細粒度並列アーキテクチャ用SISAL コンパイラにおける並列粒度調整方式

高畠志泰[†] 大澤範高[†] 弓場敏嗣[†]
佐藤三久[††] 山口喜教[†††]

並列プログラムの効率的な実行においては，並列計算機を構成する各要素プロセッサ上で並列処理される計算負荷（処理粒度と呼ぶ）の実行時間と，異なる要素プロセッサ上の処理粒度間での情報授受に要する通信時間の適正なバランスを考える必要がある．本論文では，与えられたプログラムをコンパイラで並列化する際に，処理粒度を対象とする並列計算機に適合させることによって，同プログラムの実行時間を最小にする並列粒度調整方式を提案する．粒度調整を行う際に，並列計算機上での並列プログラムの実行時間を評価するモデルとしてLogPモデルを用いる．提案する粒度調整方式を関数型言語SISALのコンパイラに組み込む．コンパイラの実装は，電子技術総合研究所のデータ駆動型並列計算機EM-Xを対象として行う．

Tuning Parallel Granularity in the SISAL Compiler for Fine-grain Parallel Architectures

MOTOYASU TAKABATAKE,[†] NORITAKA OSAWA,[†] TOSHITSUGU YUBA,[†]
MITSUHISA SATO[††] and YOSHINORI YAMAGUCHI[†††]

Optimum granularity of parallel computation is determined by a balance of processing time and communication time among parallelized computation. Parallel granularity must be tuned to an architecture of the target machine. In this paper, a granularity tuning method is proposed, which uses the LogP model for evaluating parallel execution time of a given program on a parallel computer. The method is partly implemented in a SISAL compiler of the EM-X parallel computer developed at the Electrotechnical Laboratory. Some preliminary measurements are shown, and the effectiveness of the granularity tuning method is discussed.

1. はじめに

分散記憶型並列計算機の並列化コンパイラにおいて，プログラムの並列化方法やデータの分散方法，通信時間の隠蔽などにより，実行速度の向上が行われている．また，要素プロセッサ（PE）の台数やPE間の通信を意識しないプログラミングを可能とする研究が進められている[6]．しかしながら，分散記憶型並列計算機においては，使用するPE数を増やし，各PE上での計算処理の時間を短くしても，プログラム全体の実行時間が必ずしも短くなるとは限らない．一般には，使用するPE数を増やし過ぎるとPE間の通信回数が増加し，実行時における通信時間が長くなるため，全体の実行時間はかえって長くなる．よって，対象とする並列計算機の通信時間を考慮に入れて計算負荷を調整することが重要である．

本論文では，与えられた並列プログラムにおいて，静的に解析可能な並列性と通信時間を考慮した並列粒度調整方式を提案する．提案する方式により，プログラム全体の実行時間を短くする最適なPE数と，そのときの処理粒度を求める．このとき，対象並列計算機上で並列プログラムを実行したときの所要実行時間を与える評価モデルとしてLogPモデル[2]を用いる．

コンパイラの設計と実装にあたり，コンパイラの処理する言語は，コロラド大学とローレンスリバモア研究所で開発された関数型言語SISAL（Streams and Iteration in a Single Assignment Language）[4]を用いる．対象とする並列計算機は，電子技術総合研究所

† 電気通信大学大学院 情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 新情報処理開発機構
Real World Computing Partnership
††† 電子技術総合研究所
Electrotechnical Laboratory

で開発された EM-X[3]を使用する．EM-X は，細粒度の並列処理が可能な高並列計算機である．

本論文で扱う並列処理粒度とは，1 つの PE において，他の PE と通信を行わずに連続に処理される計算の実行時間を指し，粒度調整とは，その実行時間を調整することとする．粒度調整で扱う粒度の単位は，通信処理を計算機のクロックサイクル数，計算処理をアセンブリ言語の命令数で表す．粒度調整を行う部分は，1）並列ループと，2）関数呼び出しである．

2. 関数型言語 SISAL と細粒度並列計算機

SISAL[4]は，単一代入規則を持ち，汎用性のある並列処理記述に適した関数型言語である．繰返し構造の実現方法にループアンフォールディングという機構を用い，ストリームという構造体も扱える．この言語の持つ単一代入性，関数の局所的性質により，各 PE における処理粒度を自由に調整可能となる．SISAL で記述されたプログラムは，既存の SISAL コンパイラ[1]を用いて，中間言語である MIDC（Machine Independent Dataflow Code）[5]表現のプログラムへ変換できる．MIDC プログラムは，複数の命令が 1 つのノードで与えられるマクロデータフロー図として表すことができる．

細粒度並列計算機には，EM-X[3]を用いる．EM-X は，80 台の PE が高速な相互接続網で接続された分散記憶型並列計算機であり，データ駆動と逐次実行を融合させた実行モデルに基づいている．マルチスレッド機構を持ち，命令水準の細粒度並列処理が可能である．

