数理モデル化と問題解決 38－6
（２００２．３．４）

一対一プロセッサ間通信の一括化を考慮したタスクスケジューリングアルゴリズム

山本 裕己† 譚 林† 藤本 典幸† 萩原 兼一†
†大阪大学 大学院基礎工学研究科

本研究では，分散メモリ環境において実行時間の短い並列プログラムの生成を目的とした，タスクスケジューリングモデルとそのモデル上で動作するアルゴリズム SPPC を提案する．提案モデルの特徴は，通信の一括化の効果を定式化するために，通信処理におけるソフトウェアオーバヘッドを直接のパラメータとする点にある．通信の一括化は分散メモリ環境で効果的であることが知られている最適化技術である．SPPC は並列スケジュール生成の際，通信の一括化を考慮し，通信処理のソフトウェアオーバヘッドを効果的に削減する．評価実験の結果，SPPC は既存のアルゴリズムに比べて，実行時間の短い並列プログラムを生成できることがわかった．

A Task Scheduling Algorithm with Message Packaging Method for Point-to-Point Communications

Yuuki YAMAMOTO†, Lin TAN†, Noriyuki FUJIMOTO†, and Kenichi HAGIHARA†
† Graduate School of Engineering Science, Osaka University

Consider task scheduling for distributed memory machines. Our purpose is to generate a fast parallel program from a task schedule. In this paper, we first propose a task scheduling model in order to formulate the effect of message packaging. Message packaging is famous and important optimization technique for parallel programs on distributed memory machines. Then, we give a task scheduling algorithm named SPPC. SPPC runs under the proposed model. So, SPPC can decide well if message packaging should be applied. Some experimental results show that SPPC generates faster parallel programs than an existing algorithm.

1 はじめに

タスクスケジューリング手法を用いて，分散メモリ型並列計算機を対象にした実行時間の短い並列プログラム（以下，高性能並列プログラム）を生成するためには，通信の一括化 [1][7] を行う必要がある [4]．しかし，古典的タスクスケジューリングモデル（以下，古典的モデル）では，プロセッサ間の通信遅延をモデル化する際，ソフトウェアオーバヘッド（以下，オーバヘッド）の影響を無視しているため，効果的な通信の一括化を考慮することは困難である．

そこで，本研究では古典的モデルの通信遅延の定義を拡張することで，オーバヘッドを直接のパラメータとし，さらに，通信が同期する必要のないタスクスケジューリングモデルを提案する．提案モデルでは，LogP モデル [3] を用いて通信遅延のモデル化を行う．

さらに，提案モデル上で動作するタスクスケジューリングアルゴリズム SPPC(Scheduling with Packaged Point-to-point Communications) を提案する．SPPC は提案モデルにおけるオーバヘッドのパラメータを用いて，適切な通信の一括化を行う．SPPC の生成する並列スケジュールでは，プロセッサ間通信に自由度の高い一対一プロセッサ間通信を用いる．

ガウスジョルダン法と LU 分解の並列処理を用いて，SPPC と既存のアルゴリズムの性能比較を行った．結果より，SPPC は利用可能なプロセッサ数を有効に使うことができる並列スケジュールを生成するアルゴリズムであることがわかった．また，SPPC は適切な通信の一括化を選んで適用するアルゴリズムであると考えられる．

2 SPPC モデル

本研究では，古典的モデルにおける通信遅延の定義を拡張し，通信遅延のモデル化に LogP モデルを用いたタスクスケジューリングモデル（SPPC モデル）を提案する．LogP モデルでは，$o_s, o_r > g$ である並列計算機環境においては，g を省略できる [2]．SPPC モデルでは，$o_s, o_r > g$ であると仮定し，g をパラメータとして扱わない．

SPPC モデルにおける並列スケジュールの長さを**拡張メイクスパン**と呼ぶ．拡張メイクスパンは LogP モデル上で評価した並列スケジュールの長さである．また，通信遅延は通信量に関らず LogP モデルの各パラメータを用いて，$o_s + L + o_r$ として算出する．SPPC モデルでは並列スケジュール上の全ての通信の通信遅延は一定である．

