社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ARC－157　(17)
2004－HPC－ 97　(17)
2004／3／2

Multi-master divisible load model における漸近最適スケジューリング

須　田　礼　仁†

負荷の均衡化のためのデータ再分散を最適化するためには、プロセッサごとの通信所要時間の違いを考慮して割り当てるデータ量を調整することと、データを分割して複数回に分けて通信を行うこととを考慮しなければならない。一般にはこれは非常に難しい問題であるため、本稿では問題をできるだけ簡単化した multi-master divisible load (MMDL) モデルを提案する。さらに、通信性能が均一なクラスタ（計算性能はヘテロでもよい）上での MMDL に対する漸近最適スケジューリングを示す。提案手法では、「一定した通信性能」と「一貫した計算性能低下」という 2 つの条件を満たしていれば、通信と計算がどの程度オーバーラップできるかをよらずに一定のアルゴリズムでスケジューリングができる。

Asymptotically Optimum Scheduling for a Multi-Master Divisible Load Problem

Reiji SUDA†

Optimization of data redistribution for load balancing must include task allocation that absorbs the differences of the communication timings of the processors and multi-round communications with split data. It is a difficult problem in general, and this paper proposes to simplify the problem into the Multi-Master Divisible Load (MMDL) model. Asymptotically optimum scheduling without communication contention for MMDL on clusters of homogeneous communication performance (computational performance can be heterogeneous). The scheduling algorithm is applicable to various assumptions on the degree of parallelism of communication and computations, on the conditions of “constant communication performance” and “consistent degradation of computational performance”.

1. はじめに

1.1 ERXPP におけるデータ再分散問題

著者は 1) において、並列計算環境の処理性能を数値ライブラリの部分で最大限に引き出すことにより、アプリケーションプログラムの実装の複雑さをおさえたままでネットワーク計算環境の性能を引き出すこと (ERXPP: Easy and Robust Extension of Parallel Performance) を提案した。その中で、次のような状況が例として挙げられている。

(1) アプリケーションがホモなクラスタ上で実行されていて、数値ライブラリはさらにネットワーク上の計算資源を追加して利用する
(2) アプリケーションがシステム（あるいはライブラリ）に適したデータ分散をしていない（ヘテロ性能なクラスタ上で一様に分割しているなど）

これらの状況では、アプリケーションと数値ライブラリとの境界で、数値ライブラリ側の負荷を均衡化するためのデータ再分散が必要となる。ここで必要となるデータ再分散のアルゴリズムについては従来から研究はあるものの、上述のような問題設定に対して十分な答えを出してはいないように思われる。

1.2 再分散に関する既存の研究

データの再分散というのは並列処理の古いテーマの一つである。データ再分散の論文で最も多いのはブロックサイクリック分散でパラメタや分散次元が変化する場合の再分散に関するもの[8)~11)] である。HPF の再分散指示に対するコンパイラの処理などもこれに非常に近い。FFT などで生じる行列転置、あるいは全対全通信も一種のデータ再分散と理解することもできる[5)] が、これについても各種の研究がある。これらはいずれも（ほぼ）一様な分散から（ほぼ）一様な分散への再分散である。

また、動的負荷分散の一部の手法は、不均一な分散間での再分散を必要とする。この場合、集団通信としてみると（往々にして不均一な）全対全通信であり、不均一全対全通信のアルゴリズムの研究[6)] も関連研究として挙げることができる。通信と計算のオーバーラップについて考察している論文[7)] もいくつかあるようであるが、通信遅延までは負荷分散の最適化問題には含まれず、再分散の通信は単純にオーバーヘッドと見なされていることが多い。

不均一な再分散では一般に通信の開始・終了時刻はプロ

† 東京大学 情報理工学系研究科/科学技術振興機構 CREST
G.S. of Information Science and Technology, the University of Tokyo/CREST, Japan Science and Technology Agency

