばれるインターコネクトを用いた並列計算機で，MPI を使用した通信性能測定プログラムを実行した結果を図 8 に示す。結果は横軸に一度に転送したデータのサイズ，縦軸にバンド幅が示されている。

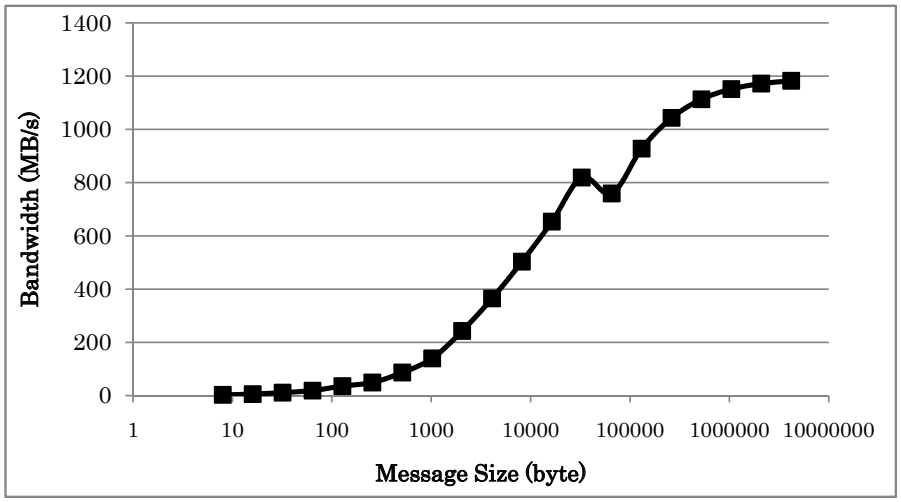

図 8 基本通信性能

測定に用いた Myrinet-10G は理論性能が 10Gbps のインターコネクトであるため，最高性能が 1250MB/s である。グラフの一番右の点はメッセージサイズが 4MB の場合で，そのときの性能は 1188MB/s である。これは理論性能の 95%程度であり，実際に得られる性能としては妥当な値である。

グラフからわかるように，理論性能に近い性能を得るためには 4MB 程度のメッセージサイズが必要であり，小さいメッセージではネットワークの性能を十分に生かすことはできない。これは通信に必要な準備作業の影響であり，小さなメッセージではこの準備作業に要する時間が全体通信時間の大部分を占めるため，ネットワークの物理性能が生かせない。したがって，高い通信性能のためには通信準備にかかる時間が短いことも重要である。上記の測定で用いた Myrinet-10G の場合，4byte のメッセージを送信するのに要する時間は 2.8 マイクロ秒程度である。同じ計算機に搭載されている 1Gbps の Ethernet を使用した場合は 50 マイクロ秒程度の時間がかかるので，インターコネクトとして設計された Myrinet-10G は非常に高性能であることがわかる。

スパコンプログラミングのためのヒント

通信は，なるだけまとめて行うようにする方が効率的である。また，MPI のタイプを使用するときも注意が必要である。MPI で規定されているユーザー定義型を使用すると，非連続領域を一回の通信操作で送受信できる。これはプログラムの見た目はシンプルであるが，実際には細かい非連続領域の集まりなので，小さなメッセージの大量送信になる可能性がある。MPI の実装によっては一度大きな内部バッファにデータを集める可能性もあるが，いずれにしても連続領域を通信するよりは時間のかかる作業となるので，MPI プログラム上で 1 行であることは必ずしも高速に通信できることにはつながらない。

5.4 通信の仕組み

前述したようにインターコネクトには様々なものがあり，種類によって通信の仕組みは異なる。また，メーカー独自のものは動作原理が公開されていないことも多い。したがってここでは PC アーキテクチャで使用されているインターコネクトを例に通信の仕組みを述べる。具体的なインターコネクトとしては Infiniband または Myrinet が該当する。本稿で述べる範囲においては両者に大きな差はない。