タスクスケジューリングアルゴリズム SPPC
入力：タスクグラフ $G=(V,E,\lambda)$,
通信遅延 $D=\{L,o_s,o_r\}$,
プロセッサ集合 $P=\{P_0,\cdots,P_p\}$
出力：SPPC スケジュール SCH
1) G からクラスタグラフ G' を生成
2) G' を元に，同一プロセッサに割り当てる
クラスタの集合（クラスタセット）を決定
3) クラスタセットを元に SCH を生成

図 1: SPPC の概要

SPPC モデルでは，古典的モデルの並列スケジュールの定義に加えて並列スケジュール上に通信処理を明示する．通信処理は，送信処理と受信処理に分けて明示し，送信処理と受信処理は必ず一対一で対応が取れていなければならない．各通信毎に転送するデータ（タスクの実行結果）も明示する．送信処理（受信処理）の開始時刻から，$o_s(o_r)$ の間，送信処理を行うプロセッサは他のどのような処理も実行できない．

3 SPPC モデルと古典的モデルの差異

古典的モデルでは，タスク間で通信を実行すると考える．送信元タスク A からの 1 回の通信では A の結果しか送ることができない．また，A の実行完了直後に次のタスクの実行に移ることができると考える．

一方，SPPC モデルでは，A の実行が完了した時点で，A を実行するプロセッサが保持している全てのタスクの結果を一括して送信することが可能である．また，タスク A の実行完了後，送信処理を開始すると考え，次のタスクの実行可能開始時刻は o_s 後になる．

4 アルゴリズム SPPC

SPPC は並列スケジュール上のオーバヘッドの軽減を狙ったアルゴリズムである．オーバヘッドを削減する手法として，通信の一括化と依存関係のあるタスクを同一プロセッサに割り当てることで起きる通信数の削減を利用する．SPPC の概要を**図 1** に示す．

4.1 クラスタグラフの生成

タスクグラフは並列処理を重みつき有向グラフで表したものである．タスクグラフの各節点はタスクを表す．有向辺はタスク間の通信に対応する．**図 2** にタスクグラフの例を示す．図 2 では，簡単のために重みを省略している．

SPPC は，まず最初に，タスクグラフからクラスタを生成する．**クラスタ**はタスクグラフの任意の連結成分である．図 2 の破線で囲った各連結成分がそれぞれクラスタである．SPPC ではクラスタ単位で一つのプロセッサにタスクを割り当てるため，同一クラスタ内の各タスクは通信なしに実行できる．クラスタ生成の詳細については，紙面の都合上，省略する．

SPPC はタスクグラフを元に，クラスタを節点とした単純有向グラフ（**クラスタグラフ**）を生成する．**図 3** に図 2 のタスクグラフに対するクラスタグラフを示す．クラスタグラフの有向辺は，タスクグラフの複数の有向辺に対応する．このため，クラスタグラフの有向辺に対応する通信は，複数のタスク間の通信を一括化したものになる．

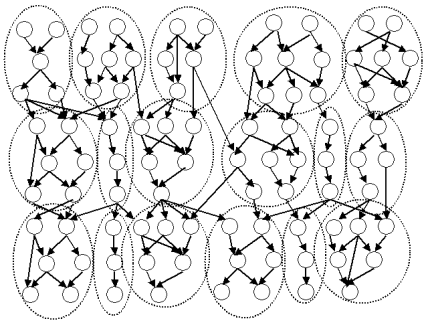

図 2: タスクグラフとクラスタ

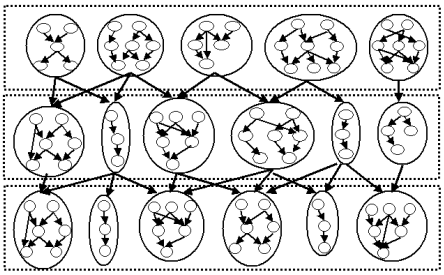

図 3: クラスタグラフ

4.2 クラスタセットの決定

SPPC はクラスタグラフを元にして，同一プロセッサに割り当てるクラスタの集合（**クラスタセット**）を決定する．クラスタグラフにおいて，クラスタから出次数が 0 であるクラスタへの各経路上に存在する有向辺の数の最大値を，そのクラスタの**レベル**と呼ぶ．また，レベルを基準にクラスタグラフの各クラスタを分類し，それぞれの集合を**レイヤ**と呼ぶ．SPPC は各レイヤ内のクラスタ間に有向辺が存在しないようにクラスタを生成する．このため，各レイヤのクラスタは並列実行可能である．図 3 の破線で囲ったクラスタ集合が，それぞれレイヤである．