セッサによって異なる。通信所要時間の差異があるので、それを考慮して割り当てる仕事量を調整しなければ最適とはならない。しかしそこまで考察した研究は少なく、主な事例はマスター・ワーカー型に限られ、その中の主なものは divisible load theory (略して DLT)[2] と呼ばれる一連の研究である。さらに、仕事の再分散は一度に送るのではなく、一般には分割して送ったほうが所要時間は短くて済む。DLT ではこのように分割してタスク転送を行う手法を multi-installment あるいは multi-round と呼んでいるが、これを考慮したものは DLT でもさらに事例が少ない[3),4)]。

データ並列という視点を離れて見ると、データフローグラフのような形に表現されたタスクを再分散・処理するという問題と見なすこともできる[12)]。この場合、最適解を出すのは極めて非現実的で、近似解法を用いることになるものと思われる。しかし問題をデータ並列の形に制限しても、上述の考察のように最適解を求めるのは非常に難しく、近似的に最適化せざるをえないのが現実である。それを考えると、「データ並列というパラダイムを捨てる」という選択肢も、十分に検討に値するように思われる。しかし、データ並列という特性を活かした近似アルゴリズムと、汎用的な近似アルゴリズムとでは、計算量のオーダーが異なるであろう。実行時にすばやく再分散の内容を決定したいような環境では、データ並列という特性を考慮した高速な近似アルゴリズムの方が望ましいという場合も多いのではないだろうか。本稿ではデータ並列のパラダイムを採用し、DAG スケジューリングのようなタスク並列は考えないことにする。

1.3 本稿の提案

データ並列というパラダイムに制限しても、通信の不均一性と計算量の調節、それに最適なマルチラウンド処理という課題は十分すぎるほど難しい。そこで問題の枠組みをできるだけシンプルにして、小さな計算量で確実に最適解が求まることを目指すことにした。タスクのモデルとして最もシンプルなものを考えると、前述の divisible load theory のモデルということになる。これをマスター・ワーカーの枠から開放し、負荷の均衡化に伴うデータ再分散という問題に定式化しなおしたものが、本稿で提案する **multi-master divisible load (MMDL)** のモデルである。

システム側のモデルとしても本稿では最も簡単なものを取り上げる。具体的には計算性能がヘテロなプロセッサからなるクラスタである。

本稿では上記のような最も単純な問題設定において、性能向上のタイトな上限の計算方法を示し、さらにその上限に漸近する通信衝突のないスケジューリングを示す。

2. Multi-master divisible load のモデル

本節では本稿で取り上げる問題を定式化する。前半ではタスクのモデルとして、multi-master divisible load のモデルを提案する。後半ではシステムのモデルとして、通信性能が一様なクラスタを取り上げることを説明する。

2.1 タスクのモデル

今回提案する **multi-master divisible load (MMDL)** のモデルは以下の通りである。

複数のプロセッサがあり、最初は第 i プロセッサに x_i の量のタスクが割り当てられている。タスクは相互に依存がなく、独立に処理することができ、一様で、どのプロセッサでも処理することができる。タスクは任意のサイズに切り分けることができ、タスクを移送する場合のメッセージサイズはタスクサイズに比例する。処理の結果は収集する必要はないものとする。

従来の divisible load theory[2] では、最初すべてのタスクがマスターと呼ばれるひとつのプロセッサに置かれていて、他のプロセッサ（ワーカー）には初期割り当てがない。すなわちマスター・ワーカー型の並列処理だけしか考えられておらず、適用範囲が狭かった。今回提案する MMDL ではマスター・ワーカーという区別はなく、タスクを輸出するか輸入するかで結果的にマスターとワーカーに分かれることになる。

2.2 システムのモデル

システム側の問題設定としても、今回は以下のように最も簡単なものを選択した。

プロセッサはクロスバスイッチにより接続されており、スループットはどのプロセッサ間でも同じで、単位サイズのタスクに対して β の時間がかかるとする。通信の立ち上がり遅延は無視する。プロセッサの計算性能は異なってもよく、第 i プロセッサ上で単位サイズのタスクを処理するために γ_i の時間がかかるとする。各プロセッサは通信と計算を同時に行うことができないとする。