EM-X のデータ駆動と逐次実行を融合させた実行モデルにより，MIDC のマクロデータフロー図で表現された MIDC プログラムを EM-X オブジェクトコードへ変換することが容易となる．

3. LogP モデルと EM-X パラメータ

並列計算機上での並列プログラムの実行時間を評価するために，PE 間の通信時間を考慮した LogP モデル[2]を使用する．同モデルにおいては，並列計算機の PE 数と PE 間の通信時間を以下の 4 つのパラメータを使って表す．これらのパラメータは，対象とする並列計算機の性能仕様によって固定的に定まる．

- L：1 語のメッセージを送るのに要する時間の遅れの上限（通信遅延）．
- o：PE においてメッセージの送受信に要する時間（送受信オーバヘッド）．
- g：メッセージを連続して送受信するときの最小時間間隔（バンド幅の逆数）．

表 1 EM-X における LogP モデルのパラメータ
Table 1 LogP model parameter of EM-X.

LogP モデルのパラメータ	値
通信遅延の上限（L）	8 サイクル
メッセージの送信時間（o_s）	1 サイクル
メッセージの受信時間（o_r）	1 サイクル
バンド幅の逆数（g）	2 サイクル
PE 数（P）	80 台

- P：PE 数．

EM-X における LogP モデルパラメータは，クロックサイクルを単位として，**表 1** で与えられる．この値は，計算機の性能から得られる理論値である．送受信オーバヘッド（o）は，送信（o_s）と受信（o_r）とに分けて考慮する．なお，EM-X では，通常の命令は 1 クロックサイクルで実行される．ここで，EM-X では，スレッドを用いて計算処理を行うので，LogP モデルを拡張する．スレッドを実行中の PE とは異なる PE に生成する場合，そのスレッドを割り当てる PE を決める時間と生成時間を考慮する．スレッドを割り当てる PE を決める時間は 29 サイクル，スレッドの生成時間は 21 サイクルとすると，スレッド生成に関するオーバヘッドは 50 サイクルとなる．この値は，メッセージの送受信時間や通信遅延とは別の通信オーバヘッドとして考慮する．

- F：スレッド生成に関するオーバヘッド．

4. 並列ループの粒度調整方式

4.1 最適粒度の求め方

MIDC におけるループ構造の種類は，各イタレーションが独立である並列ループとイタレーション間に依存関係のある逐次ループの 2 種類である．ここでは，並列ループを対象として粒度調整を行う．並列ループを分割統治法で分割することで，イタレーションを並列に実行することができる．実際には，ループの中のいくつかのイタレーションをまとめて 1 つのスレッドとして生成し，1 つの PE に割り当てる．粒度調整は，そのスレッドの中に含まれるイタレーションの数を変えることによって行う．

並列ループの粒度調整における最適粒度を以下の手順で求める．

(1) 並列計算機上の実行過程を表す基本形
与えられたプログラムの並列ループに対して，分割統治法により 2 分木構造をつくる．並列ループの実行過程は，**図 1** (a) のように 2 分木を折り返した構造で表現される．

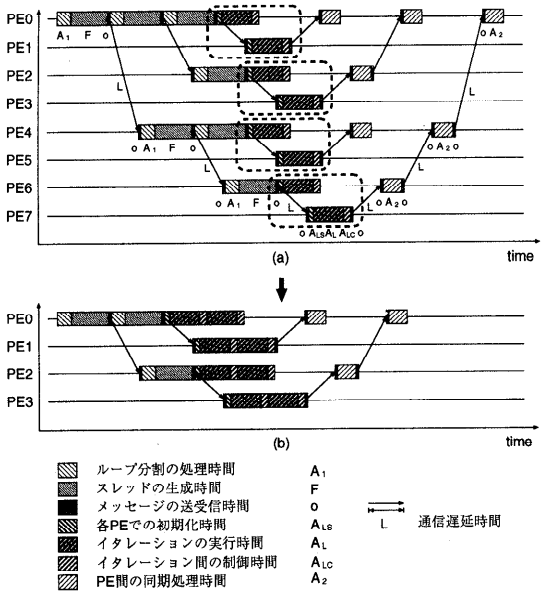


図 1 並列ループの実行過程と粒度調整結果
Fig. 1 Execution processes of parallel loop and a result of granularity tuning.

(2) 実行時間を求める式の作成

コンパイル時に，以下のパラメータの命令数を調べる．これらのパラメータはプログラムに依存する．これら値をコンパイル時に静的に求められることが，粒度調整を行うための必要条件となる．

- 各 PE での初期化時間 A_{LS}
- イタレーションの実行時間 A_L
- イタレーション間の制御時間 A_{LC}
- ループ分割の処理時間 A_1
- PE 間の同期処理時間 A_2

