高性能コンピューティング特論講義メモ補助資料（7-2）

～科学技術計算におけるプロセッサ間交信～

１．転送性能の指標

・遅延時間（Latency）

メッセージの先頭が送信元プロセッサを出発してから、送信先プロセッサに到着するまでの時間。場合によっては、ソフトウェア的な latency とハードウェア的な latency に分割される。さらに、ソフトウェア latency は送信元における処理と送信先における処理に分割される。

・スループット（Throughput）

メッセージが転送されている間、単位時間当たりに転送可能なメッセージの量。通常は byte/sec 等で表すが、転送時間全体を求める場合、しばしばその逆数が用いられる。

転送時間の一般式：

$$T = a + \frac{N}{b}$$

ただし、a はソフトウェア・ハードウェアを含めた遅延時間、b はスループット、N はメッセージ長である。

この関係から、見かけ上の転送スループット b' が以下のように求まる。

$$b' = \frac{N}{T} = \frac{N}{a + \frac{N}{b}} = \frac{Nb}{ab + N}$$

ここで、見かけ上の転送スループットがちょうど本来のスループットの半分になるようなメッセージ長を「半性能長」と呼び $N_{1/2}$ と表す（”n-half”と読む）。

$$b' = \frac{N_{1/2}b}{ab + N_{1/2}} = \frac{1}{2}b$$

より

$$N_{1/2} = ab$$

すなわち、スループットが一定の場合、半性能長は転送遅延に比例する。従って、転送遅延を小さくすればするほど、半性能長が短くなり、いろいろなメッセージ長の転送を高速に実現することが可能になる。

ハードウェア的な転送遅延は、一般的に送受信プロセッサ間の距離に比例する。この時重要なのが、メッセージの転送方式である。

・ store&forward 方式：

メッセージが複数の中継ノードを経由する際、各中継ノードで一旦メッセージを全てバッファに格納し、次のノードへの転送準備ができてから（転送経路がオープンになってから)、バッファからのメッセージ送り出しを行う方式。

- wormhole 方式：
  中継ノードにおいて、メッセージの（非常に小さい）一部のみを保持し、それらが論理的につながっているようにパイプライン的に、中継ノード間をメッセージが転送される方式。
- virtual cut through 方式：
  store&forward に似ているが、中継ノードから次の中継ノードに向かう経路が空いていれば転送を継続し、塞がっていたらその中継ノードのバッファにメッセージを溜める。しかし、経路が空き次第、順次パイプライン的にメッセージを転送していく方式。

転送方式として store & forward 方式を取っている場合は各中継ノードにおけるメッセージの store 時間が加算され、それは store の回数、すなわち中継ノード数に比例する。wormhole 方式で転送される場合は、中継ノードの通過当たりの時間がほぼ一定の場合が多く、転送遅延はやはりそのノード数に比例する。Virtual cut through 方式は最も効率が高く、最良な場合は wormhole と等しく、最悪な場合は store & forward に等しい。

また、メッセージの転送パターンによっては、同一の経路（または経路上の一部）を複数のメッセージが同時に通過する場合がある。このような場合、一般的に見かけ上のスループットが低下する。先頭のメッセージが通過した後で 2 番目が、その後 3 番目が…というように順番に転送されると想定すれば、m 個のメッセージが同一経路を通過する場合、その見かけ上のスループットは b'/m に低下すると考えることができる。

２．隣接転送

1, 2, 3 次元空間に対する偏微分方程式を並列アルゴリズムで解き、問題空間をプロセッサにブロック的に割り当てれば、必然的に各次元サイズに応じた隣接転送が必要となる。

この性質を利用したのがメッシュ/トーラス結合である。また、ハイパキューブでは、サイズが適合すれば隣接プロセッサ間転送になるようなマッピングを行なうことが可能である。さらに、ハイパクロスバの場合は、プロセッサの次元数が問題空間の次元数以上であれば、どのような構成の隣接転送空間も、ネットワーク城でのメッセージ転送が無衝突になるようにマッピングできる。ただし、どのようなネットワークでも、全メッセージが無衝突に転送されるためには、全プロセッサが同期して転送開始・終了しなければならない。