ここで本質的なのは通信性能が一様であり、通信の立ち上がり遅延が無視できることである。現実的には、通信の立ち上がり遅延については追加的なオーバーヘッドと見なすことにより、近似的な最適解を構成することが可能である。しかし通信性能が非一様である場合には問題は一気に難しくなる。

MMDL では、プロセッサの計算性能のヘテロ性は何の難しさも引き起こさない。一方で通信性能がヘテロになってしまうと状況がワンランク複雑になり、解の候補として一段広いクラスを想定しなければならなくなる。計算性能だけがヘテロという状況は一般にあまり難しい問題ではなく、丁寧に考察をしてやればうまく対応ができる一方で、通信のヘテロ性は一段難しい問題であるようである。

3. 漸近最適解の構成

前節ではタスクとシステムのモデルを定義した。本節ではこれを処理するための所要時間をできるだけ短くするスケジューリングを考える。

3.1 漸近最適解の構成上の制約

まず、スケジューリングに対して 2 つの制約を課す。こ

れらの制約は解の漸近的な最適性を阻害しないことが重要である。

3.1.1 リレーなしの制約

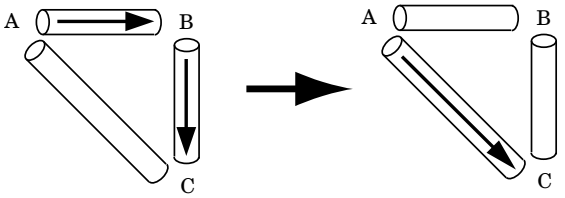

図 1 タスクのリレーの解消
Fig. 1 Removing relay of tasks

通信スループットが一様で立ち上がり遅延が無視できる場合には、タスクのリレーが行われない最適解が存在する。すなわち、図 1 に示すように、プロセッサ A から B にも B から C にもタスクが転送される解があれば、A から C にタスクを転送することにより A から B または B から C のいずれかのタスクの転送がない解を構成することができる。

そこで、以下では**タスクのリレーが生じない解**のみを考えることとする。しかし、スループットが一様でない場合や、立ち上がり遅延が無視できない場合には、リレーをしなければ最適解が得られないような例題を簡単に構成することができる。

3.1.2 定常ラウンド 反復の制約

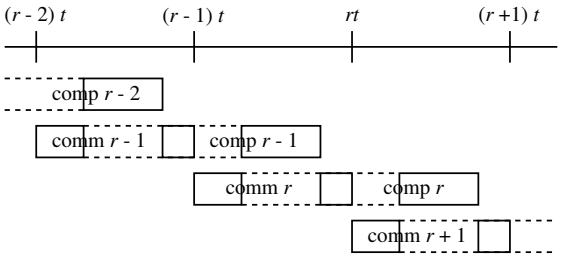

図 2 定常ラウンド反復スケジューリング
Fig. 2 Scheduling by repeating stable rounds

さらに今回は解に対して次の制約を課す。すなわち、(最初と最後を除いて) 同一の手順からなる「**ラウンド**」を連続的に反復するというものである。さらに、第 r ラウンドで処理されるタスクは第 $r-1$ ラウンドで転送されるものとする。図 2 は通信と計算がオーバーラップできない場合のスケジューリングの模式図である。このように制約をすることで、問題は定常状態における 1 ラウンド分のスケジューリングを求めることに帰着される。これらの制約は Beaumont ら[3] が提案した (single-master) divisible load に対する漸近最適解のものと本質的に同じである。

ラウンド数を R とすると、通信の立ち上がり遅延をゼロとした場合の 1 ラウンドの所要時間 (以下、ラウンド時間という) は T/R と書ける。また、通信の立ち上がり遅延を含めたラウンド時間はある定数 α を用いて $T/R+\alpha$ となるとする。最初と最後のラウンドではそれぞれ通信と計算のみが行われるので、全体の所要時間は $T+T/R+R\alpha$ となる。このとき $R=\sqrt{T/\alpha}$ で所要時間は最小値 $T+2\sqrt{T\alpha}$ を取る。ここで α は定数であるから、問題サイズを大きくしていった極限で所要時間は T に漸近する。