N をループ回数（イタレーション数）とする．図 1 (a) において，PE0 → PE4 → PE6 → PE7 → PE6 → PE4 → PE0 をたどる実行過程を考える．ループの実行時間 T を求める近似式は，上記のパラメータと LogP モデルを用いることで，次式で与えられる．

$$\begin{aligned} T &= (F + o_s + L + o_r + A_1)\log_2(P) \\ &\quad + A_{LS} + (A_L + A_{LC})\frac{N}{P} \\ &\quad + (o_s + L + o_r + A_2)\log_2(P) \\ &= (F + 2L + 2o_s + 2o_r + A_1 + A_2) \\ &\quad \times \log_2(P) \\ &\quad + (A_L + A_{LC})\frac{N}{P} + A_{LS} \end{aligned} \tag{1}$$

式 (1) の第 1 項はスレッドの生成，ループ処理に必要なデータの転送，および PE 間の同期に必要な時間，第 2 項は各 PE 上でのイタレーションの実行時間である．EM-X では，機械語レベルの send 命令で 1 ワードのデータの転送ができる．そのため，通信データ量は，EM-X の通信命令数を数えることで調べることができる．そして，バンド幅の逆数 (g) は，コンパイル時に求まる各パラメータ A_* に含まれる．

(3) 最適粒度と PE 数を与える式

使用する PE 数で 1PE あたりの実行イタレーション数が決まるので，最適粒度は PE 数に依存する．式 (1) を PE 数 P で微分し，最小の実行時間を与える PE 数，すなわち，最適な PE 数 P_{opt} を求めると，以下の式を得る．

$$P_{opt} = \frac{(A_L + A_{LC})N\log_e 2}{F + 2L + 2o_s + 2o_r + A_1 + A_2} \tag{2}$$

したがって，各 PE における実行イタレーション数（最適粒度）は，以下の式になる．

$$\frac{N}{P_{opt}} = \frac{F + 2L + 2o_s + 2o_r + A_1 + A_2}{(A_L + A_{LC})\log_e 2} \tag{3}$$

4.2 コンパイル時の処理

式 (3) は，LogP モデルのパラメータとコンパイル時に決定する実行時間のみの式になり，その値は定数になる．ループ回数 N には関係しないので，コンパイラに式 (3) を組み込むことにより，コンパイル時に最適粒度を求めることができる．

各 PE における最適な実行イタレーション数（最適粒度）は，式 (3) よりループ回数 N に依存しない．しかし，最適な PE 数は，式 (2) によりループ回数 N に依存する．上で述べた最適粒度を求める方法で得た最適粒度は，PE 数が無限の場合では，どのような大きさの並列ループにでも対応できる．しかし，現実の並列計算機において実際に使える PE 数は有限である．

式 (2) から得られる最適な PE 数を最適 PE 数，実機の PE 数を実 PE 数とする．最適 PE 数が実 PE 数を超える場合，すべての PE へイタレーションの数が均等に割り付けられるようにし，式 (3) から得られた最適粒度より実際の粒度を大きくする．つまり，すべての PE の負荷を均等にする．

最適 PE 数が実 PE 数を超える場合を，コンパイル時に並列ループのループ回数 N が求まる場合と求まらない場合にわけ，それぞれ異なるオブジェクトコードを生成することにする．それぞれのオブジェクトコードの説明は，4.3 節の中で行う．

4.3 実行時の振舞い

実行は，分割統治法を用いてループを2つに分割することを繰り返し，分割が終わったところでループの並列実行を行う．

ループ回数がコンパイル時に求まっている場合，1つのPE上の処理がコンパイル時に求まった最適粒度のイタレーション数に近くなるまでループの2分割を繰り返す．2分割が終わったところで部分ループを実行する．各PE上で実行されたイタレーション数が粒度調整を行った結果の粒度である．

一般には，コンパイル時にはループ回数が分からない．実行時にループの入口でループ回数が分かる場合，以下の処理を行い，実行時に粒度調整を行う．それ以外の場合では，本粒度調整方式の対象外とする．

コンパイル時に求まっている最適粒度のイタレーション数を N_{opt}，実PE数を P_{max} とする．コンパイル時に，高さ $\lceil \log_2(P_{max}) \rceil$ の2分木を生成しておく．この2分木は，ループを分割統治法を繰返し用いて分割する．ここでの高さは葉の方から数えるものとする．

(1) 実行時にループの入口でループ回数 N を求める．$\lceil \log_2(N/N_{opt}) \rceil$ の値の高さの部分木から実行を開始する．$\lceil \log_2(N/N_{opt}) \rceil$ の値が木の最大の高さ $\lceil \log_2(P_{max}) \rceil$ の値を超える場合は，高さ $\lceil \log_2(P_{max}) \rceil$ から実行を開始する．

(2) 2分木のノード上では，部分ループの始まりの値と終わりの値からループを2等分に分割し，各ループを子ノードへ渡す．

(3) 葉のノード上では，部分ループの始まりの値から終わりの値までの部分ループの実行を行う．

以上のように，実行時に木の高さを計算で求め，その高さから実行できるようなコードを生成する．実際には，各高さにおいて1つのノードにエントリポイントをおく．$\lceil (N/N_{opt}) \rceil$ の値で条件分岐を行い，対応した高さのノードのエントリポイントへ実行が移る．