5.4.1 DMA 転送と見えないバッファ

周辺機器はプロセッサの関与しないところでメモリと直接データのやりとりができる。これを DMA(Direct Memory Access)と呼ぶ。ネットワークによる通信はネットワークインタフェースカードとメモリの間で DMA 転送をすることにより，プロセッサへの負荷を最小限に抑えながら通信を行う。

DMA 転送を行う際にはネットワークカードからメモリに対して読み込みまたは書き込み要求を発行する。周辺機器にはプロセッサのページテーブルに相当するものは存在しないため DMA 転送は物理アドレスを用いて行われる（厳密には「バスアドレス」であるがここでは物理アドレスと同一視する）。一方，送受信対象となるデータは各プロセスのメモリ空間に存在するため，DMA 転送のためには転送対象のデータが置かれているメモリ領域の仮想アドレスを物理アドレスに変換した上でネットワークカードに知らせる必要がある。オペレーティングシステムの章で述べたように，仮想アドレスと物理アドレスの変換表（ページテーブル）はオペレーティングシステムが作成しているため，オペレーティングシステムに問い合わせればこのアドレス変換は可能である。しかしこうしてアドレス変換を行うだけでは実は危険な場合がある。それは DMA 転送中にオペレーティングシステムが実行プロセスの切り替えを行い，別のプロセスにそのメモリ領域を割り当ててしまった場合である。このような場合でもネットワークカードは

知らずに DMA 転送を続けるため，最悪の場合はネットワークを使用しているプロセスとは関係のないプロセスのデータを送信してしまうことがあり，正しい計算が行われないばかりか無関係なプロセスのデータ破壊や情報漏洩の危険も生じる。

このような危険な状況を避けるため，通信ライブラリは転送に使用するメモリ領域をあらかじめ一定量を確保した上で，オペレーティングシステムに物理アドレスと仮想アドレスの対応を決して変更しないよう依頼している。これをメモリレジストレーションと呼ぶ（ピンダウン，メモリロックなど他の呼び方もある）。結局，DMA 転送はハードウェア的には任意のメモリアドレスに対して実行可能であるものの，実質的にはメモリレジストレーションが完了している領域に対してしか使用できない。

このような制約のため，送受信はメモリレジストレーションが完了しているユーザーには見えないバッファを介して行われている。具体的には，送信の際には通信ライブラリが送信対象のデータをまずメモリレジストレーションされた領域にコピーした上でネットワークカードに送信依頼を出している。また受信データはネットワークカードがメモリレジストレーションされた領域に DMA を行った後，通信ライブラリがその領域からユーザーが希望する受信バッファにコピーしている。

スパコンプログラミングのためのヒント

ユーザーが気をつけるべき点は，通信ライブラリの中にメモリレジストレーション用の見えないバッファ領域が存在している点である。高い通信性能のためにはメモリレジストレーションをするバッファは無視できない程度のサイズが必要であり，並列計算機の規模とネットワークの種類によってはこのバッファだけで 1GB 近くのメモリを使用することがある。したがって，アプリケーションが計算用に使用できるメモリ量は，各ノードの物理メモリ量からオペレーティングシステムなどのシステム領域を減じ，さらに通信バッファの量を減じた残りの部分である。具体的な数字はシステム管理者から目安が示されることが多い。

5.4.1 ゼロコピー通信

前述のように DMA 転送はメモリレジストレーションされた領域に対してしか実行できないため，送受信データは一度送受信用のバッファにコピーされる。ところがユーザーが非常に大きな領域に対して送受信命令を発行した場合は，あらかじめレジストレーションされた領域にコピーをするよりも，ユーザーから指定されたバッファを新たにレジストレーションし，そこから DMA 転送を行った方が高速な場合がある。このような通信をゼロコピー通信と呼ぶ（送受信バッファへのコピーがないのでゼロコピーと呼ばれる）。メモリレジストレーションはコストのかかる操作であるため，コピーをなくすメリットがメモリレジストレーションのコストを上回らなければならない。ゼロコピー通信の方が高速になるメッセージサイズは計算機によって異なるが，現在の計算機では 32KB 程度が境目になることが多い。

