1対1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[††]

本論文では，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分かった．

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

Yuuki Yamamoto,[†] Lin Tan,[†] Noriyuki Fujimoto[††] and Kenichi Hagihara[††]

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 a famous and significant optimization technique for parallel programs executed 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 parallel programs faster than those generated by existing algorithms.

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，通信の一括化を行う必要がある[10]．通信の一括化は，分散メモリ型並列計算機におけるプロセッサ間通信処理のソフトウェアオーバヘッド（以下，オーバヘッド）を軽減する手法である．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，オーバヘッドの影響を無視しているため，古典モデル上で効果的な通信の一括化を考慮することは困難である．

これまで我々は，高性能並列プログラムを生成するタスクスケジューリングアルゴリズムBCSH[10]（Bulk Communication Scheduling Heuristic）を開発してきた．BCSHは独自のタスクスケジューリングモデル（以下，BCSHモデル）上で動作する．BCSHモデルに基づく並列スケジュールでは，全プロセッサが同期して通信を実行する．この際，BCSHは全対全通信[14]を用いて通信を実行し，通信の一括化を実現している．全対全通信は，複数のプロセッサ間の通信パターンである集合通信[16]の1種である．BCSHモデルは全対全通信の通信遅延に含まれるオーバヘッドの大まかな近似値を間接的にモデルのパラメータとして扱うことができる．しかし，BCSHモデルではオーバヘッドを直接のパラメータとしていないため，通信の一括化による効果を正確に扱うことができない．また，プロセッ

† 大阪大学大学院基礎工学研究科情報数理系専攻
Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
†† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

サ間通信が同期しなければならないという制約があるため，通信のタイミングの自由度が低い．

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[6]を用いて通信遅延をモデル化して扱う．

さらに，本研究では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ことができる．また，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 1 対 1 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 BCSH の性能比較を行った．生成した並列プログラムを，実際に並列計算機上で実行し測定した時間で性能比較を行った．評価実験より，並列処理に利用するプロセッサ数の増加にともない，SPPC の生成する並列プログラムの実行時間が BCSH を上回る結果を得た．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分かった．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．以上のことから，提案モデルは高性能プログラムの生成に有効であるといえる．

2. LogP モデル

LogP モデルは，並列計算の実行時間を少ないパラメータ数で比較的精度の高いモデル化を行うことができるバランスの良いモデルとして広く知られている．本研究では，提案モデルの並列スケジュールのコストモデルとして，LogP モデルを採用する．

LogP モデルの各パラメータを以下に示す☆．

- L：プロセッサ間通信遅延の最大値．メッセージがネットワーク上を移動する時間．
- o_s：送信処理時のオーバヘッドの最大値．
- o_r：受信処理時のオーバヘッドの最大値．
- g：連続送信（受信）した場合のインターバルの下限．
- P：プロセッサ数．

☆ 文献 5) では，オーバヘッドを単一のパラメータ o で扱うモデルとして提案されているが，後に文献 6) において，本論文中で紹介したパラメータに拡張されている．

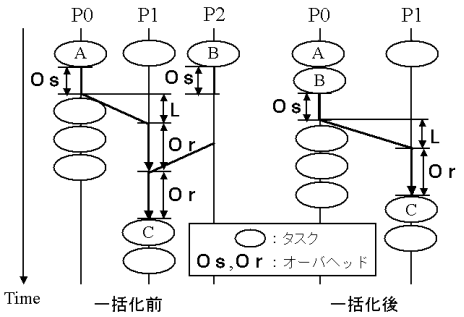


図 1 並列プログラムに対する通信の一括化の例
Fig. 1 Message packaging for a parallel program.

3. 通信の一括化

通信の一括化とは，複数回に分けて通信されていたデータを 1 回の通信にまとめることで通信回数を削減し，オーバヘッドを軽減する手法である[2),21)]．一般に，オーバヘッドは無視できない大きさであるため，並列プログラムの高速化手法として有効である．

図 1 左図は，3 台のプロセッサ P0，P1，P2 でタスク A，B，C を含む処理を実行する並列プログラムを LogP モデルに基づき模式化したものである．タスク A，B を別のプロセッサで実行するため，P1 への通信を 2 回行う．しかし，図 1 右図のように，タスク B を P0 で実行することで，通信を 1 回にまとめることができ，オーバヘッドの軽減につながる．この一括化により，タスク A，B の処理が完了してから通信を開始することになるため，送信開始時刻が遅くなるが，この例では，受信オーバヘッド o_r が 1 回で済むので，P1 で実行する他のタスクの実行時刻を遅れさせることなく，タスク C の実行時刻を早めることができる．

4. 古典的モデル

4.1 タスクグラフ

タスクを重み付きの節点，タスク間の先行制約を重み付き有向辺で表すことで，並列計算をモデル化したものが**タスクグラフ**である．タスクグラフは閉路のない重み付き有向グラフである．タスク間の先行制約は，タスク間にデータ依存が存在することにより生じる．タスクの重みはタスクの実行にかかる時間を，有向辺の重みはプロセッサ間の通信にかかる時間を表す．**図 2** にタスクグラフの例を示す．

4.2 並列スケジュール

並列スケジュールはタスクグラフの各タスクをいつどのプロセッサが実行するかを定義する．全タスクの実行完了時刻の中で，最も遅い時刻を並列スケジュー

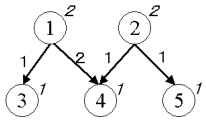


図 2 タスクグラフ
Fig. 2 A task graph.

ルの**メイクスパン**いう．並列スケジュール上で，プロセッサ Pi にタスクが割り当てられていない時間を Pi の**アイドルスロット**という．本論文では，並列スケジュールをガント図[9]で表す．

図 3 は，図 2 のタスクグラフで表される並列計算を 2 台のプロセッサ P0，P1 で実行する並列スケジュールを表したものである．図中の矢印はプロセッサ間通信を示す．P0 は時刻 0 にタスク 1 の実行を開始する．タスク 1 の重みが 2 なので時刻 2 にタスク 1 の実行が完了する．タスクグラフ上にタスク 1 からタスク 3 への有向辺が存在するが，P0 はタスク 1，3 の両方を実行するので，プロセッサ間通信によりタスク 1 の結果を送信する必要はない．このため，タスク 3 はタスク 1 の実行完了直後に実行を開始できる．タスク 1 からタスク 4 にも有向辺が存在するが，両タスクを異なるプロセッサで実行するため，タスク 1 の結果を P0 から P1 に通信しなければならない．タスク 4 の実行はタスク 1 の結果の受信を待って，時刻 4 に開始する．P0 の時刻 3 から 5，P1 の時刻 3 から 4 の間はアイドルスロットであり，この並列スケジュールのメイクスパンは 5 である．

4.3 古典的モデルの問題点

古典的モデルでは，プロセッサ間通信の送信元タスクの実行が完了した直後に次のタスクの実行を開始できると考える．たとえば，図 3 では，送信元タスクであるタスク 1 の実行直後にタスク 3 を実行できると考えている．しかし，実際には，送信側，受信側それぞれのプロセッサでオーバヘッドが発生し，オーバヘッドの間，プロセッサはタスクを実行できない．分散メモリ型並列計算機では，タスクの重みに比べて通信遅延が大きい．たとえば，タスクが実数演算 1 回であるとすると，並列計算機により異なるが，0 バイトのデータを通信する場合の通信遅延はタスクの重みの約 30 倍から約 28000 倍である[8]．さらに，通信遅延の中でオーバヘッドが占める部分の割合は約 29%から約 36%である[6]ため，古典的モデルではメイクスパンと実行時間の差が大きくなる[10]．このため古典的モデルで高速とされる並列スケジュールは，必ずしも高性能並列プログラムに対応しない．

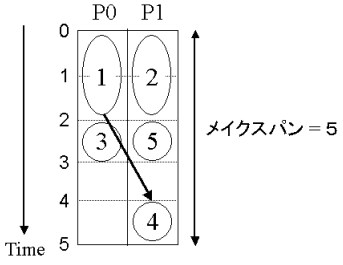


図 3 並列スケジュール
Fig. 3 A schedule.