以上により、定常状態における 1 ラウンドのスケジューリングを最適にすれば、定常的なマルチラウンドの解は所要時間の下限 T に漸近することが分かる。

通信の立ち上がりを無視すると、ラウンドのスケジューリングはラウンド数 (分割数) によらないので、$R=1$ としてよい。このときラウンド時間は所要時間の下限 T である。

3.2 所要時間下限・性能上限の計算

次に、定常ラウンドの最適スケジューリング問題を考えるが、その第 1 段階は ($R=1$ における) ラウンド時間 T の最小値を求めることである。これは全体の所要時間の下限であり、性能の上限を求めることに等しい。

通信性能が計算性能に比べて極端に低く、$\beta \geq \gamma_i$ が成り立つ場合には、仮にプロセッサ i が他のプロセッサよりも多くのタスクを持っていたとしても、他のプロセッサに転送するよりも自分で処理してしまったほうが速い。したがってこの場合にはプロセッサ i はマスターになることはできない。そしてラウンド時間 T は x_i/γ_i を下回ることはできない。

逆に $\beta < \gamma_i$ が成立する場合には、プロセッサ i のタスク x_i はできるだけ多くのタスクを他のプロセッサに送りつけたほうが速く処理ができる。よって最短のラウンド時間は x_i/β となる。そこで、

$$T_i = \min\{x_i/\beta, x_i/\gamma_i\}$$

とすると、ラウンド時間 T は

$$T_L = \max\{T_i\}$$

よりも短くなることはできない。

ここで、各プロセッサに対して「**追加タスク量**」y_i を考える。すなわち、$y_i > 0$ の場合には第 i プロセッサは y_i だけのタスクを他のプロセッサから受け取り、$y_i < 0$ の場合には他のプロセッサに y_i だけのタスクを送るのである。

システム全体としてタスク処理に過不足があってはならないので、$\sum y_i = 0$ が成り立つ。またラウンド時間が T であることから、

$$(x_i + y_i)\gamma_i + |y_i|\beta \leq T$$

が必要である。ここで無駄なタスクの移動を除けば、$x_i\gamma_i \geq T$ でプロセッサ i がマスターとなる場合には等式がなりたつことは明らかである。逆に $x_i\gamma_i < T$ の場合にはプロセッサ i はワーカーであり、受け入れられるタスク量の上限は

$$\bar{y}_i = (t - x_i\gamma_i)/(\beta + \gamma_i)$$

となる。そこでこの $\bar{y}_i$ をマスターにも拡張して

$$\bar{y}_i = \begin{cases} (x_i\gamma_i - t)/(\beta - \gamma_i) & \text{if } x_i\gamma_i \geq t \\ (t - x_i\gamma_i)/(\beta + \gamma_i) & \text{otherwise} \end{cases}$$

と定義する。すると $y_i \leq \bar{y}_i$ がすべてのプロセッサに対して成立しなければならない。

もし $T \geq T_L$ で

$$Y(T) = \sum \bar{y}_i(T) \geq 0$$

であれば、これまでに述べてきた条件をすべてを満たす y_i が得られる。従って $Y(T_L) \geq 0$ であれば T_L が最小のラウンド時間である。そうでない場合、$Y(T) = 0$ を満たす T が存在すれば、その T が最小のラウンド時間となる。

上記の最小の T を求める問題は一種の線形計画問題となり、多項式時間で解ける。しかしここでは $Y(T)$ は T に関して連続で狭義単調増加な区分的折れ線関数である。$Y(T)$ のグラフの傾きが変わる節点は高々 p 個であるので、二分探索法を用いれば最小ラウンド時間 T はプロセッサ数 p に対して $O(p \log p)$ という非常に少ない計算量で決定することができる。