このときの各PE上で実行される部分ループに含まれるイタレーション数が粒度調整を行った結果の粒度となる．

ループ回数がコンパイル時に求まる場合と比べると実行時には，以下のようなオーバヘッドが生じる．

(1) 実行時する木の高さを決める時間．

(2) 2分木上での部分ループの始まりの値と終わりの値の計算時間．

(3) 部分ループの始まりの値と終わりの値の送受信時間．

(1)は，$\lceil \log_2(N/N_{opt}) \rceil$ の計算を行う部分であるが，実行コードでは $\lceil (N/N_{opt}) \rceil$ の値で条件分岐を行い，logの計算は行わない．この実行時間は一定になるので，式(1)にこの値を加えても粒度調整に影響を与えない．(2)と(3)は2分木上での処理になるので，式(1)では A_1 の中に含まれる．(2)の計算時間は，定まった計算なので定数になる．(3)の送受信時間はLogPモデルにより定数である．よって，A_1 は定数になるので，粒度調整方式には問題は起きない．しかし，A_1 の値は，ループ回数が静的に求まるときよりも大きくなるので，求まる最適粒度は，ループ回数が静的に求まる場合の値と同一ではない．

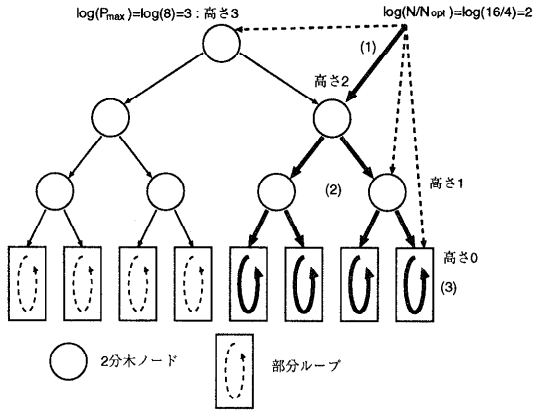


図2 コンパイル時にループ回数が求まらないときの実行モデル
Fig. 2 An execution model for inability to decide loop count in compiling.

4.4 例

具体的に1から N までの和を求めるループの例をあげて説明する．実PE数を8台，ループ回数を16と仮定する．

コンパイル時の処理で，まず，式(3)より最適粒度 $N_{opt} = 4$ を求める．そして，高さ $\lceil \log_2(P_{max}) \rceil = \lceil \log_2(8) \rceil = 3$ の2分木を静的に作る．

実行時の振舞いは，以下のようになる．

(1) 実行時に木の高さ $\lceil \log_2(N/N_{opt}) \rceil = \lceil \log_2(16/4) \rceil = 2$ を求め，木の高さ2の所からループの2分割を進める（**図2**(1)）．

(2) ループ1~16が，高さ2で，1~8と9~16へ分割され，さらに高さ1で1~4，5~8，9~12，13~16へ分割される（同図(2)）．

(3) 各々の分割された部分ループを実行する（同図(3)）．

5. 関数呼び出しの粒度調整方式

5.1 最適粒度の求め方

関数では，その呼び出しと復帰の実行過程を表現す

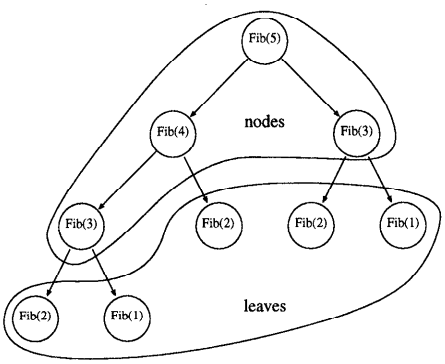


図3 フィボナッチ関数による関数呼び出しの関係
Fig. 3 Relations of Fibonacci function.

るデータフロー図を作ることができる．関数呼び出しは葉になる関数と節になる関数の2つの種類に分けられる（**図3**）．節になる関数は，その関数内で関数呼び出しを行う．葉になる関数は，その関数内で関数呼び出しを行わない．データフロー図において，葉の関数と葉を呼び出す節の関数をまとめることにより，粒度を大きく調節することができる．実際には，1つのスレッドに1つの関数を対応させる．関数を呼び出すときに，呼び出された関数のスレッドを呼び出した関数のスレッドと同じPE上で実行することで，PE間の通信がなくなり粒度調整が可能になる．つまり，粒度調整は，同じPE上で実行されるスレッドの数を変えることによって行う．

関数呼び出しの粒度調整が行える条件は，コンパイル時に関数呼び出し関係が静的に求まる場合であり，かつ，関数の実行時間が一定である必要がある．また，同時に呼び出すことのできる関数の数は2つまでとする．関数が再帰呼び出しをする場合では，再帰が終わる条件式をコンパイラで判断できなければならない．

関数呼び出しの粒度調整における最適粒度を以下の手順で求める．

(1) 並列計算機上の実行過程を表す基本形
図4(a)のように木構造を折り返した形をデータフロー図の基本として考える．同図(b)で示すように，葉のノードとその親のノードを同じPEへ割り当て，より大きな1つの葉のノードにすることにより粒度を調整する．