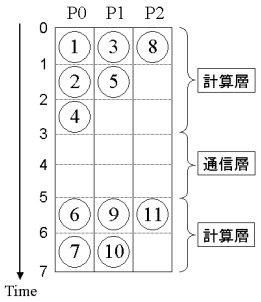


図 4 バルク同期スケジュール
Fig. 4 A bulk-synchronous schedule.

5. BCSH モデル

5.1 バルク同期スケジュール

通信の一括化を利用し，分散メモリ型並列計算機に適した並列スケジュールを生成するタスクスケジューリングアルゴリズムとして BCSH がある．BCSH はバルク同期スケジュール[20]を生成するアルゴリズムである．バルク同期スケジュールは計算層と通信層が交互に現れるという制約を課した古典的モデルに基づく並列スケジュールである．バルク同期スケジュールに対応する並列プログラムの計算層では，各プロセッサは他のプロセッサと独立してタスクの実行のみを行い，通信は行わない．一方，通信層では，通信のみを行い，タスクの実行は行わない．**図 4** にバルク同期スケジュールの例を示す．

5.2 全対全通信による通信の一括化

バルク同期スケジュールでは，通信処理を通信層内でしか実行しないため，すべての通信の送信開始時刻は計算層の終了時刻以降に延期される．そこで，BCSH は通信層の開始時刻ですべての通信を一括化し，全対全通信を行う．**図 5** 左図のバルク同期スケジュールの通信を一括化した例が図 5 右図である．図 5 右図に示すように，通信層における全対全通信では，並列処理

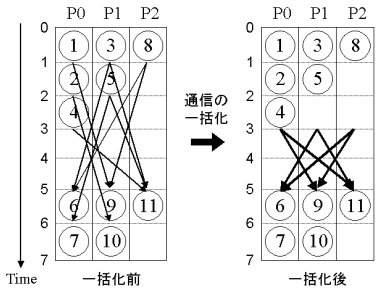


図 5 バルク同期スケジュールにおける通信の一括化
Fig. 5 Message packaging for a bulk-synchronous schedule.

に参加するすべてのプロセッサ間で通信を実行する．全対全通信では，全プロセッサが自身以外のプロセッサへそれぞれ異なるデータを送信する．

全対全通信内の各通信の通信遅延は，通信層の開始時刻から通信層の終了時刻までの時間に等しいと見なすことができる．通信層内ではタスクを実行しないため，オーバヘッドによりタスクの実行が妨げられることはない．よって，通信層の実行にかかる時間を通信遅延とすることで，オーバヘッドをモデルの直接のパラメータとして扱うことなく，通信の一括化を近似的にメイクスパンに反映できる．

6. SPPC モデルの提案

6.1 SPPC モデル

通信の一括化を適用すると，送信開始時刻の遅延により並列プログラムの実行時間の悪化を招く場合がある．そのため，メイクスパンの改善に効果がある通信の一括化を適切に選ぶ必要がある．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[5]．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を $SCH = (TASK, COMM)$ で表す．SCH はタスク命令集合 $TASK$ と通信命令集合 $COMM$ からなる．$TASK$ は，時刻 u，プロセッサ P_k，タスク t_a の 3 つ組 $\langle u, P_k, t_a\rangle$ の集合からなる．3 つ組 $\langle u, P_k, t_a\rangle$ は，時刻 u にタスク t_a の実行をプロセッサ P_k で開始することを表す．さらに，通信命令集合 $COMM$ により，通信処理を明示する．$COMM$ は，送信命令を表す 4 つ組 $S\langle u, P_s, P_d, T\rangle$ と，受信命令を表す 4 つ組 $R\langle u', P_d, P_s, T'\rangle$ の集合からなる．$S\langle u, P_s, P_d, T\rangle$ は，時刻 u に，プロセッサ P_s でプロセッサ P_d へタスク集合 T の結果を一括して送信する処理を開始することを表す．また，$R\langle u', P_d, P_s, T'\rangle$ は，時刻 u' に，プロセッサ P_d でプロセッサ P_s からタスク集合 T の結果を一括して受信する処理を開始することを表す．

SPPC モデルにおける並列スケジュールのメイクスパンを**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．拡張メイクスパンは，並列スケジュールから生成した並列プログラムを，実際に並列計算機上で実行するときにかかる時間を SPPC モデル上で評価した値である．また，通信遅延は通信量にかかわらず 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．つまり，SPPC モデルでは並列スケジュール上に明示されたすべての通信の通信遅延は一定である．

6.2 古典的モデルとの差異

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で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 1 対 1 で対応がとれていなければならない．たとえば，プロセッサ P0 から，プロセッサ P1 への送信処理が明示されているにもかかわらず，P1 で P0 からの受信処理が明示されていない並列スケジュールは SPPC モデルの制約に反する．また，**タスク複製**[13]により，同一のタスクを複数のプロセッサで実行することがある．そこで，複製されたタスクのうち，どのプロセッサで実行するどのタスクの結果をどの通信で通信するかを陽に示すために，各通信ごとに転送するデータ（タスクの結果の集合）を明示する．これにより，1 回の通信で複数のタスクの結果を一括して転送することを表現できる．

通信遅延は LogP モデルに基づくため，送信処理開始から受信完了までの時間は，少なくとも $o_s + L + o_r$ 以上である．図 1 の例のように，一括化した通信に関しても，一括化前と同様の通信遅延で扱う．さらに，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（受信処理）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．

以下では，図 1 左図の並列プログラムに対応する並列スケジュールを例にして，古典的モデルと SPPC モデルの差異を説明する．古典的モデルでは，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．一方，SPPC モデルでは，A の実行が完了した時点で，P0 が保持しているすべ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．古典的モ

デルでは，通信処理は他のタスクや通信処理の実行に影響を与えないと考える．そのため，P1 は P0 と P2 からの受信を同時に実行できると考える．SPPC モデルでは，受信処理は 1 つずつ明示され，同時に実行できないと考える．このため，それぞれの受信処理に続く処理は，o_r 後の時刻になる．また，古典的モデルでは，タスク 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する．一方，SPPC モデルでは，タスク A の実行完了後，送信処理を開始するため，並列スケジュール上に送信処理を明示し，次のタスクの実行開始は o_s 後の時刻になるとする．

7. クラスタグラフ

7.1 クラスタグラフの定義

SPPC モデルにおける，タスクグラフを $G = (V, E, \lambda)$ と書く．ここで，V は節点の集合，E は有向辺の集合，λ は節点から節点の重みへの関数である．節点 u から節点 v への有向辺を (u, v) と書き，タスク v をタスク u の**直接後続タスク**と呼ぶ．SPPC モデルでは，通信遅延は通信量によらず一定である．そのため，有向辺の重みはタスクグラフのパラメータとして扱わない．また，タスク v からタスク u への**最大経路長**とは，v から u への各経路上に存在する有向辺の数の最大値である．出次数が 0 であるタスクからタスク u への最大経路長を u の**タスクレベル**と呼び，$Tlv(u)$ と表す．定義より，出次数が 0 であるタスクのタスクレベルは 0 である．また，タスクレベルを基準にタスクを分類し，それぞれのタスク集合を**タスクレイヤ**と呼ぶ．$Tlv(u) = i$ のタスク u が属するタスクレイヤを TL_i と表す．

タスクグラフ $G = (V, E, \lambda)$ の任意の誘導部分グラフを**クラスタ**と呼ぶ．**図 6** にタスクグラフとクラスタの例を図示した．図 6 中の破線で囲った有向グラフそれぞれが，クラスタである．

C_j と C_k をそれぞれ G のクラスタとする $(C_j \neq C_k)$．$v \in C_j$ かつ $u \in C_k$ かつ $u \notin C_j$ である (u, v) が E に存在するとき，かつそのときに限り C_j は C_k に**直接依存する**という．さらに，以下の条件 C1 または C2 が満たされるとき，C_j は C_k に**依存する**という．