3.3 マスターワーカーマッチング

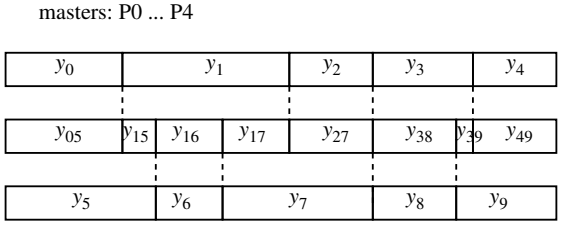

図 3 区間交差マッチング
Fig. 3 Interval Intersection Matching

以上のようにして最小ラウンド時間 T と追加タスク量 y_i が求まるが、どのマスターからどのワーカーにどれだけのタスクを移送するかはまだ決まっていない。通信スループットが一様で、通信立ち上がり遅延を無視しているので、実はどのように設定しても構わない。しかし、ここでは図 3 に示すようなマスターとワーカーのマッチングとタスク移送量の決め方を提案する。

i よりも若い番号を持つマスター（ワーカー）の追加タスク量の絶対値の総和を M_i (W_i) とする。第 i プロセッサがマスターの場合にはそれを $[M_i, M_{i+1})$ という区間に、ワーカーの場合には $[W_i, W_{i+1})$ という区間に対応させる。そしてマスター i とワーカー j との間では対応する区間の重複部分の長さ

$$y_{ij} = \max\{0, \min\{M_{i+1}, W_{j+1}\} - \max\{M_i, W_j\}\}$$

のタスクを転送するものとする。

この手法を「**区間交差マッチング (interval intersection matching)**」と呼ぶことにする。このように y_{ij} を決めると、各ラウンドでのタスク転送のためのメッセージの総数は $p-1$ 以下となる。どのようなマッチングをしてもそれぞれのマスターとワーカーは 1 つ以上のメッセージを扱うので、このメッセージ総数は最小値の 2 倍未満であることが分かる。また、マスターとワーカーを $|y_i|$ の順に並べておくと、各プロセッサが扱うメッセージ数が比較的少なくて済むと期待される。（マスターとワーカーを $|y_i|$ の大きさで逆順に並べるのが最悪である。）

3.4 衝突のない通信スケジューリング

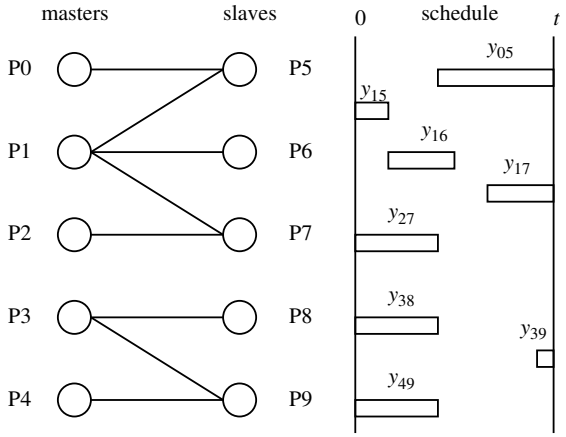

図 4 図 3 の例に対する通信スケジューリング
Fig. 4 Communication scheduling for the example of Figure 3

マスターワーカーマッチングとして前節の区間交差マッチングを仮定する。すると、以下のようにして通信が衝突しないようなスケジューリングを決めることができる。

まず、区間交差マッチングを用いている場合には、あるマスターが複数のワーカーにタスクを送信する場合、最初と最後以外のワーカーは、そのマスター以外とは通信しないことに注意する。従って、最初と最後以外のワーカーとの通信のタイミングは、マスターの都合で決めて差し支えない。これはマスターとワーカーの立場を入れ替えても同じである。

マスターが複数のワーカーと通信する場合には、最初のワーカーへの通信をラウンドの最初に、最後のワーカーへの通信をラウンドの最後にスケジューリングすることにする。ワーカーから見ると逆に、最初のマスターからの通信がラウンドの最後に、最後のマスターからの通信がラウンドの最初に来ることになる。最初と最後の相手以外との通信は、先行する通信が終了し次第行うこととする。これを示したのが図 4 である。左のグラフはマスターとワーカーのマッチングを示し、右は通信スケジューリングを示す。