(2) 実行時間を求める式
N を葉の関数の数，A を関数の命令数とする．図4(a)において，PE0→PE4→PE6→PE7→PE6→PE4→PE0をたどる実行過程を考える．全体の実行時間 T を求める近似式は，上記のパラメータとLogPモデルを用いることで，

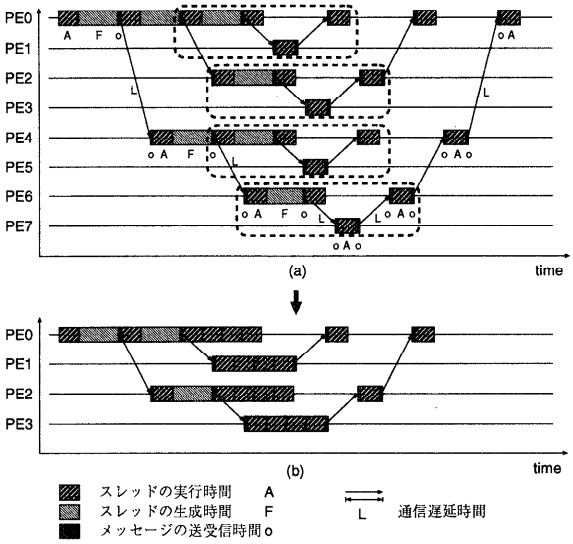


図4 関数呼び出しの実行過程と粒度調整結果
Fig. 4 Execution processes of function call and a result of granularity tuning.

次式で与えられる．

$$\begin{aligned} T &= (o_s + L + o_r + F + A)\log_2(P) \\ &\quad + A\left(\frac{3N}{P} - 2\right) \\ &\quad + (o_s + L + o_r + A)\log_2(P) \\ &= (F + 2L + 2o_s + 2o_r + 2A)\log_2(P) \\ &\quad + A\left(\frac{3N}{P} - 2\right) \end{aligned} \tag{4}$$

式(4)の第1項は節に置かれた関数の実行時間と通信時間である．第2項は粒度調整の対象となる部分の関数の実行時間である．

(3) 最適粒度とPE数を与える式
式(4)から最適なPE数 P_{opt} を計算すると，以下の式になる．

$$P_{opt} = \frac{3AN\log_e 2}{F + 2L + 2o_s + 2o_r + 2A} \tag{5}$$

最適粒度（命令数）を与える式としては，式(4)の第2項より，次式を得る．

$$A\left(\frac{3N}{P_{opt}} - 2\right) = \frac{1}{\log_e 2}(F + 2L + 2o_s + 2o_r + 2A) - 2A \tag{6}$$

5.2 コンパイル時の処理

式(6)をコンパイラへ組み込むことによりコンパイル時に最適粒度を求めることができる．式(6)は，LogPモデルから決まるパラメータとコンパイル時に求まるパラメータのみである．そのため，この式の値は定数になり，葉の関数の数（N）に依存しない．この粒度調整方式が適用できる条件としては，関数の命

令数がコンパイル時に求まることである．

関数呼び出しの粒度調整には，関数が葉になるか節になるかの判定が必要になる．また，葉と節の関数のスレッドの割当て場所を決める方法も必要になる．関数呼び出しにおける粒度調整のコンパイル手順を以下に示す．

(1) スレッドの割当て場所を決める部分以外の部分について，MIDCプログラムをEM-Xの実行コードに変換する．

(2) 関数の実行命令数を求める．関数が再帰呼び出しの場合には，葉になるか節になるかの条件式と，葉になる場合と節になる場合の両方の命令数を求める．

(3) 葉になる関数を調べる．その葉を呼ぶ節の関数が1つの関数呼び出ししかしない場合は，葉と節の関数は，同じPE上で実行するようにする．その葉の関数の命令数と節の関数の命令数を加える．また，その節の関数を呼ぶ関数を同じように調べること繰り返す．

(4) 葉の関数を呼ぶ節の関数が，同時に2つの関数呼び出しができる場合，葉の関数の命令数を式(6)で得られた最適粒度と比較する．小さければそれを呼ぶ節の関数の命令数と，同時に呼ぶ関数の命令数を加える（図4(a)の点線で囲まれた部分）．そしてまた，最適粒度と比較する．この手順を最適粒度に近い値になるまで繰り返す．

(5) 関数が再帰呼び出しの場合，再帰呼び出しが終わる条件分岐の式に対して，上で繰り返した回数を考慮したものをスレッドの割当て場所に利用する．

(6) スレッドの割当て場所を決める部分を含んだ関数全体のコンパイルをする．

スレッドの割当て場所を決める部分には，呼び出した関数と同じPE上にスレッドを生成するか，それとも異なるPE上に生成するかの情報のみである．呼び出された関数を生成する先のPEをコンパイル時には決めない．生成先のPEは，実行時のランタイムルーチンが決める．