(C1) C_j は C_k に直接依存する．

(C2) C_j が C_i に直接依存し，かつ C_i が C_k に依存する G のクラスタ C_i が存在する．

G のクラスタの集合 $\mathcal{C}$ が $\cup_{(V_c, E_c, \lambda) \in \mathcal{C}} V_c = V$ を満たすとき $\mathcal{C}$ は G を**被覆する**という．G を被覆するクラスタ集合 $\mathcal{C}$ に属するクラスタを節点とし，$C_i \in \mathcal{C}$ が $C_j \in \mathcal{C}$ に直接依存するとき，かつそのときに限り，有向辺 (C_i, C_j) が存在する単純有向グラフのうち，閉路のないグラフを G の**クラスタグラフ**という．図 6 に図示したタスクグラフのクラスタグラフのの 1 つを**図 7** に図示する．

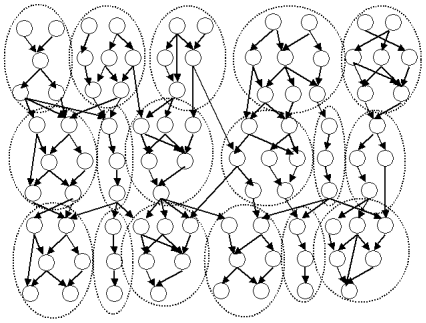


図 6 タスクグラフとクラスタ
Fig. 6 A task graph and clusters.

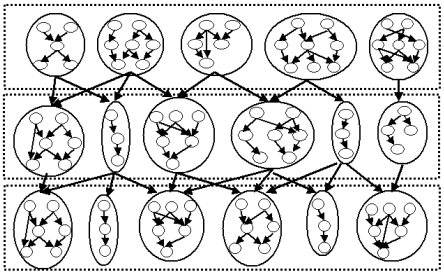


図 7 クラスタグラフ
Fig. 7 A cluster graph.

クラスタグラフ G' を $G' = (V', E', \Lambda)$ で表す．V' は節点の集合を表し，E' は有向辺の集合を表す．例 Λ は節点から節点の重みへの関数で，クラスタの重みを表す．クラスタの**重み**はクラスタの実行にかかる時間をモデル化した値である．そこで，Λ は，送受信時のオーバヘッドを考慮し，次式で算出する．$T(C)$ はクラスタ C のタスクの集合を表す．

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

クラスタグラフの有向辺は，タスクグラフの有向辺の集合を表している．つまり，クラスタグラフの有向辺に対応した通信は，タスクグラフ上の複数の有向辺にそれぞれ対応する通信に対し，通信の一括化を適用した通信に対応する．また，各クラスタ内のすべてのタスクを同一プロセッサで実行することで，クラスタ内の有向辺に対応する通信は実行しなくてよい．

出次数が 0 であるクラスタからクラスタ C_j への複数の経路長のうち，最大値を C_j の**レベル**と呼び，

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

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

$$
S = \begin{cases} 1 & \textbf{if } Succ(C_p) \cup Succ(C_q) = \phi \\ min(|Succ(C_p) \cup Succ(C_q)|, \mathrm{P}) & \textbf{otherwise} \end{cases}
$$

図 8 クラスタ間親和度
Fig. 8 Affinity between C_p and C_q : $A(C_p, C_q)$.

$lv(C_j)$ と表す．定義より，出次数が 0 のクラスタのレベルは 0 である．レベルで分けたクラスタ集合を**レイヤ**と呼ぶ．図 7 中の破線で囲ったクラスタ集合がそれぞれ，レイヤである．$lv(C) = i$ のクラスタ C が属するレイヤを L_i と表す．$C_k(C_j)$ が $C_j(C_k)$ に直接依存するとき，$C_j(C_k)$ を $C_k(C_j)$ の**直接先行（後続）クラスタ**と呼ぶ．C_j が C_k に依存するが直接依存はしないとき，$C_j(C_k)$ を $C_k(C_j)$ の**間接先行（後続）クラスタ**と呼ぶ．また，クラスタ C の直接先行クラスタ集合を $Pred(C)$，直接後続クラスタ集合を $Succ(C)$ と表す．

7.2 クラスタ間親和度

クラスタグラフ上のクラスタ C_p，C_q に対して，C_p の直接先行クラスタ集合と C_q の直接先行クラスタ集合の共通要素数を**先行クラスタ親和度**$\rho(C_p, C_q)$ と呼ぶ．同様に，直接後続クラスタ集合の共通要素数を**後続クラスタ親和度**$\sigma(C_p, C_q)$ と呼ぶ．通信の一括化の効果を表す値として，**クラスタ間親和度**を定義する．クラスタ間親和度が大きければ大きいほど，そのクラスタ対からの通信に対する一括化の効果が大きいことを表す．クラスタ C_p，C_q のクラスタ間親和度 $A(C_p, C_q)$ の定義式を**図 8** に示す．

先行クラスタ親和度は，C_p，C_q を同一プロセッサに割り当てた場合における受信処理の削減数を表す．C_p と C_q を別々のプロセッサに割り当てた場合，それぞれの直接先行クラスタから結果を受信するため，最悪時に $|Pred(C_p)| + |Pred(C_q)|$ 回の受信を実行する．一方，C_p と C_q を同一プロセッサに割り当てることで，共通の直接先行クラスタからの通信を一括化できるため，受信処理は $|Pred(C_p)| + |Pred(C_q)| - |\rho(C_p, C_q)|$ 回に削減できる．この先行クラスタ親和度と受信時のオーバヘッドの重みの積をとることで，C_p と C_q を同一プロセッサに割り当てた場合の受信オーバヘッドに対する通信の一括化の効果を表現することができる．つまり，クラスタ間親和度の定義式の第 1(2) 項で受信（送信）オーバヘッドに対する一括化の短縮効果を定式化している．

また，第 3 項は通信の一括化による拡張メイクスパンの増分を定式化している．C_p と C_q を同一プロセッサに割り当てて C_p と C_q からの通信を一括化する場合は，C_p を単独で処理する場合と比較して C_q の処理が加わる分，C_p からの通信を受信する直接後続クラスタ集合の最悪時通信完了時刻が遅延する．**最悪時通信完了時刻**とは，同一プロセッサに割り当てた，あるクラスタ集合に属するすべてのクラスタの実行が完了した後に通信を行う場合の通信完了時刻である．1 対のクラスタを同一プロセッサに割り当てた場合，どちらのクラスタを先に実行するかを判定するために，それぞれの直接先行クラスタの数を算出する．経験的に直接先行クラスタの数が大きいクラスタの方が，実行時刻が遅くなる傾向があるので，直接先行クラスタの数を用いてクラスタの処理順序を決定する．ここでは，後で実行するクラスタの重みの分だけ，後続クラスタの最悪時通信完了時刻が遅延すると考える．直接後続クラスタは C_p と C_q の結果を受け取るため，最悪時通信完了時刻が遅延する．直接後続クラスタ集合は $Succ(C_p) \cup Succ(C_q)$ である．つまり，$|Succ(C_p) \cup Succ(C_q)|$ 個の直接後続クラスタが遅延の影響を受ける．しかし，実際には P 台のプロセッサで並列実行するため，遅延される直接後続クラスタ集合の要素数はたかだか P になる．また，直接後続クラスタ集合が空集合の場合も，C_p と C_q を同一プロセッサに割り当てることで，そのプロセッサで実行する C_p，C_q 以降の処理が遅延するので，遅延する直接後続クラスタ集合の要素数は少なくとも 1 であると考える．

8. アルゴリズム SPPC

8.1 SPPC の概要

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の削減を利用する．

SPPC の概要を以下に示す．

入力：タスクグラフ $G = (V, E, \lambda)$**，**
通信遅延 $D = \{L, o_s, o_r\}$**，プロセッサ数** P**，**
プロセッサ集合 $Pr = \{P_1, \cdots, P_p\}$
出力：並列スケジュール SCH