4.2.1 クラスタ間親和度

並列処理に利用可能なプロセッサ数を P とすると，並列実行可能なクラスタ数は高々P である．まず，SPPC は各レイヤ内のクラスタを，各レイヤ毎に P 以下の**クラスタグループ**にまとめる．その際，SPPC はクラスタ間親和度を用いて，通信の一括化に効果の高いクラスタグループを生成する．**クラスタ間親和度**は，通信の一括化の効果を定式化した値で，大きければ大きいほど拡張メイクスパンの短縮に効果が大きいと考える．クラスタ C_p への有向辺を持つクラスタ集合を $Pred(C_p)$，C_p からの有向辺を持つクラスタ集合を $Succ(C_p)$ とする．$\rho(C_p,C_q)=|Pred(C_p)\cap Pred(C_q)|$, $\sigma(C_p,C_q)=$

$$
\begin{aligned}
A(C_p, C_q) &= o_r \times \rho(C_p, C_q) \\
&+ o_s \times \sigma(C_p, C_q) \\
&- \Lambda(C) \times S
\end{aligned}
$$

$$
C = \begin{cases} C_p & \textbf{if } |Pred(C_p)| > |Pred(C_q)| \\ C_q & \textbf{otherwise} \end{cases}
$$

$$
S = \begin{cases} 1 & \textbf{if } \psi(C_p, C_q) = 0 \\ min(\psi(C_p, C_q), P) & \textbf{otherwise} \end{cases}
$$

図 4: クラスタ間親和度

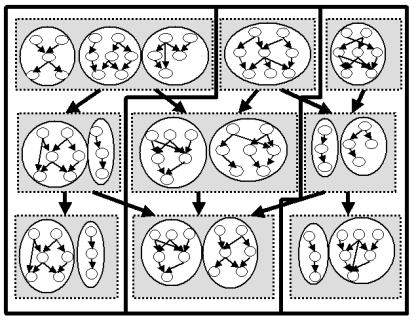

図 5: クラスタグループ

$|Succ(C_p) \cap Succ(C_q)|$，$\psi(C_p, C_q) = |Pred(C_p) \cup Pred(C_q)|$ とする．さらに，クラスタ C の重み $\Lambda(C)$ を次式で定義する．

$$
\begin{aligned}
\Lambda(C) &= |Pred(C)| \times o_r + |Succ(C)| \times o_s \\
&+ \sum_{t_j \in T(C)} \lambda(t_j)
\end{aligned}
$$

図 4 にクラスタ C_p と C_q のクラスタ間親和度 $A(C_p, C_q)$ の定義式を示す．

SPPC は最初にクラスタグラフの各クラスタを要素としたクラスタグループを生成する．クラスタグループをクラスタとみなして，クラスタ間親和度が最大であるクラスタ対を 1 つのクラスタグループにまとめる．これを繰り返し，全レイヤのクラスタグループを決定する．**図 5** の破線で囲った部分は，図 3 のクラスタグラフを元に決定した，各レイヤのクラスタグループである．クラスタグループ間の有向辺は，クラスタグループに属する複数のクラスタ間の通信を一括化した通信に対応する．