5.3 実行時の振舞い

実行時に関数呼び出しが起きるとき，呼び出し側の関数と同じPE上に呼び出された関数のスレッドを生成するかどうかを調べる．同じPE上に生成する場合には，呼び出される関数のスレッドを呼び出した関数のスレッドと同じPE上に生成する．異なるPE上に生成する場合には，ランタイムルーチンを使い，呼び出された関数のスレッドを生成するPEを決める．ランタイムルーチンには，あらかじめPEごとにスレッドを生成する先のPEを決めた情報を持たせておく．この情報は，関数が同時に2つ呼ばれる場合を想定して実行時に2分木を構成するようにしておく．2分木が深くなり，実PE数を超えるような関数呼び出しが起きた場合には，呼び出した関数と同じPE上で呼び出された関数が実行するようにランタイムルーチンの情報に書いておく．

実際に各PE上で実行された複数の関数の合計命令数が粒度調整を行った結果の粒度となる．

5.4 例

フィボナッチ数列を求める関数を例に示す．この関数は，直接再帰関数である．引数を n，再帰呼び出しをしないときの命令数を A_1，再帰呼び出しをするときの命令数を A_2 とする．

コンパイル時の処理は以下のようになる．

(1) フィボナッチ関数のスレッド生成の割当て場所を決める部分以外をコンパイルする．

(2) コンパイル時に，葉になるか節になるかの条件（$n < 2$）を調べ，葉になる場合の命令数 A_1 と，節になる場合の命令数 A_2 を求める．

(3) フィボナッチ関数が関数を呼び出すときには，つねに2つの関数を呼び出すため，葉の関数を調べる必要はない．

(4) 葉の部分の関数の命令数 A_1 を2つと節の部分の関数の命令数 A_2 を加算する．さらに，その加算した値と，その節の親の関数の命令数を加算する．それを最適粒度に近くなるまで繰り返す．

$((2A_1 + A_2) + (2A_1 + A_2) + A_2) =$ 式(6)

(5) 上で3回繰り返したとすると，条件分岐で $n < 2 + 3 = 5$ のときにスレッドの割当て場所を変える．n の値が5より小さいときは，関数呼び出しをするPEと同じPEにスレッドの割当てを行う．5以上のときは，他のPEにスレッドの割当てを行う．

(6) スレッドの割当て場所を決める部分を含め，フィボナッチ関数全体をコンパイルする．

このようにして，関数のスレッドの割当て場所を変えることにより，**図5**(a)から(b)への粒度調整が可能になる．

6. 予備的評価

6.1 並列ループの粒度調整

1から N までの和を求めるプログラムにおいて，並

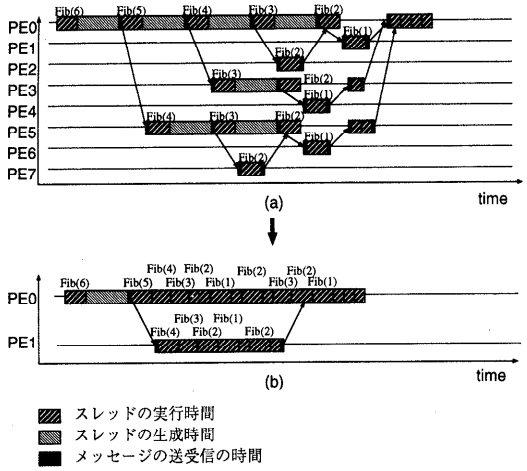


図 5 フィボナッチ関数の粒度調整
Fig. 5 Granularity tuning for Fibonacci function.

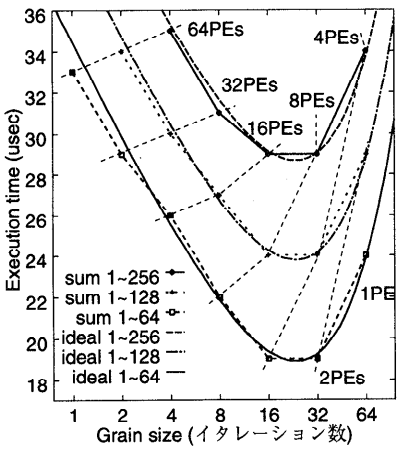


図 6 並列ループの粒度調整効果
Fig. 6 Granularity tuning effect of parallel loop.