(1) 8.1.1 項のクラスタグラフ生成アルゴリズムを用い，クラスタグラフを生成する．
(2) 8.1.2 項のクラスタグループ生成アルゴリズム

を用い，同一プロセッサに割り当てるクラスタグループを決定する．

(3) 8.1.3 項のクラスタグループ割当てアルゴリズムを用い，各クラスタグループを割り当てるプロセッサを決定する．

(4) 8.1.4 項の並列スケジュール生成アルゴリズムを用い，並列スケジュールを生成する．

8.1.1 クラスタグラフ生成アルゴリズム

このアルゴリズムでは，タスクグラフをクラスタグラフに縮約することで，タスク間の通信の一括化を狙っている．また，グラフ上の節点数の削減により，以降の処理の計算量を低減することができる．**図 9** にアルゴリズムの詳細を示す．

SPPC は，まず，タスクレベルが 0 のタスクを各要素とするクラスタを生成する．次に，1 つ大きいタスクレベルのタスクレイヤに属する各タスクを，そのタスクの直接後続タスクが属するクラスタに加える．たとえば，あるタスク t_1 の直接後続タスク t_2 と t_3 が，それぞれ t_2 はクラスタ C_2 に，t_3 はクラスタ C_3 に属する場合を考える．SPPC は，t_1 を C_2 と C_3 の要素に加える．t_1 を加えることで，C_2 や C_3 は新たな誘導部分グラフとなる．SPPC はタスクを加える際に，t_1 のように別々のクラスタに属する複数の直接後続タスクを持つタスクを複製して，それぞれのクラスタに加える．この処理により，クラスタ内に含む有向辺に対応する通信を削減する．以降，SPPC はさらに大きいタスクレベルに属するタスクに対し，タスク複製を利用しながら，クラスタを生成していく．

タスク複製には通信を削減する効果があるが，実行するタスクの総量が増えるため，過度の複製は計算量の増大を招く．SPPC は複製タスクの数を調整するために，クラスタ対のマージを行う．クラスタグラフ $G' = (V', E', \Lambda)$ のクラスタ C_i と C_j の**マージ**とは，$T(C_i) \cup T(C_j)$ の誘導部分グラフに対応するクラスタを新たにクラスタ集合 V' の要素とし，マージ前のクラスタ対 C_i と C_j を V' から取り除くことをいう．複製タスク量はクラスタをマージすることで削減できるが，同時にクラスタグラフの並列性も削減される．そこで，SPPC では，クラスタ C_i と C_j をマージするかどうかの判定を行い，適切なクラスタ対に対して，マージを行う．マージによる複製タスクの削減は，以下の方法で行う．

マージ前のクラスタ集合を $\mathcal{C}_a$ とする．$\mathcal{C}_a$ に含まれる任意の 1 対のクラスタをマージすることで得られるクラスタの集合を $\mathcal{C}_b$ とする．以下の説明では，クラスタ対 C_x，C_y をマージする場合を考える．クラスタ

クラスタグラフ生成アルゴリズム
入力：タスクグラフ $G = (V, E, \lambda)$
通信遅延 D**，プロセッサ集合** Pr
出力：クラスタグラフ $G' = (V', E', \Lambda)$
/* タスクレベルの最大値*/
$l \leftarrow max\{Tlv(t_j) | t_j \in V\}$
クラスタ集合 $\mathcal{C}$，$\mathcal{C}'$
$i \leftarrow 0$
while $i \leq l$ **do begin**
/* クラスタ生成の開始 */
$k \leftarrow 0; \mathcal{C} \leftarrow \cup_{v \in TL_i}\{\{v\}\}$
repeat
$k \leftarrow k + 1; \mathcal{C}' \leftarrow \mathcal{C}$
/* 最大タスクレベルで，クラスタ生成を終了 */
if $i + k > l$ **then goto** L1
/* タスク複製を利用したクラスタの生成 */
タスクレイヤ TL_{i+k} に属する各タスクを
その直接後続タスクが属するクラスタ ($\in \mathcal{C}$) に加える
$\mathcal{C}$ 中のクラスタをマージして複製タスクを減らす
until $\mathcal{C}$ の負荷バランスが悪い
L1: $\mathcal{C}'$ の各クラスタを V' に加える
$i \leftarrow i + k - 1$
end /* while ループの本体終わり */
V' のクラスタ間の依存関係を判定し E' を生成する

図 9 クラスタグラフ生成アルゴリズム
Fig. 9 An algorithm to generate a cluster graph.

を独立した 1 つのタスクと見て，$\mathcal{C}_a$，$\mathcal{C}_b$ を，それぞれ P 台のプロセッサに，Graham のアルゴリズム LDSH (List-Decreasing Scheduling Heuristic)[11] を用いてスケジューリングする．生成した並列スケジュールの拡張メイクスパンをそれぞれ $LM(\mathcal{C}_a, P)$，$LM(\mathcal{C}_b, P)$ とする．$LM(\mathcal{C}_a, P) \geq LM(\mathcal{C}_b, P)$ ならば，SPPC は C_x と C_y をマージする．そうでなければマージしない．クラスタ C_x，C_y は以下の順で選ぶ．まず，タスクの重みの総和の昇順に C_x を選び，選んだ C_x に対して，$T(C_x) \cap T(C_y)$ のタスクの重みの総和の降順に C_y を選ぶ．ただし，1 つのクラスタは，たかだか 1 回しかマージしない．

SPPC はタスク複製を利用して，タスクレベルの低いタスクから順にクラスタを生成していく．クラスタ生成の際に，各クラスタの重みの差が大きくなることがある．クラスタの重みの差が大きくなると，重みの大きいクラスタの結果を通信する際に，受信プロセッサで受信待ち時間が発生し，並列プログラムの実行時間が悪化する．そこで，SPPC はクラスタの重みのバランス（以下，負荷バランス）を判定し，負荷バランスが良い場合は，タスク複製によるクラスタの生成をさらに大きいタスクレベルまで続行する．一方，負荷バランスが悪いと場合は，処理中のタスクレベルの 1 つ小さいタスクレベルまでで，いったん，クラスタの生成を中断する．ここまでの処理で生成したクラスタ

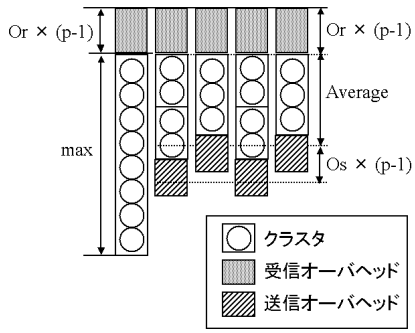


図 10 クラスタ集合の負荷バランス
Fig. 10 Balance of a set of clusters.

集合を V' の要素とする．中断後は，クラスタの生成が完了したタスクレベルより 1 つ大きいタスクレベルに属するタスクを各要素とするクラスタを生成し，クラスタ生成処理を同様に再開する．

負荷バランスが悪いと判定する例を，**図 10** に示す．重みが最大のクラスタに含まれるタスクの重みの総和を max，最大のクラスタを除く残りのクラスタに含まれるタスクの重みの総和を $amount$ とする．各クラスタを P 台のプロセッサで実行する場合を考える．簡単のため，各クラスタの実行開始時刻を 0 とする．重みが最大のクラスタからの送信は行わない場合でも，重みが最大のクラスタを実行するプロセッサがクラスタの処理をすべて完了する時刻は，遅くとも $o_r \times (P-1) + max$ になる．また，残りのクラスタに属するすべてのタスクを，均等に残りの $P-1$ 台のプロセッサに割り当てたとすると，それぞれのプロセッサがタスクを実行する時間は $Average = amount/(P-1)$ になる．残りのプロセッサが，そのプロセッサ自身以外のプロセッサ $P-1$ 台に対し，通信を行うと，送信処理に $o_s \times (P-1)$ の時間がかかる．よって，この $P-1$ 台のプロセッサが，送信処理，受信処理含めたクラスタの処理をすべて完了する時刻は，遅くとも $o_r \times (P-1) + Average + o_s \times (P-1)$ になる．これより，少なくとも $max - (Average + o_s \times (P-1))$ のアイドルスロットが発生すると考え，$max - (Average + o_s \times (P-1)) > 0$ の場合は，バランスが悪いと判定する．