さらに，SPPC はレイヤの異なるクラスタグループを P 個のクラスタセットにまとめる．最初に P 個のクラスタセットを空集合で用意し，まず，最大レベルのレイヤの各クラスタグループを別々のクラスタセットの要素とする．次に，レベルが 1 つ小さいレイヤの各クラスタグループをそれぞれのクラスタセットに加えた場合の各クラスタセットの実行完了時刻を算出する．各クラスタセットの実行完了時刻に基づいて，各クラスタグループをまとめるクラスタセットを決定する．以下，レベルが 0 のレイヤに対する処理が完了するまで，同様の処理を続ける．このようにして，クラスタグラフから P 個のクラスタグループを得る．図 5 の実線で囲ったクラスタグループ集合が，それぞれクラスタセットである．

4.3 並列スケジュールの生成

各クラスタセットの中でレベルの高いレイヤに属するクラスタグループから順に，並列スケジュールを生成していく．並列スケジュール中で実行しなければならない通信は，クラスタセット間の有向辺に対応する．各レイヤのクラスタグループの実行は，高いレイヤに属するクラスタグループからの受信，低いレイヤに属するクラスタグループへの送信，クラスタ内のタスクの実行からなる．SPPC では，受信，送信，タスクの実行の順に優先度を付け，並列スケジュールを生成していく．これにより，できるだけ各プロセッサが受信待ちのアイドル状態にならないようにする．

5 評価実験

SPPC の並列スケジュールと BCSH[4] の並列スケジュールの性能を比較する．BCSH は，集合通信を用いた通信の一括化を行うアルゴリズムで，実行時間の短い並列プログラムを生成する．実験対象として，ガウスジョルダン法と LU 分解の並列処理を取り上げる．

評価基準として，次式で定義する拡張メイクスパンのスピードアップ S_m と，並列プログラム全体のスピードアップ S_e を用いる．$S_m(S_e)$ の値が大きいほど，並列スケジュール（並列プログラム）の並列化による性能向上が大きいことを表す．プロセッサ 1 台の場合の拡張メイクスパンは，タスクグラフ上のタスクの重みの総和に等しい．

並列プログラムの実行は，PC クラスタ上で行った．PC クラスタの構成は，ノードプロセッサ (PentiumII 450MHz, メモリ 512MB) 16 台を，Myrinet [6] ネットワークにより接続している．また，生成する並列プログラムは MPI ライブラリを使用した C 言語で記述する．使用した MPI ライブラリの実装は MPICH [5] である．並列プログラムは，我々の開発した並列スケジュール変換プログラムを用いて自動生成した．

LogP モデルの各パラメータは 1 代入文の平均実行時間を単位時間として算出した．評価実験では，予備実験から得られた平均的な通信量で通信を行う場合の測定値をパラメータとして与える．

5.1 実験結果と考察

ガウスジョルダン法（要素数 128），LU 分解（要素数 128）に対する，SPPC と BCSH のスピードアップ S_e の測定結果を**図 6**，**図 7** に示す．プロセッサ数の増加に伴い SPPC の S_e が上昇し，BCSH を上回る結果を得られた．また，図 6，図 7 どちらの場合も，SPPC の S_m と S_e は同様の傾向を示すことから，拡張メイクスパンを基準としたスケジューリングは有効であると言える．

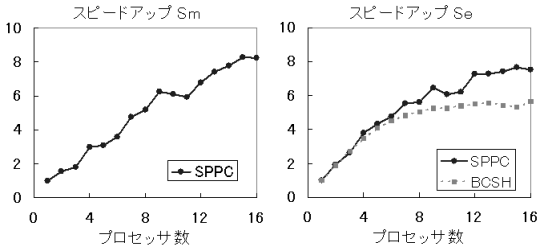

図 6: スピードアップ（ガウスジョルダン法）

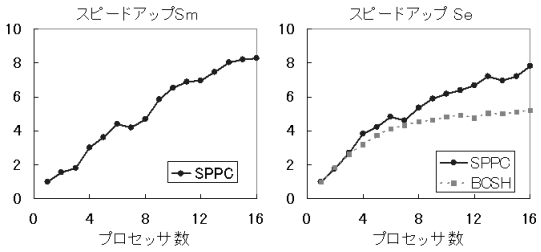

図 7: スピードアップ（LU 分解）

5.1.1 プロセッサ稼働率

並列スケジュールの拡張メイクスパンを ms，プロセッサ数を p，並列スケジュール中のアイドル時間の総和を I として，**プロセッサ稼働率**を次式で定義する．

$$\text{プロセッサ稼働率} = \frac{ms \times p - I}{ms \times p}$$

プロセッサ稼働率は，並列スケジュール全体に対してのタスクまたは通信処理を実行する時間の割合である．**図 8** に LU 分解についての，それぞれのプロセッサ稼働率を示す．