結果的に通信の順序は非常に単純である。送信側は番号順に送信し、受信側は番号の逆順に受信すればよい。これだけで通信の衝突が避けられ、しかも決められたラウンド時間の中ですべての通信が終了できる。これを「**逆順通信スケジューリング (reverse ordering of communication scheduling)**」と呼ぶことにする。（最悪のスケジューリングは送受信両方が番号順にした場合で、すべて

の通信が逐次化されてもっとも時間がかかることになる。）

最小所要時間 T を求めた段階で、各プロセッサでの計算時間と通信時間を合わせて T 以下になるようになっている。そこで、計算は通信をしていない時間帯に適当に詰め合わせればよい。以上により、漸近的に最適な定常ラウンドスケジューリングが得られる。

4. アルゴリズムの拡張

前節では、2 節で導入した問題に対する漸近最適スケジューリングを提案した。本節では提案手法をいくつかの問題に拡張する方法について述べる。

4.1 計算結果を集計する場合

前節では、タスクは分散させるが処理結果は集計しなかった。処理結果を送信元のプロセッサに集計するように問題を変更してもほとんど同じように解くことができる。なお、通信と計算が重複できないので、送信と受信も同時にはできないと仮定する。

最小所要時間 T と追加タスク量 y_i の決定においては、通信時間として分散と集計の両方をカウントする。マッチングは従来どおりである。スケジューリングにおいては分散の通信コストと集計の通信コストの比に従って各ラウンドを 2 分割し、前半では前のラウンドの集計を、後半では次のラウンドの分散を行えばよい。

4.2 通信と計算がオーバーラップできる場合

通信と計算がオーバーラップできる場合も、方程式を若干変更するだけで対応することができる。

通信と計算がオーバーラップできるという場合でも、通信と計算が同時に行われた場合に両方の性能がまったく変化しないということは多くはない。通信と計算が並行したときにそれぞれがどれだけの性能になるかということはハードウェアに依存する問題である。

しかし、最初に述べたように、プロセッサ間の通信の性能に少しでも違いがあると、タスクのリレーを考慮したスケジューリングに拡張しなければならなくなる。これを避けるためには、「通信と計算が同時に行われた場合には、計算性能は低下するが、通信性能は低下しない」という仮定が必要となる。この条件を「**一定した通信性能 (constant communication performance)**」と呼ぶことにする。

通信が同時に行われている場合に単位サイズのタスクを処理するのにかかる時間を γ_i' とする。そしてプロセッサ i がマスターであると仮定する。できるだけ多くのタスクを他人に押し付けるにしても、その通信中にもできるだけの計算をするほうが有利である。ラウンド時間 T 中に常に通信と計算が行われている状況を想定すると、通信時間が $-\beta y_i = T$、計算時間が $(x_i + y_i)\gamma_i' = T$ となり、これから

$$T = x_i\beta\gamma_i'/(\beta + \gamma_i')$$

となる。しかし β や γ_i' があまりに大きい場合には自分ですべて計算してしまったほうが速いこともある。そこで、プロセッサ i のタスクをすべて処理するために最低限必要な時間は

$$T_i = \min\left\{x_i\gamma_i, x_i\frac{\beta\gamma_i'}{\beta + \gamma_i'}\right\}$$

ということになる（この式は $\gamma_i' \to \infty$ で前節の T_i の定義に一致する）。これより、ラウンド時間 T は

$$T_L = \max\{T_i\}$$

よりも短くなることはできない。

次に、ラウンド時間 $T \geq T_L$ が与えられたものと仮定する。このとき $x_i\gamma_i \geq T$ であればプロセッサ i はマスターであり、$y_i \leq 0$ である。プロセッサ i が仕事を他人に送信している間も計算は行うほうが有利であるから、マスターは常に計算をしていると仮定してよい。ラウンド時間 T のうち通信を行っている $-\beta y_i$ の間に処理できるタスクの量を x_i' とすると、

$$-\beta y_i = x_i'\gamma_i'$$

となる。それ以外の時間で残りのタスクが処理されなければならないので

$$T + \beta y_i = (x_i - x_i' + y_i)\gamma_i$$

となる。これらの式から x_i' を消去することにより、追加タスク y_i が

$$y_i\left(\gamma_i - \beta\left(1 - \frac{\gamma_i}{\gamma_i'}\right)\right) = T - x_i\gamma_i$$