SPPC は，ここまでの処理で，タスクグラフからクラスタ集合を生成する．次に，クラスタ間の依存関係を以下の方法で判定し，クラスタグラフの有向辺の集合を生成する．

SPPC はクラスタ生成時にタスク複製を利用するので同一タスクが複数のクラスタに存在することがある．このため，直接（間接）先行クラスタの候補は複数存在する．しかし，特定の先行クラスタに有向辺が集中すると，そのクラスタを実行するプロセッサからの通信が増加し，負荷分散の観点から好ましくない．そこで，SPPC は最も重みの小さいクラスタを先行クラスタとして選択する．この選択した先行クラスタとの間に対してのみ，有向辺を生成する．

8.1.2 クラスタグループ生成アルゴリズム

クラスタグループとは，同一のプロセッサで実行するクラスタ集合のことで，レイヤごとに生成する．レイヤの定義より，同一レイヤに属するすべてのクラスタは並列実行可能である．各レイヤの要素数をそのレイヤの**並列度**という．8.1.1 項のアルゴリズムが生成するクラスタグラフでは，レイヤの並列度が P より大きくなることがある．しかし，実際に並列処理に利用できるプロセッサ数は P であるため，並列度が P より大きいレイヤでは，1 つのプロセッサに複数のクラスタを割り当てなければならない．そこで，このアルゴリズムでは，各レイヤごとに最大 P 個のクラスタグループを生成し，同一プロセッサに割り当てるクラスタ集合を決定する．

SPPC は，まず，各クラスタを要素とするクラスタグループを生成する．以降の処理は，その時点での並列度が P より大きいレイヤに対してのみ実行する．次に，クラスタ間親和度が最大であるクラスタ対を選んでマージする．クラスタグラフ G' 上のクラスタ C_j と C_k をマージする場合，C_j と C_k の要素の和集合を新たに C_j とし，C_k および C_k を始点あるいは終点とする有向辺を G' から削除することで，G' を更新する．さらに，C_j の属するクラスタグループと，C_k の属するクラスタグループの 1 つの和集合にまとめる．1 回のマージにより，マージの対象となるクラスタ対が属していたレイヤの並列度とクラスタグループ数が，それぞれ 1 減少する．以降，マージを実行しなければならないレイヤがなくなるまで，以上のクラスタ対の選択とマージを続ける．アルゴリズムの詳細は，**図 11** に示す．

8.1.3 クラスタグループ割当てアルゴリズム

このアルゴリズムでは，クラスタグループをそれぞれどのプロセッサに割り当てるかを決定する．ある 1 つのクラスタの実行時刻が遅くなると，通信を介して，以降のレイヤのクラスタにまで影響が出る．そこで，各レイヤに属するクラスタグループの実行完了時刻が最悪の場合になることを避ける方針をとる．**図 12** に，アルゴリズムの概要を示す．

クラスタグループ γ_k の**最悪時実行完了時刻** $WT(\gamma_k, P_j)$ は，γ_k をプロセッサ P_j に割り当てて

```
クラスタグループ生成アルゴリズム
 入力：クラスタグラフ G' = (V', E', Λ)，通信遅延 D，
      プロセッサ数 P
 出力：クラスタグループ集合Γ
 レイヤ数 l ← max{lv(C_j)|C_j ∈ V'}
 Γ ← φ
 foreach C_p ∈ V' do
  /* γ_p は C_p が属するクラスタグループ */
  γ_p ← {C_p}
 while |L_i| > P であるレイヤが存在 do begin
  |L_i| > P であるレイヤ集合全体から
  クラスタ間親和度が最大である (C_p, C_q) を探す
  C_p と C_q をマージして G' を更新する
  /* クラスタグループの和集合をとる */
  γ_p ← γ_p ∪ γ_q
 end /* while ループの本体終わり */
 for i = 0 to l do begin
  foreach C_j ∈ L_i do
   Γ ← γ_j ∪ Γ
 end /* 外側の for ループの本体終わり */
```

図 11 クラスタグループ生成アルゴリズム
Fig. 11 An algorithm to generate cluster groups.

γ_k のクラスタの直接先行クラスタからのすべての通信を最悪時通信時刻に行う場合に，γ_k の実行が完了する時刻を示す．以下，プロセッサ P_q で実行するクラスタグループを $CS(P_q)$ で表し，**クラスタセット**と呼ぶ．

SPPC は，レベルの最も大きいレイヤのクラスタグループから順に，割当てを決定していく．まず，レイヤ内の各クラスタグループを Pr の各プロセッサに割り当てた場合の最悪時実行完了時刻を算出する．各クラスタグループにとって，最悪時実行完了時刻が最も早いプロセッサを見つけ，割当ての候補とする．このとき，あるクラスタグループ γ_k が割当て候補として選んだプロセッサ P_i が，他のクラスタグループの割当て候補でない場合は，$CS(P_i)$ の要素に γ_k を加える．そうでない場合は，そのような複数のクラスタグループの中で，P_i に割り当てたときの最悪時実行完了時刻が最も遅いものを $CS(P_i)$ に加える．続いて，まだ割当ての決まってないクラスタグループとプロセッサに対して，同様の処理を繰り返し，クラスタセットを決定していく．以上の処理を，各レイヤに対して実行し，クラスタセットを決定する．アルゴリズムの詳細を図 12 に示す．

8.1.4 並列スケジュール生成アルゴリズム

ここでは，クラスタセット集合を元に並列スケジュールを生成する．その際，通信はレイヤの異なるクラスタグループ間で行うように，並列スケジュールを生成する．クラスタグループ間の通信は，クラスタグラフの複数の有向辺に対応した通信である．これにより，

```
クラスタグループ割当てアルゴリズム
 入力：クラスタグラフ G'，クラスタグループ集合 Γ，
      通信遅延 D，プロセッサ集合 Pr
 出力：クラスタセット集合 {CS(P_x)|P_x ∈ Pr}
 レイヤ数 l ← max{lv(C_j)|C_j ∈ V'}
 foreach P_p ∈ Pr do
  CS(P_p) ← φ /* 空集合に初期化 */
 for i = l downto 0 do begin
  /* 割当てを決定するクラスタグループ集合 */
  L' ← L_i /* 割当て前のクラスタグループ集合 */
  Pr' ← Pr /* 割当て候補となるプロセッサ集合 */
  foreach P_k ∈ Pr do
   /* P_k を割当て候補とするクラスタグループ集合 */
   CC_k ← φ
  while L' ≠ φ do begin
   foreach C_p ∈ L' do begin
    WT(C_p, P_k) が最小となる P_k ∈ Pr' を見つける
    CC_k ← CC_k ∪ {C_p}
   end /* foreach ループの本体終わり*/
   foreach P_k ∈ Pr' do begin
    if |CC_k| = 1 then begin
     CS(P_k) ← CC_k ∪ CS(P_k)
     /* 割当て決定済を除去 */
     L' ← L' − CC_k ; Pr' ← Pr' − {P_k}
    end /* if 文の終わり */
    else if |CC_k| > 2 then begin
     CC_k の中で，WT(γ_a, P_k) が最大である
     クラスタグループ γ_a を見つける
     CS(P_k) ← {γ_a} ∪ CS(P_k)
     /* 割当て決定済を除去 */
     L' ← L' − {γ_a} ; Pr' ← Pr' − {P_k}
    end /* else if 文の終わり */
   end /* foreach ループの本体終わり*/
  end /* while ループの本体終わり*/
 end /* 外側の for ループの本体終わり */
```

図 12 クラスタグループ割当てアルゴリズム
Fig. 12 An algorithm to assign cluster groups to processors.