列ループによる粒度調整の実験結果を示す．N のループ回数は定数で与えられるものとする．式 (3) に表 1 のパラメータと，コンパイル結果から得られるの命令数を代入すると，最適粒度として，1PE あたりのイタレーション数 23 を得た．**図 6** に 1 から 64，128，256 の和を求めるプログラムを実機上で実行した結果と粒度を求めるときに使った式 (1) から得られる理論的な実行時間を示す．ループ分割は 2 等分するので，実験に用いる粒度は 2 の巾乗になる．使用する PE 数は，(ループ回数)/(粒度) で求めることができる．図 6 において，最適粒度の 23 の付近で実行時間が最小になっていることが分かる．また，それぞれの実行結果が理論値に近い値を示している．粒度が最適粒度より小さい場合，使用する PE 数が多くなり通信時間が長くなるため，全体の実行時間が長くなる．粒度が最適粒度より大きい場合，各 PE での部分ループの実行時間が長くなりすぎるため，全体の実行時間が長くなる．この評価実験により本粒度調整方式が有効であることが分かる．

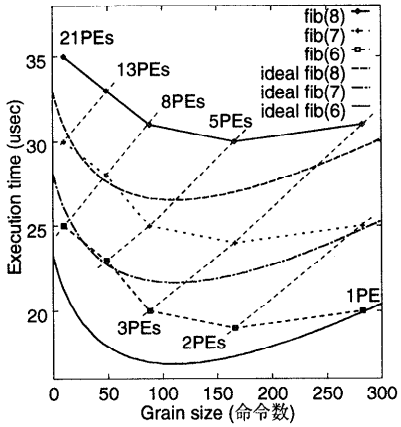


図 7 関数呼び出しの粒度調整効果
Fig. 7 Granularity tuning effect of function call.

6.2 関数呼び出しの粒度調整

フィボナッチ数列を求める関数 $fib(n)$ の粒度調整の実験結果を示す．式 (6) に表 1 のパラメータとコンパイル時に求まる関数の命令数を代入すると，最適粒度として，命令数 114 を得る．$fib(1)$，$fib(2)$ を実行するときの命令数は 10 である．$fib(3)$ を 1PE 上で逐次実行するときの命令数は 49 である．同様に $fib(4)$ は 88 命令，$fib(5)$ は 166 命令，$fib(6)$ は 283 命令である．これらの命令数を粒度とし，**図 7** に，$fib(6)$，$fib(7)$，$fib(8)$ を実機上で実行した結果と粒度を求めるときに使った式 (4) から得られる理論的な実行時間を示す．使用する PE 数は，フィボナッチ関数では 2 分木の構成が完全 2 分木ではないため図 7 に示す数になる．そのため，実行結果と理論式 (4) に差が起きている．実行結果では命令数 166 ($fib(5)$ を逐次実行) のときに全体の実行時間が最小となっていることが分かる．しかし，理論値では，命令数 88 ($fib(4)$ を逐次実行) の方が最適粒度 114 に近い．図 7 のような結果を得た理由は，式 (4) において，節の関数の実行時間と通信時間の部分の命令数を調べると $F+2L+2o_s+2o_r+2A=99$ となるからである．このため，$fib(4)$ の実行を逐次で行い，$fib(5)$ を $fib(4)$ と $fib(3)$ とで並列実行したときの命令数は，$88+99=187$ となる．この値は $fib(5)$ を逐次実行したときの命令数 166 よりも大きい値になるので，全体の実行時間が長くなる．よって，$fib(5)$ を逐次実行したときの方が良い結果となった．

理論値と実験値の差において，縦軸の実行時間の差

はあるが，横軸で最も良い性能を出している粒度では大きな差がない．そのため，本粒度調整方式が有効であることが分かる．

7. 関連研究

坂井ら[10]の文献では，1からNまでの和を求めるプログラム例で，ハードウェアが持つ負荷情報により，動的に粒度調整を行っている．本論文で提案している粒度調整方式は，コンパイラで静的に粒度調整を行っているので，特別なハードウェアを必要としない．

ループを分割する方法として，ある一定の大きさに分割するチャンクスケジューリングがある．ループの粒度調整方式は，チャンクサイズ（粒度）を決めるときの一手法として位置づけることができる．

関数呼び出しの粒度調整では，関数を1つのマクロタスクと考えると，既存の粒度調整方式が適用可能だが，本粒度調整では，複雑なアルゴリズムを用いずに，定式化された式を用いることで，コンパイル時に簡単に粒度調整を行うことが可能となる．

8. おわりに

並列ループと関数呼び出しの粒度調整について，本論文で提案する粒度調整方式の有効性を示した．この粒度調整方式によって，プログラマは使用するPE数を考慮せずに，プログラムの実行時間を最小にすることができる．使用する並列計算機のPE数よりもプログラムの並列度の低い場合に有効である．

SISALのプログラムはおもにループを用いて書かれるので，並列ループの粒度調整方式の適用範囲は広い．並列ループの粒度調整方式の場合は，他の分散記憶型並列計算機にも応用できる．その際には，対象とする並列計算機の実行モデルに合わせた式を作成する必要がある．また，EM-Xにはない他のオーバヘッドやキャッシュ等も考慮しなければならない．関数の粒度調整方式の場合は，SISALのような関数型言語において有効である．

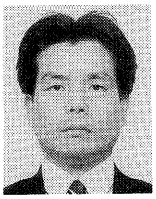
今後の課題として，実用プログラムによる並列粒度調整の効果とその評価，並列粒度調整の適用範囲の拡大があげられる．実用プログラムにおいてループ内で配列の計算処理を行う場合，配列データの分散方法によってプログラムの実行時間が大きく変わる．提案した粒度調整方式に適合した配列の分散方法を検討する必要がある．