実は，通信ライブラリはメッセージサイズによって前節で解説したコピーを伴う通信とゼロコピー通信を使い分けている。通信性能の章で示した性能のグラフ（図 8）で，メッセージサイズを増やしているにもかかわらず性能が下がっている点があるが，これはこの通信方法の切り替えのためである。図 8 の場合は，ゼロコピー通信のメリットが出始めると判断したサイズが小さすぎたため，性能が落ち込む点が出てしまっている。適切に管理されているスーパーコンピュータの場合は，このような切り替え点は最適な点に設定されているため図 8 のような性能カーブにはならず，単調増加になるはずである。

スパコンプログラミングのためのヒント

一度に通信する量が多いほどネットワークを効率的に扱えるのは前述した通りであるが，ゼロコピー通信の方が高速になる程度のサイズで通信を行うことはプロセッサの有効利用にもつながる。メモリレジストレーションされた領域へのコピーを伴う通信では，そのコピー作業にプロセッサの資源を使用してしまうため，その間はユーザーが本来求めている計算はできない。一方，ゼロコピー通信の場合は一度送信命令をネットワークカードに出してしまえば，あとはプロセッサを自由に使用することができる。つまり通信の完了を待たずに他の処理を続けることができるため，通信に要する時間を隠蔽してプロセッサを常に有効な計算に割り当てることができる。このような通信と計算のオーバラップのためには MPI_ISEND などの非同期送受信関数を使用し，完了を待つ前に他の処理を記述する。ただし，この間は通信対象のメモリ領域を使用することはできないので注意が必要である。

5.5 通信がアプリケーション性能に与える影響

通信の話の最後に並列計算における通信性能の重要性を示す簡単なデータを紹介する。下のグラフは並列計算機のベンチマークとして広く使われている NAS Parallel Benchmarks の中の CG を PC クラスタで実行した結果である。横軸は使用したプロセッサ数，縦軸が性能である。黒いグラフが 1Gbps の Ethernet を使用した場合であり，灰色が Myrinet 10G を用いた場合の結果である。まったく同じマシン(Dual Core Opteron 2GHz，DDR2 667MHz) で，使用するネットワークのみを変えて測定している。マシンが同じであるにも関わらず 32 プロセッサでの性能が大きく異なることがわかる。一般に並列計算ではプロセッサ数が増加するにしたがって通信量が多くなるため，ネットワークに求められる性能も高くなる。そしてネットワーク性能が追いつかない程度までプロセッサ数を増やしてしまうと全体的な性能は落ちてしまう。上の例は 16 プロセッサまでは Gigabit Ethernet でも十分に扱える通信量であるが，32 プロセッサの場合は 10Gbps のインターコネクトでないと実用的な性能が出ない通信量になっていることを示している。