通信の一括化を実現する．

並列スケジュールの生成は，レベルの大きいレイヤから順に，各プロセッサごとに実行する．並列スケジュール生成の際には，受信命令，送信命令，タスク命令をそれぞれ別々の順序リストにより管理する．順序リストは，優先度の降順に整列されるものとする．各順序リストの優先度は以下の方法で決定する．

受信命令には，受信可能時刻の早いものから順に高い優先度を与える．**受信可能時刻**とは，対応する送信命令の送信時刻を u とすると，$u+o_s+L$ であるとする．つまり，受信処理の開始が可能になる時刻を表す．

送信命令には，以下の方法で優先度を与える．送信可能になるために実行が必要なクラスタの重みの総和が小さい送信に順に，高い優先度を与える．**送信可能**とは，送信するタスクの結果のすべてを，送信処理を

```
並列スケジュール生成アルゴリズム
 入力：クラスタグラフ G′，通信遅延 D
     クラスタセット集合 {CS(P_x)|P_x ∈ Pr}
     プロセッサ集合 Pr
 出力：並列スケジュール SCH
レイヤ数 l ← max{lv(C_j)|C_j ∈ V′}
TASK, COMM ← φ
foreach P_k ∈ Pr do begin
 プロセッサ時刻 u(P_k) ← 0
 for i = 0 to l do
  受信命令順序リスト Recv(P_k, i) ← φ
end /* foreach ループの本体終わり */
送信命令順序リスト Send，クラスタ順序リスト CL
for i = l downto 0 do begin
 foreach P_k ∈ Pr do begin
  Send, CL ← φ
  foreach P_j ∈ Pr − {P_k} do begin
   プロセッサ P_j への送信命令を生成する
   送信内容の優先度を決定する
  CS(P_k) の各クラスタの優先度を決定する
  while Recv(P_k, i), Send, CL の 1 つが空でない do
   if u(P_k) が Recv(P_k, i) の先頭の受信命令の
   受信可能時刻を過ぎている then begin
    先頭命令を COMM に加え，Recv(P_k, i) から除去
    実行可能になった CS(P_k) 内のクラスタを
    CL に挿入し，CS(P_k) から除去
    u(P_k) ← u(P_k) + o_r
   end /* 受信命令の生成完了 */
   else if Send が空でない then begin
    先頭の送信命令を COMM に加え，Send から除去
    Recv(P_j, i − 1) に対応する受信命令を挿入
    u(P_k) ← u(P_k) + o_s
   end /* 送信命令の生成完了 */
   else if CL が空でない then begin
    先頭のクラスタ C_q を CL から除去する
    foreach タスク t_a ∈ C_q do begin
     if P_k が t_a の結果を持たない then begin
      TASK に t_a の実行命令を加える
      u(P_k) ← u(P_k) + λ(t_a)
     end /* if 文の終わり */
    end /* タスク命令の生成完了 */
    実行可能になった CS(P_k) 内の各クラスタを
    CL に挿入し，CS(P_k) から除去
    送信可能になった各送信内容を Send に挿入
   end /* クラスタ C_q のスケジューリング完了 */
   else u(P_k) を次の受信可能時刻まで進める
  end /* プロセッサ P_k に対する処理完了 */
 end /* レイヤ i に対する処理完了 */
end /* 全レイヤに対する処理完了 */
SCH ← (TASK, COMM)
```

図 13 並列スケジュール生成アルゴリズム
Fig. 13 An algorithm to generate SPPC schedule.

実行するプロセッサが保持している状態をいう．

クラスタの優先度は，優先度の高い送信命令で送信するタスクを含むクラスタに高い優先度を与える．複数の送信命令に関係するクラスタの場合は，その中で最も優先度が高い送信命令に基づき，クラスタの優先度を決定する．

アルゴリズムの詳細を**図 13** に示す．図 13 内の表記で，**プロセッサ時刻** $u(P_k)$ は，プロセッサ P_k の並列スケジュールをどの時刻まで生成したかを表す．ここでは，3 つの順序リストを更新しながら，並列スケジュールを生成していく．受信命令は，1 つレベルの大きいレイヤでの送信命令をスケジューリングした際に，順序リストに挿入する．また，送信命令は，受信命令，クラスタをスケジューリングした際に，各送信命令について送信可能かどうかの判定を行い，送信可能と判定した送信命令を順序リストに挿入する．送信命令と同様に，クラスタについても，受信命令，クラスタをスケジューリングした際に，各クラスタについて実行可能かどうかの判定を行い，実行可能と判定したクラスタを順序リストに挿入する．**実行可能**とは，クラスタの実行に必要なタスクの結果のすべてを，クラスタを実行するプロセッサが保持している状態をいう．

3 つの順序リスト間では，受信命令，送信命令，クラスタの順で，優先してスケジューリングする．まず，受信命令を優先することで，受信したタスクの結果を使って，実行するクラスタを早い時刻に実行可能にする．実行可能状態にあるクラスタ数が多ければ，次に実行するクラスタの選択肢が広くなり，クラスタの優先度が意味を持つ．次に，送信を優先することで，受信待ち時間の低減を狙っている．

9. 評価実験

本実験では，SPPC の並列スケジュールと BCSH の並列スケジュールの性能を比較する．既存の研究ではタスクスケジューリングアルゴリズムの性能比較に用いられるタスクグラフは文献ごとに異なっている[12]．性能比較を同一の条件下で行うためにタスクスケジューリングアルゴリズムのためのベンチマークデータとして標準タスクグラフセット[12]が提案され，タスクグラフのデータが公開されつつある．しかし，本論文の執筆時点で公開されているタスクグラフは通信遅延が 0 のグラフのみであり，SPPC が対象とする通信遅延が非 0 のタスクグラフは公開されていない．そのため本論文では，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．そこで，本実験で用いたタスクグラフでは，並列性を高めるために，1 代入文を 1 タスクとして扱う．各タスクグラフのタスク数は，ガウスジョルダン法で約 107 万，LU 分解で約 70 万である．

9.1 評価項目

SPPC と BCSH の評価基準として，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 はSPPCモデル上での，並列化による高速化の効果を表す基準である．S_e は生成した並列プログラムの並列化による実行時間の短縮効果を表す基準である．

$$S_m = \frac{\text{プロセッサ1台の拡張メイクスパン}}{\text{プロセッサ } P\text{台の拡張メイクスパン}}$$

$$S_e = \frac{\text{プロセッサ1台の実行時間}}{\text{プロセッサ } P\text{台の実行時間}}$$

$S_m(S_e)$ 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．

バルク同期スケジュールの通信層に対する拡張メイクスパンは，各全対全通信に参加するプロセッサ数 P に応じて算出する．以下，簡単のために，通信層の開始時刻を0とする．あるプロセッサに着目すると，通信に参加する残りのプロセッサへ送信する場合のオーバヘッドは，$o_s \times (P-1)$ になる．つまり，このプロセッサから最後に送信された通信が受信可能になる時刻は，$(P-1) \times o_s + L + o_r$ である．一方，すべてのプロセッサの対に対して通信が発生すると考えると，各プロセッサは通信層で少なくとも $P-1$ 回の送受信処理を行う．つまり，アイドルスロットが0で，オーバヘッドの処理しか行わない場合でも，$(P-1) \times (o_s + o_r)$ の時間がかかる．以上より，通信層の拡張メイクスパンを，$max((P-1) \times o_s + L + o_r, (P-1) \times (o_s + o_r))$ の式で算出する．

9.1.1 実行条件

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 450 MHz，メモリ512 MB）16台を，Myrinet[15]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．生成した並列プログラムでは，1対1通信は`MPI_Isend()`，`MPI_Irecv()`，`MPI_Wait()`を用いて実行する．また，全対全通信は`MPI_Alltoallv()`を用いて実行する．使用したMPIライブラリの実装はMPICH[14]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生成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た．本実験に用いた，SPPCモデルのパラメータを付録A.1に示す．

9.2 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

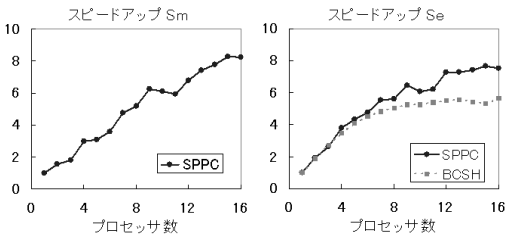


図14 スピードアップ S_m，S_e（ガウスジョルダン法）
Fig. 14 Speedup ratios of Gauss-Jordan method.