また、問題が周期境界条件を持つ場合は、ネットワークのトポロジによっては最遠点プロセッサ間の交信が必要な場合がある。単方向メッシュあるいは単方向トーラス網は、近接転送の場合であっても不利である。一般的な問題では、双方向の近接転送が必要になるため、単方向トーラスの場合、逆方向のメッセージ転送においてほとんど全てのメッセージが経路上で衝突することになる。

単方向メッシュというのは厳密には存在し得ないが、例えば経路を時分割で各方向に利用するような場合、若干不利である。ただし、この場合は全プロセッサで一斉に同期を取ることができれば、メッセージの衝突は生じない（周期境界条件の場合を除く）。

３．放送 (broadcast)

1 プロセッサのデータを複数のプロセッサが要求する場合が多く存在する。このような場合に必要となる転送を**放送（broadcast)**と呼ぶ。放送には以下の 2 種類がある。

・一対全放送（one-to-all broadcast）

全プロセッサが等しく、1 プロセッサのデータを受信する。例えば、あるプロセッサが何かの代表値を持っていて、それを全プロセッサに通知する場合等に用いられる。

・全対全放送（all-to-all broadcast）

全プロセッサが等しく、他の全プロセッサに均一にデータを送信する。例えば、あるデータ集合を全プロセッサが分割して持っており、それらを互いに交換するような場合に用いられる。

これらに対し、以下のようなアルゴリズムが用いられる。

３．１　一対全放送のアルゴリズム

送信元プロセッサが全プロセッサに一回ずつメッセージを転送する方法では、*O(P-1)*のコストがかかってしまう（P はプロセッサ台数)。そこで、これを低減するために一般に木 (tree) 構造の転送が用いられる。これによって転送コストを *O(logP)* に抑えることができる。

まず送信元プロセッサ P0 が別のプロセッサ P1 にメッセージを転送する。次のステップで P0 と P1 が、各々P2 と P3 に個別にメッセージを転送する。以下、この手順を繰り返すと全体で $\log_2 P$ ステップで放送が終了する。

この際に注意しなければいけないのは、見かけ上は転送回数が log オーダーでできても、実際のメッセージ転送がネットワーク上で衝突した場合は、その分のスループットの低下を考慮しなければならないという点である。

例えば、単純バスで結合されたシステムでこのアルゴリズムを利用すると、それ最悪値として *O(P-1)* の通信コストがかかることになる（各自証明してみよ)。すなわち、一対一転送を繰り返すのと同じコストがかかることになる。

このアルゴリズムはハイパクロスバ、ハイパキューブ、メッシュ、トーラスにおいて期待通り、通信コストを log オーダーに抑えることができる。すなわち、一般的な超並列処理システムにおいては、汎用性の高いアルゴリズムである。

転送すべき元データのサイズを s とすると、全通信処理時間は以下のようになる。ただし、転送遅延を a、スループットを b とする。また、P は 2 のべき乗であるとする。

$$T = (a + s/b)\log_2 P$$

また、この応用として、2 進木ではなく n 進木を用いるアルゴリズムも考えられる。すなわち、送信元のプロセッサは n-1 台のプロセッサにデータを順次転送し、次のステップでは送信元を含む n 台のプロセッサが各々さらに n-1 台のプロセッサに転送をする、というステップを繰り返すものである。理論的には転送ステップは $O(\log_n P)$ まで落とすことができる。ただし、この場合もその間の転送が各ネットワークにおいて衝突しないことを保証しなければ、性能低下につながる恐れがある。