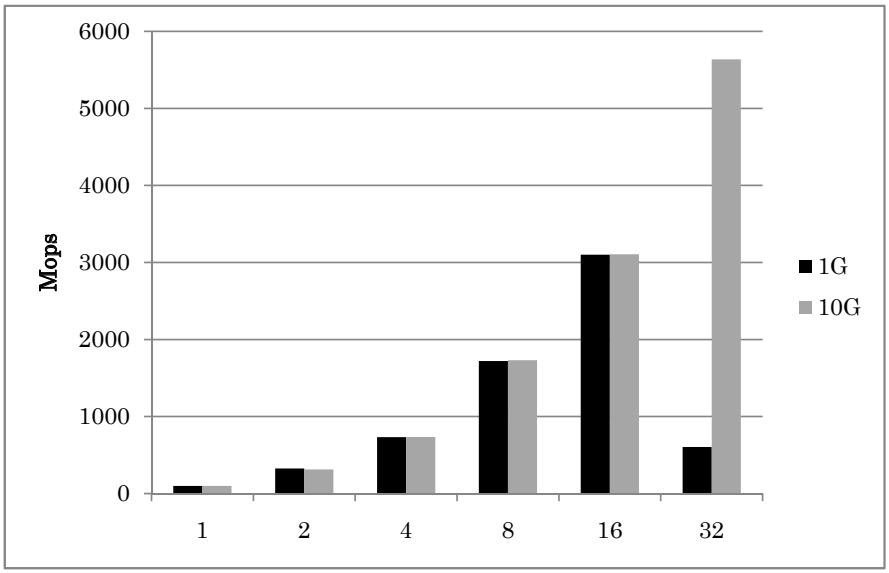

図 9　NAS Parallel Benchmarks CG (class C) の結果

スパコンプログラミングのためのヒント

並列計算機の規模に見合ったネットワーク性能を備えることは並列計算機の設計者の責任である。ただし，アプリケーション側でも通信をなるだけ少なくする工夫を行わないと，少ないプロセッサ数で性能限界に達してしまう。そして限界を超えてプロセッサ数を増やした場合は性能が上がらないのみならず，前ページのグラフのように大きく低下してしまうこともある。性能限界点は計算時間と通信時間の比で決まる。大規模化のためには，計算時間を長く，通信時間を短くしなくてはならない。一般的に，なるだけ計算時間を長くするためには各プロセスに大きな問題を割り当てるのがよい。このため，アプリケーション作成時には各ノードが担当する計算量を実行時のパラメーターとして与えられるようにしておき，何回かのテストランでそのパラメーターを調整することでノードが持つメモリを使い切るようにすべきである。

6. 参考文献

本稿ではプロセッサアーキテクチャ，オペレーティングシステム，インターコネクトについてアーキテクチャの基礎を紹介した。それぞれについて参考文献を一冊ずつ紹介する。

プロセッサアーキテクチャに関する参考書として有名なのは次の書籍である。

David A. Patterson , John L. Hennessy

“Computer Architecture Fourth Edition: A Quantitative Approach”

ただし本書はプロセッサアーキテクチャの複雑化に連動するように版を重ねるごとに内容が高度化している。アーキテクチャの基礎を学ぶには second edition 程度が適切かもしれない。

オペレーティングシステムの参考書としては以下のものがよい。

Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

“Operating System Concepts 7th edition”

通信に関しては通信の専門書よりも並列計算機のアーキテクチャ全般を解説した本の方が適切である。通信の専門書は長距離ネットワークやIPルーティングなど，インターネット技術に関連するものが多く，並列計算機の通信に特化して解説した本を探すのは困難である。並列計算機の解説書としてはやや古いが次のものがよい。

David E. Culler, Jaswinder Pal Singh, Anoop Gupta

“Parallel Computer Architecture: A Hardware/Software Approach”

本書では並列計算機の構成方式や通信，並列アプリケーションに至るまで並列計算機全般について幅広く解説されている。

7. おわりに

本稿ではスパコンプログラミングのヒントとなるよう，計算機システム全般を広く解説した。プロセッサの高速化に伴って Java やスクリプト言語のような，性能よりも再利用性や移植性を重視した抽象的なプログラミングが主流となりつつある近年だが，スーパーコンピュータは高い性能で計算してこそ意義のある計算機であり，依然として FORTRAN, C, C++ を中心に使用されている。そして，このようなハードウェアに近い言語でプログラミングをするにあたっては，高速化のため，あるいは致命的な性能低下につながるミスを避けるためにアーキテクチャの知識は重要である。特にこれからスーパーコンピュータを使い始める若い読者の方には，本稿で紹介したような並列計算機のアーキテクチャに興味を持っていただき，高速なプログラムを作成するための知識として生かしていただきたいと思う。