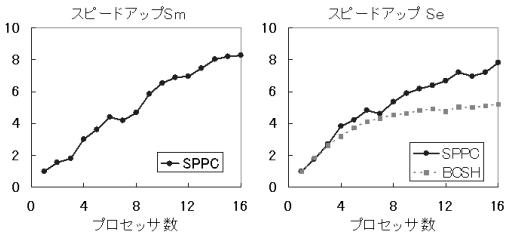


図15 スピードアップ S_m，S_e（LU分解）
Fig. 15 Speedup ratios of LU decomposition.

S_e の測定結果を**図14**右図，**図15**右図に示す．プロセッサ数の増加にともないSPPCの S_e が上昇し，BCSHを上回る結果を得られた．図14，図15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，S_m を基準とした改善は有効であるといえる．

以下，SPPCの性能がBCSHを上回る要因について分析する．

9.2.1 同期コストの改善

SPPCの性能が上回る要因として，SPPCは非同期式の1対1プロセッサ間通信（以下，1対1通信），BCSHはバルク同期式の全対全通信を行うことが考えられる．バルク同期スケジュールに対応する並列プログラムの全対全通信を1対1通信に変更した場合のガウスジョルダン法の S_e を**図16**に示す．

プロセッサ数の増加にともない，1対1通信に変更した並列プログラムの性能が向上している．全対全通信では通信時に他のプロセッサと同期するため，実際には少し前の時刻で送信可能になっている通信が待たされたり，あるいは，他のプロセッサの受信が終わるまで，次の計算層の実行ができなかったりすることがある．これらの性能悪化につながる部分（**同期コスト**）を，1対1通信を用いて改善できることが分かる．プロセッサ数が増加するにつれて，同期コストが発生するプロセッサ数も増加し，1対1通信による改善効果が大きくなる．

9.2.2 プロセッサ稼働率

SPPCとBCSHのプロセッサ稼働率の差にも要因

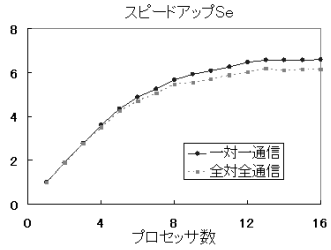


図 16 1 対 1 通信によるバルク同期スケジュールの改善
Fig. 16 Improvement S_e of BCSH using point-to-point communications.

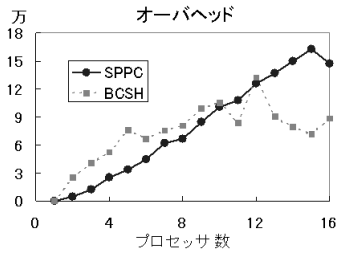


図 18 オーバヘッド（ガウスジョルダン法）
Fig. 18 Amount of software overhead (Gauss-Jordan method).

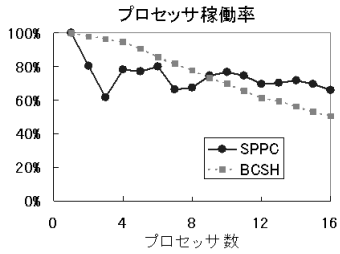


図 17 プロセッサ稼働率（LU 分解）
Fig. 17 Processor usage (LU decomposition).

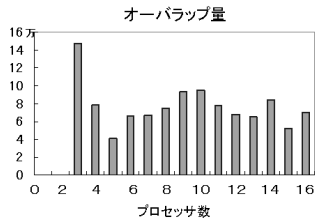


図 19 オーバラップ量（ガウスジョルダン法）
Fig. 19 Amount of overlap (Gauss-Jordan method).

があると考える．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，並列スケジュール上の全プロセッサのアイドルスロットの総和を I として，**プロセッサ稼働率**を次式で定義する．

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

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 17** に LU 分解におけるプロセッサ稼働率を示す．

バルク同期スケジュールでは通信層の実行にかかる時間は，プロセッサ数に比例する．通信層の実行時間の増加はメイクスパンの増加を招くので，BCSH ではプロセッサ数節約[10]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．このプロセッサ数節約の結果，BCSH はプロセッサ数が大きい場合に，必ずしもすべ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 17 より，プロセッサ数が大きい場合に，SPPC はプロセッサを有効に利用できていることが分かる．また，プロセッサ数が小さい場合では，SPPC のプロセッサ稼働率が BCSH を下回る．図 15 を見ると，プロセッサ数が小さい場合にも，SPPC のスピードアップ S_e は BCSH の S_e と同等以上である．しかし，プロセッサ数が大きい場合に比べて差は小さい．これは，プロセッサ稼働率が低いためであると考える．

9.2.3 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 18** に示す．プロセッサ数の増加にともない，SPPC スケジュールのオーバヘッドが BCSH を上回る．オーバヘッド量が BCSH に比べて多いにもかかわらず，図 14 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．

9.2.4 オーバラップ量

受信待ち時間の低減に対する評価値として，オーバラップ量を用いる．**オーバラップ量**は，1 つのレイヤに対する処理の中で，送信から次の送信までに実行するタスクの重みの総和で算出する．オーバラップ量が大きいほど，送信可能になった通信から順に早い時刻に送信できていると考える．

SPPC スケジュールのオーバラップの総量を**図 19** に示す．オーバラップ量の必ずしもすべてが受信待ち時間の低減に効果があるとはいえない．しかし，BCSH のオーバラップ量がつねに 0 であることを考慮すると，オーバラップの発生が，SPPC の拡張メイクスパンと実行時間が BCSH に比べて良好な結果を示す一因であると考えられる．

10. 関連研究

通信遅延を考慮した既存のタスクスケジューリングアルゴリズムの中で，メイクスパンの小さい並列スケジュールを生成するアルゴリズムとして，DSH（Duplication Scheduling Heuristic）[13]，BTDH（Bottom-Up Top-Down Duplication Heuristic）[3]，LWB（Lower Bound）[4]，LTCD（Linear Clustering with Task Duplication）[19]，Papadimitriou らのアルゴリズム[18]，CPFD（Critical-Path First Duplication）[1]，ECPFD（Economical CPFD）[1]，Palis らのアルゴリズム[17]，Darbha らのアルゴリズム[7]などがよく知られている．これらのアルゴリズムは，古典的モデルに基づくアルゴリズムであり，通信の一括化を考慮した並列スケジュールを生成しない．また，これらのアルゴリズムのうち，利用可能プロセッサ数をパラメータで指定することができるのは，DSH と ECPFD のみである．それ以外のアルゴリズムでは，必要に応じてプロセッサを何台でも利用できると仮定している．

通信の一括化の利用を目指したアルゴリズムとして，我々の知る限りでは，BCSH と本論文で提案する SPPC がある．既存のアルゴリズムに対し，通信の一括化の効果を考慮したアルゴリズム SPPC の有用性を示すために，SPPC，BCSH，DSH，ECPFD との比較実験を行った．既存のアルゴリズムの中から DSH と ECPFD を比較対象として選んだ理由は，これらのアルゴリズムは利用可能プロセッサ数を指定することができ，同数のプロセッサを利用する場合で比較できると考えるからである．