のように決定される。これも $\gamma_i' \to \infty$ で前節の結果に一致する。

一方 $x_i\gamma_i < T$ であればプロセッサ i はワーカーで、$y_i \geq 0$ である。やはり通信をしていても計算は行うほうが有利であるから、受け入れタスク量を最大にするにはワーカーが常に計算をしている状況を想定すればよい。マスターの場合と同様に考察すると、β の符号だけが異なる

$$\bar{y}_i\left(\gamma_i + \beta\left(1 - \frac{\gamma_i}{\gamma_i'}\right)\right) = T - x_i\gamma_i$$

が受け入れタスク量の上限となることが分かる。

以上のように T_i と $\bar{y}_i$ が再定義されるが、これ以外の部分は通信と計算が同時に実行できない場合とまったく同じである。

すなわち、提案手法は通信と計算が同時に実行できる場合にも拡張可能である。但し、「一定した通信性能」の仮説が成立しなければならない。しかし、計算性能は低下しない場合でも、完全にストップしてしまう場合でも、送信と受信とで性能が異なる場合でも対応することができる。

4.3 通信が full duplex の場合

通信と計算が同時に実行できる場合には通信が full duplex であることも想定できる。タスクの処理結果を集計する場合にこれが問題となる。もし half duplex であれば 4.1 節で述べたとおりの手法で問題がない。しかし full duplex であれば、タスクの分散と結果の収集を同時に行うことが可能となる。

ここで問題となるのが、タスクの分散と結果の収集とが同時に行われた場合に、計算性能の低下がどれだけになるかということである。今回提案している手法では、追加タ

スクの量 y_i はスケジューリングを考慮せずに決めることができる。この簡単さを維持するためには、ある種の仮定が必要である。

プロセッサ i がマスターであると仮定する。単位サイズのタスクの送出にかかる時間はこれまでどおり β とし、このときの計算性能を γ_i' とする。単位サイズのタスクの処理結果の受信にかかる時間を β' とし、このときの計算性能を γ_i'' とする。さらに、タスクの送出と結果の受信の両方が同時に行われているときの計算性能を γ_i''' とする。

タスクの送出と結果の受信の両方が同時に行われている時間を τ とする。この間に x_i''' のタスクが処理されるとすると

$$\tau = x_i'''\gamma_i'''$$

でなければならない。タスクの送出だけが行われている時間帯では

$$-\beta y_i - \tau = x_i'\gamma_i'$$

結果の収集だけが行われている時間帯では

$$-\beta' y_i - \tau = x_i''\gamma_i''$$

となる。通信が行われていない時間帯には

$$T - \beta y_i - \beta' y_i + \tau = (x_i + y_i - x_i' - x_i'' - x_i''')\gamma_i$$

でなければならない。ここで τ はスケジューリングの結果によって決まる値であり、それに T の決定が依存して欲しくない。

そこで上記より x_i', x_i'', x_i''' を消去し、得られる y_i の式の τ の係数が 0 となる条件を求めると、

$$\frac{1}{\gamma_i} - \frac{1}{\gamma_i'''} = \left(\frac{1}{\gamma_i} - \frac{1}{\gamma_i'}\right) + \left(\frac{1}{\gamma_i} - \frac{1}{\gamma_i''}\right)$$

となる。すなわち、送受信同時に行われている場合の計算スループットの低下は、送信のみの場合の計算スループットの低下と受信のみの場合の計算スループットの低下の和となるという条件である。この条件を「**一貫した計算性能低下 (consistent degradation of computational throughput)**」と言うことにする。