BCSH ではプロセッサ数が多くなると，**プロセッサ数節約** [4] を行い，並列処理に参加するプロセッサ数を制限する場合がある．プロセッサ数節約の結果，BCSH はプロセッサ数が多い場合に，必ずしも全てのプロセッサを使わない．一方，SPPC は並列スケジュールの並列度をプロセッサ数未満に下げることはない．図 8 より，プロセッサ数が多い場合に，SPPC はプロセッサを有効に利用できている．

5.1.2 オーバヘッド

SPPC スケジュールと BCSH スケジュール上のオーバヘッドの総和を，**図 9** に示す．プロセッサ数の増加に伴い，SPPC スケジュールのオーバヘッドが BCSH を上回る．オーバヘッドが BCSH に比べて多いにも関らず，図 6 では，SPPC のスピードアップ S_e が高い．これは，SPPC が通信の一括化を行うべきかどうかを適切に判断しているためと考えられる．

6 おわりに

本研究ではオーバヘッドを直接のパラメータとすることで，通信の一括化を定式化して扱えるタスクスケジューリングモデル SPPC モデルを提案した．SPPC モデルでは一対一通信に対する通信の一括化の効果

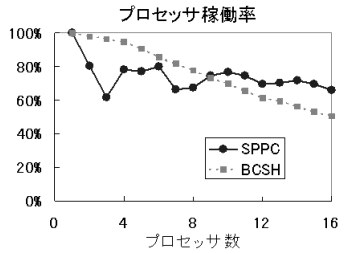

図 8: プロセッサ稼働率（LU 分解）

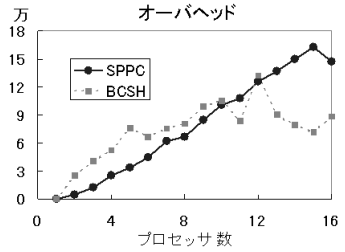

図 9: オーバヘッド（ガウスジョルダン法）

を定式化して扱うことができるため，既存の集合通信を利用した通信の一括化を行うモデルと比較して，並列スケジュールの自由度が高い．さらに本研究では，SPPC モデルに基づくアルゴリズム SPPC を提案し，SPPC は通信の一括化を利用することで，既存のアルゴリズム BCSH に比べて，性能の良い並列プログラムを生成できることがわかった．

謝辞 本研究の一部は，日本学術振興会未来開拓学術研究推進事業（JSPS-RFTF99I00903）および栢森情報科学振興財団の補助による

参考文献

[1] Bacon, D. F., Graham, S. L., and Sharp, O. J.: “Compiler transformations for high-performance computing”, acm computing surveys, Vol.26, No.4, pp.345–420, December (1994).

[2] Culler, D. E., Karp, R. M., Patterson, D. A., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T.: “LogP:Towards a Realistic Model of Parallel Computation”, 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, May (1993).

[3] Culler, D. E., Liu, L. T., Martin, R. P., and Yoshikawa, C. O.: “Assessing Fast Network Interfaces”, IEEE MICRO, Vol.16, No.1, pp.35–43, February (1996).

[4] Fujimoto, N., Baba, T., Hashimoto, T., and Hagihara, K.: “On Message Packaging in Task Scheduling for Distributed Memory Parallel Machines”, The International Journal of Foundations of Computer Science, Vol.12, No.3, pp.285–306 (2001).

[5] MPICH: A Portable MPI Implementation, http://www.mcs.anl.gov/mpi/mpich/

[6] Myrinet: Myricom, http://www.myri.com/

[7] 湯淺 太一，安村 通晃，中田 登志之：“はじめての並列プログラミング”，共立出版 (1998).