謝辞 本研究の一部は文部省科学研究費（基盤研究(B)(2) 課題番号07458055「細粒度並列処理における粒度調整メカニズムの研究」）による．本研究について，助言をいただいた本多弘樹助教授，佐藤直人助手，ならびに研究室のメンバーの方々に感謝いたします．並列計算機EM-Xの利用環境を提供していただいた電子技術総合研究所計算機方式研究室の方々に感謝いたします．

参考文献

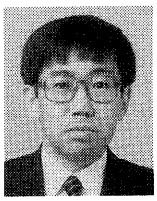
1) Cann, D.C.: The Optimizing SISAL Compiler: Version 12.0, Technical Report UCRL-MA-110080, Lawrence Livermore National Laboratory (1992).
2) Culler, D.E., Karp, R., Patterson, D., Sahay, A., Schauser., K., Santos, E., Subramonian, R. and Eicken, T.V.: LogP: Towards a Realistic Model of Parallel Computation, *Proc. 4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp.1–12 (1993).
3) Kodama, Y., Sakane, H., Sato, M., Yamana, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 22nd Annual Int. Symp. on Computer Architecture*, pp.14–23 (1995).
4) McGraw, J., Skedzielewski, S., Allan, S., Oldehoeft, R., Glauert, J., Kirkham, C., Noyce, B. and Thomas, R.: *SISAL: Streams and Iteration in a single assignment language: Language Reference Manual: Language Version 1.2* (1985).
5) Shankar, B.: Machine Independent Dataflow Code MIDC, *Computer Science Department, Colorado State University* (1995).
6) Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, Addison-Wesley (1991).
7) 高畠志泰，大澤範高，弓場敏嗣：SISALコンパイラへの並列粒度調整機能の組み込み，情報処理学会研究報告，96-HPC-62, Vol.96, No.81, pp.75–80 (1996).
8) 高畠志泰，大澤範高，弓場敏嗣：Doacrossループにおける粒度調整方法の検討，情報処理学会研究報告，97-PRO-14, Vol.97, No.78, pp.1–6 (1997).
9) 高畠志泰，大澤範高，弓場敏嗣，佐藤三久，山口喜教：EM-X用SISALコンパイラにおける並列粒度調整方式，並列処理シンポジウムJSPP '97, pp.37–44 (1997).
10) 坂井修一，児玉祐悦，佐藤三久，山口喜教：超並列計算機における粒度最適化機構の検討，並列処理シンポジウムJSPP '92, pp.235–240 (1992).

(平成9年10月31日受付)
(平成10年4月3日採録)



高畠 志泰（学生会員）

昭和47年生．平成7年電気通信大学電気通信学部情報工学科卒業．平成9年同大学大学院情報システム学研究科情報ネットワーク学専攻博士前期課程修了．現在，同大学院博士後期課程在学．並列化コンパイラ，特にスケジューリング法の研究に従事．

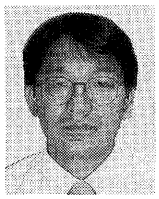


大澤 範高（正会員）

昭和58年東京大学理学部情報科学科卒業．昭和60年同大学大学院理学系研究科情報科学専攻修士課程修了．昭和63年同博士課程修了．平成5年電気通信大学大学院情報システム学研究科助手．理学博士．並列分散システムソフトウェアに興味を持つ．ACM，IEEE-CS各会員．

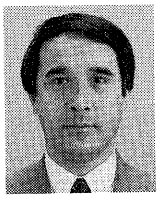
弓場 敏嗣（正会員）

昭和16年9月22日生まれ．昭和41年3月神戸大学大学院工学研究科修士課程修了．野村総合研究所を経て，昭和42年通商産業省工業技術院電気試験所（現，電子技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機などの研究に従事．その間，計算機方式研究室長，情報アーキテクチャ部長等を歴任．平成5年4月より，電気通信大学大学院情報システム学研究科教授．並列処理一般に興味を持つ．工学博士．電子情報通信学会，日本ロボット学会，日本ソフトウェア科学会，ACM，IEEE-CS，各会員．



佐藤 三久（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和61年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成3年，通産省電子技術総合研究所入所．平成8年より，新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術等の研究に従事．日本応用数理学会会員．



山口 喜教（正会員）

昭和47年東京大学工学部電子工学科卒業．同年通商産業省工業技術院電子技術総合研究所入所．以来，高級言語計算機，並列計算機アーキテクチャ，特にデータフロー，細粒度マルチスレッド型並列計算機や実時間並列処理システムなどの研究に従事．現在，情報アーキテクチャ部・並列アーキテクチャラボ・リーダー，筑波大学連携大学院教授．工学博士．平成3年情報処理学会論文賞，平成7年市村学術賞受賞．著書「データ駆動型並列計算機」（共著，オーム社）．電子情報通信学会，IEEE，ACM各会員．