ガウスジョルダン法（要素数 64）に対する，SPPC，BCSH，DSH，ECPFD の各アルゴリズムのスピードアップ S_e の結果を**図 20** に示す．DSH，ECPFD は並列スケジュール生成にかかる計算量が大きいアルゴリズムである．タスクグラフのタスクの集合を V，有向辺の集合を E，プロセッサ数を P とすると，DSH の計算量は $O(|V|^4)$，ECPFD は $O(P|V||E|)$ である．ガウスジョルダン法（要素数 64）を入力に選ぶ理由は，DSH および ECPFD が現実的な時間で並列スケジュールを得られる範囲で，かつ，大規模であるタスクグラフであるからである．ガウスジョルダン法（要素数 64）のタスクグラフは，9 章に述べた要素数 128 の場合と同様に，処理を分割して 1 代入文を 1 タスクとした．タスク数は約 14 万である．

図 20 にガウスジョルダン法（要素数 64）に対する，各アルゴリズムの S_e の測定結果を示す．結果より，

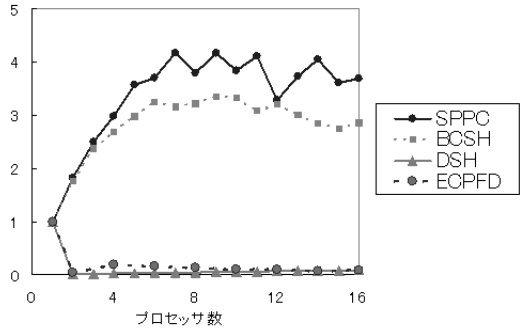


図 20 S_e の比較（ガウスジョルダン法：要素数 64）
Fig. 20 Results of speedup ratio S_e of SPPC, BCSH, DSH and ECPFD (Gauss-Jordan method : 64 elements).

通信の一括化を利用するアルゴリズムである SPPC，BCSH が，通信の一括化を利用しないアルゴリズムである DSH，ECPFD より，実行時間の短い並列プログラムを生成できることが分かる．特に，DSH，ECPFD の S_e は 1 を下回る．つまり，DSH，ECPFD の生成する並列プログラムの実行時間は，逐次プログラムの実行時間よりも長い．一方，SPPC，BCSH の S_e は 1 を上回ることより，並列処理による高速化を実現する並列プログラムを生成しているといえる．

11. 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 モデルでは通信の一括化を適用した 1 対 1 通信を行うことができるため，既存の全対全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 モデル上で動作するアルゴリズム 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分かった．

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) Ahmad, I. and Kwok, Y.: On Exploiting Task Duplication in Parallel Program Scheduling, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.9, pp.872–892 (1998).
2) 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 (1994).
3) Chung, Y.C. and Ranka, S.: Application and Performance Analysis of a Compile Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors, *Proc. Supercomputing '92*, pp.512–521 (1992).
4) Colin, J.Y. and Chritienne, P.: C.P.M. Scheduling with Small Communication Delays and Task Duplication, *Operations Research*, Vol.39, No.4, pp.680–684 (1991).
5) 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).
6) 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 (1996).
7) Darbha, S. and Agrawal, D.P.: Optimal Scheduling Algorithm for Distributed Memory Machines, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.1, pp.87–95 (1998).
8) Dongarra, J.J. and Dunigan, T.H.: Message-passing Performance of Various Computers, Report ORNL/TM-13006, Oak Ridge National Laboratory, Oak Ridge, TN, February (1996).
9) El-Rewini, H., Lewis, T.G. and Ali, H.H.: *Task Scheduling in Parallel Distributed Systems*, pp.56–105, PTR Prentice Hall (1994).
10) 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).
11) Graham, R.L.: Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, Vol.17, pp.416–429 (1969).
12) Kasahara Lab., Waseda Univ: *Standard Task Graph Set.* http://www.kasahara.elec.waseda.ac.jp/schedule/index.html
13) Kruatrachue, B.: Static task scheduling and packing in parallel processing systems, Ph.D. diss., Department of Electrical and Computer Engineering, Oregon State University, Corvallis (1987).
14) MPICH: A Portable MPI Implementation. http://www.mcs.anl.gov/mpi/mpich/
15) Myrinet: Myricom. http://www.myri.com/
16) Pacheco, P.S.: *Parallel Programming with MPI*, Morgan Kaufmann Publishers, Inc. (1997).
17) Palis, M.A., Liou, J. and Wei, D.S.L.: Task Clustering and Scheduling for Distributed Memory Parallel Algorithms, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.1, pp.46–55 (1996).
18) Papadimitriou, C.H. and Yannakakis, M.: Towards An Architecture-Independent Analysis of Parallel Algorithms, *SIAM J. Comput.*, Vol.19, No.2, pp.322–328 (1990).
19) Shirazi, B., Chen, H. and Marquis, J.: Comparative study of task duplication static scheduling versus clustering and non-clustering techniques, *Concurrency: Practice and Experience*, Vol.7, No.5, pp.371–390 (1995).
20) Valiant, L.G.: A bridging model for parallel computation, *Comm. ACM*, Vol.33, No.8 (1990).
21) 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).

付　　録

A.1 評価実験に用いた通信遅延のパラメータ

ガウスジョルダン法（要素数 128），LU 分解（要素数 128）に対する評価実験における SPPC モデルの通信遅延に対するパラメータ を**表 1** に示す.

表 1 評価実験に用いたパラメータ o_s, o_r, L
Table 1 Parameters (o_s, o_r, L) of SPPC model for experiments.

プロセッサ数 P	o_s	o_r	L
2	132	42	362
3	128	41	358
4	124	40	354
5	120	39	350
6	116	38	346
7	112	37	342
8	108	36	338
9	104	35	334
10	100	34	330
11	96	33	326
12	92	32	322
13	88	31	318
14	84	30	314
15	80	29	310
16	76	28	306

表 2 評価実験における SPPC のスケジューリング時間 [分]
Table 2 Scheduling time of SPPC [min.].

プロセッサ数	ガウスジョルダン法	LU 分解
2	696	462
3	254	194
4	275	242
5	288	227
6	290	199
7	279	231
8	281	224
9	311	208
10	293	251
11	273	214
12	285	261
13	340	195
14	239	254
15	352	270
16	320	238

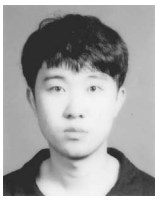
A.2 評価実験における SPPC のスケジューリング時間

ガウスジョルダン法（要素数 128），LU 分解（要素数 128）に対する評価実験における SPPC のスケジューリング時間☆を**表 2** に示す．

(平成 14 年 2 月 4 日受付)
(平成 14 年 3 月 27 日再受付)
(平成 14 年 4 月 25 日採録)

山本 裕己

平成 12 年大阪大学基礎工学部情報科学科卒業．平成 14 年同大学大学院基礎工学研究科博士前期課程修了．タスクスケジューリング手法を用いた並列プログラムの生成に興味を持つ．



譚 林

平成 10 年大連理工大学計算機系卒業．同年，株式会社東大アルパインソフトウエア入社．平成 13 年大阪大学大学院基礎工学研究科博士前期課程入学．現在，同大学院基礎工学研究科博士前期課程在学中．並列プログラム生成系に興味を持つ．

藤本 典幸（正会員）

平成 4 年大阪大学基礎工学部情報工学科卒業．平成 6 年同大学大学院基礎工学研究科博士前期課程修了．平成 9 年同大学院基礎工学研究科博士後期課程単位取得退学．平成 9 年同大学院基礎工学研究科助手．平成 14 年より大阪大学大学院情報科学研究科助教授．工学博士．並列アルゴリズム，並列言語の処理系・開発環境などに興味を持つ．

萩原 兼一（正会員）

昭和 49 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学大学院基礎工学研究科博士課程修了．工学博士．同大学基礎工学部助手，講師，助教授を経て，平成 5 年奈良先端科学技術大学院大学教授．平成 6 年より大阪大学教授．平成 4 年〜5 年文部省在外研究員（米国メリーランド大）．現在，並列処理の基礎および応用に興味を持っている．

☆ ガウスジョルダン法のスケジューリングに用いた計算機の構成は（Athlon 1.1 GHz，メモリ：1 GB），LU 分解のスケジューリングに用いた計算機の構成は（Pentium II 450 MHz，メモリ：2 GB）である．