３．２　全対全放送のアルゴリズム

全対全放送のアルゴリズムとして最も単純に考えられるのは、一対全放送のアルゴリズムを、送信元プロセッサを順次変えながら P 回実行することである。しかし、その方法による通信時間は

$$T = P(a + s/b)\log_2 P$$

となり、特に超並列システムで P が大きくなった場合に不利である。

そこで、全てのプロセッサが常に何らかのデータを転送するようにアルゴリズムを改良する。この方法で最も単純なものは、隣接するプロセッサ同士が、隣から来たエデータを「バケツリレー式」にさらに隣に順次転送していく方法である。あるプロセッサ P_i を考える。第一ステップで、P_i は P_{i+1} に自分のデータを送信し、同時に P_{i-1} からデータを受信する。この時点で P_{i-1} のデータが入手できる。次に、今入手した P_{i-1} からのデータを P_{i+1} に送信し、同時に P_{i-1} から次のデータを受信する。このデータは、P_{i-1} がその前のステップで P_{i-2} から受信したデータであるため、結果的に P_{i-2} のデータが受信できることになる。このステップを $(P-1)$ 回繰り返せば、全プロセッサのデータを全プロセッサが持つことになる。

このアルゴリズムにおける通信時間は

$$T = (P-1)(a + s/b)$$

である。これを先ほどの場合と比べると、通信時間はほぼ $1/\log_2 P$ になっている。

しかし、この方法ではやはり P が大きい場合に不利である。そこで、2 進木の木構造によるデータ転送を全プロセッサで同時に行なう方法を考える。例として $P = 8$、つまり 3 段の 2 進木の場合を考える。全プロセッサの番号は 3 bit の 2 進数として表せる。まず第一ステップで、各プロセッサは自分のデータを、自分のプロセッサ番号の最下位 bit を反転させた番号のプロセッサに送信し、同時にそのプロセッサからデータ受信する。この段階で、2 進木で最下段のレベルで隣接するプロセッサ同士が互いのデータを持つことになる。第二ステップでは、各プロセッサは自分のプロセッサ番号の下から 2 bit 目を反転させたプロセッサに、自分のデータと第一ステップで入手した隣接プロセッサのデータを一緒にして送信し、同時に相手から同等のデータを受信する。従って、このステップでは転送されるデータ長は元のデータの 2 倍になる。この結果、全プロセッサは 2 進木の下位 2 レベルの、自分を含む部分木中の全てのプロセッサのデータを持つことになる。そして第三ステップ

で自分の持つ全てのデータを、プロセッサ番号の最上位 bit を反転させたプロセッサと交換する。この結果、全プロセッサが全プロセッサのデータを持つことになる。

このアルゴリズムは転送パターンがバタフライ転送に似ているので、バタフライ・アルゴリズム、あるいはカスケード・アルゴリズムと呼ばれる。通信時間は

$$T = \sum_{i=0}^{\log_2 P-1}(a + 2^i s/b) = \log_2 P + (P-1)s/b$$

となる。これをバケツリレー方式と比べると、スループットに関する項では同じであり、転送遅延に関する項が linear オーダーから log オーダーに減っていることがわかる。従って、P が大きくなった場合に有利である。

ただし、このアルゴリズムは一対全放送を log オーダーで行なうアルゴリズムと同様、転送中のメッセージの衝突に注意する必要がある。このアルゴリズムでは常に全プロセッサがデータを送信しており、またその範囲はステップ毎に広がっていく。一般に、メッシュ/トーラス系では、カスケード・アルゴリズムは途中でのメッセージ衝突が生じるため上のような理想的な通信時間を実現することはできない。ハイパクロスバ、ハイパキューブはこのアルゴリズムを無衝突で実現可能である。

３．３　全対全個別通信

全プロセッサが同時にデータを送信するもう一つの例が、全対全個別通信(all-to-all personalized communication)である。これは、各プロセッサが他の全プロセッサにそれぞれデータを送るが、その内容が相手プロセッサ毎に異なる場合の通信である。従って、全対全放送のアルゴリズムを用いることはできない。この通信が用いられる例としては、例えば各プロセッサが行列の一部を各々保持しており、その行列を転置するような場合がある。