上記の意味で計算性能の低下が一貫していれば、最小所要時間 T はスケジューリングに依存しない。そこで、これまでどおり T_i と $\bar{y}_i$ を適当に修正してやれば、提案手法が適用できる。

以上のように、各プロセッサの計算と通信がどの程度オーバーラップできるかをかなりの自由度で設定しても、提案手法を適用することができる。必要な仮定は、「一定した通信性能」と「一貫した計算性能低下」である。上記の T_i や $\bar{y}_i$ はプロセッサごとにしか設定しないので、プロセッサごとに仮定が異なっていてもよい。しかし、full duplex か half duplex かは送信側と受信側が一致していなければならないので、システム全体でどちらかに統一されていなければならない。

5. おわりに

本稿では負荷の均衡化のためのデータ再分散のアルゴリズムの基礎的な考察として、multi-master divisible load (MMDL) のモデルを提案し、通信が一様なクラスタ上で漸近的に最適なスケジューリングを実現する極めて高速なアルゴリズムを提案した。「一定した通信性能」と「一貫した計算性能低下」の 2 つの仮定を満たせば、通信と計算が完全にオーバーラップできる場合から、まったくオーバーラップできない場合まで、連続的に一括して扱うことができることが明らかとなった。

今後は提案手法の実装と評価および既存手法との比較、提案手法のアプリケーションプログラムへの応用、モデルと手法の拡張について研究を進める予定である。

参考文献

1) 須田礼仁、「ERXPP — 数値ライブラリによる並列計算性能を簡易かつ適応的に引き出す方式の提案」、情報処理学会研究報告 2003-HPC-96, pp. 19–24.
2) T. G. Robertazzi, *Ten Reasons to Use Divisible Load Theory*, IEEE Computer, Vol. 36, No. 5, May 2003, pp. 63–68.
3) O. Beaumont, A. Legrand and Y. Robert, *Scheduling divisible workloads on heterogeneous platforms*, Parallel Computing, Vol. 29, No. 9, Sep. 2003, pp. 1121–1152.
4) Y. Yang and H. Casanova, *RUMR: Robust Scheduling for Divisible Workloads*, Proc. HPDC-12, Jun. 2003.
5) A. Dubey and D. Tessera, *Redistribution strategies for portable parallel FFT; a case study*, Concurrency Computat: Pract. Exper. Vol. 13, No. 13, Mar. 2001, pp. 209–220.
6) W. Liu, C.-H. Wang and V. K. Prasanna, *Portable and Scalable Algorithms for Irregular All-to-All Communication*, Proc. ICDCS'96, pp. 428–436.
7) S.J.Deitz, B.L.Chamberlain, S.-E.Choi, L.Snyder, *The design and implementation of a parallel array operator for the arbitrary remapping of data*, Proc. PPOPP 2003, pp. 155–166.
8) J. Garcia, E. Ayguade and J.Labarta, *A framework for integrating data alignment, distribution, and redistribution in distributed memory multiprocessors*, IEEE Trans. Par. Dist. Sys., Vol. 12, No. 4, Apr. 2001, pp 416–431.
9) C.-H. Hsu, S.-W. Bai, Y.-C. Chung, C.-S. Yang, *Generalized basic-cycle calculation method for efficient array redistribution*, IEEE Trans. Par. Dist. Sys., Vol. 11, No. 12, Dec, 2000, pp. 1201–1216.
10) N. Park, V. K. Prasanna, C. S. Raghavendra, *Efficient algorithms for block-cyclic array redistribution between processor sets*, IEEE Trans. Par. Dist. Sys., Vol. 10, No. 12, Dec, 1999, pp. 1217–1240
11) F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert, *Scheduling block-cyclic array redistribution*, IEEE Trans. Par. Dist. Sys., Vol. 9, No. 2, Feb. 1998, pp. 192–205.
12) S. Ranaweera, D. P. Agrawal, *A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems*, Proc. IPDPS'00, pp. 445–450.