場合によっては、送信相手が全プロセッサでなく、その一部であることもある。例えば 2 次元のプロセッサ空間を想定し、その上に 3 次元の行列をブロック式に配置し、その転置を行なう場合等がこれに当たる。しかし、一般的な転送アルゴリズムの考え方は全対全の場合と同じでよい。従ってここでは全対全の場合のみを考える。

問題の性質上、全プロセッサは $P-1$ 回の送信と受信を行なう必要がある。最も単純なアルゴリズムは、各ステップにおいて、全プロセッサがデータを送信する相手プロセッサ番号が相対的に等しくなるようにする方法である。こうすれば、少なくとも 2 つのデータが 1 つのプロセッサに同時に転送されるようなことが防げる。問題は、この転送パターンがネットワーク上でメッセージ衝突を起こすか否かである。

先述の全対全放送の場合のように、このアルゴリズムは全プロセッサからのデータ送出と、その距離が段々長くなるという傾向があるため、メッシュ/トーラス系では不利である。これに対し、ハイパクロスバ、ハイパキューブではこれを無衝突に処理できる（ハイパキューブの場合、マッピングをうまくする必要がある)。

このアルゴリズムで転送が無衝突でできた場合の通信時間は

$$T = (P-1)(a + s/b)$$

である。

３．４　縮退転送 (reduction)

あるスカラ（またはベクタ）値を、その大きさを変えずに全プロセッサで順々に処理し、最終的に１つのスカラ（またはベクタ）値に縮退する転送がしばしば用いられる。例えば、全プロセッサがそれぞれ、あるデータの部分和を持っており、その総合計を求めたい場合等に必要となる。あるいは、各プロセッサの持つ値の中で最大値を求めたい場合等もこれに当たる。このような転送を縮退転送と呼ぶ。

縮退転送では、転送の最中に何らかの計算を行なう必要がある。総和を求める場合なら加算、最大値を求める場合なら比較演算である。これは一般的には転送→演算→転送…という順序で交代に行なわれる。そこで、ここではそのうちの転送の部分だけを取り出し、通信時間を考える。

縮退転送は、基本的に一対全放送や全対全放送で用いたような木構造のアルゴリズムを用いることができる。ただし、通常の場合、最終的に求められた縮退値は、全プロセッサに通達される必要がある。それを考慮すると以下のようなアルゴリズムが考えられる。

まず、一対全放送の場合と逆の手順で、プロセッサ台数をステップ毎に半分に減らしながら、データの転送と演算を行なう。これを繰り返していき、最後に代表プロセッサにデータを集め、縮退を完了する。次にそのデータを一対全放送の要領で再度全プロセッサに通達する（つまり縮退時とは逆の方向で木構造をたどっていく）。途中の演算を省いた通信時間は以下の通りである。ただし、s は縮退対象となるデータのサイズである。

$$T = 2\log_2 P(a + s/b)$$

この方法では「行きと帰り」のデータ転送が必要であった。しかし、一般的に縮退時のデータを全プロセッサが必要とすることを考えれば、縮退が完了した時点で、全プロセッサにそのデータが存在するのが望ましい。そこで、全対全放送を木構造で行なった時のカスケード・アルゴリズムを適用することを考える。

手順は先述のカスケード・アルゴリズムとほとんど同じである。唯一の違いは、全対全放送の場合は、各ステップ毎にデータのサイズが倍になっていったのに対し、縮退の場合は途中の演算により、常にデータ・サイズは固定となるため、総転送量が異なってくる点である。従って、通信時間は以下のようになる。

$$T = \sum_{i=0}^{\log_2 P-1}(a + s/b) = \log_2 P(a + s/b)$$

よって、先出の縮退アルゴリズムの半分の通信時間ですむ。縮退転送では加算を行なう例が最も代表的なので、特にその場合をカスケード・サムと呼ぶことがある。ただし、これまでの木構造アルゴリズムと同様、メッセージの衝突に関して考慮する必要がある。

３．５　その他の転送パターン

これ以外にも、科学技術計算で頻出する転送パターンとして、縮退転送中にデータを保持するスキャン転送、FFT 処理の最終段階で出てくるシャッフル転送等がある。