ハイパフォーマンス
コンピューティング 48－14
（１９９３．８．２０）

多段先行評価方式の並列計算機ＥＭ－４上での予備評価

山名早人　佐藤三久　児玉祐悦　坂根広史　坂井修一*　山口喜教

電子技術総合研究所　*新情報処理開発機構

本報告では，多段の条件分岐に渡る投機的実行手法である多段先行評価方式を，データ駆動機構を持つ分散メモリ型並列処理計算機ＥＭ－４上にインプリメントし，予備評価を行った結果を報告する．本方式は，(1)先行評価時の副作用問題の解決，(2)必要プロセッサ数増大問題の解決，(3)条件分岐の多段に渡る先行評価の実現，(4)一般的な並列処理システムを対象，という特徴を持つ．ＥＭ－４上での予備評価の結果，タスクサイズが１０命令の時，制御オーバヘッドを０と仮定した理論性能向上の約５０％，タスクサイズが１００命令の時，約９０％の性能を引き出せることがわかった．

An Experimental Evaluation of the Multi-stage Specurative Execution Scheme on the EM-4 Multiprocessor

Hayato YAMANA　Mitsuhisa SATO　Yuetsu KODAMA　Hirohumi SAKANE　Shuichi SAKAI*
Yoshinori YAMAGUCHI

Electrotechnical Laboratory　*Real World Computing Partnarship
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan

The purpose of this paper is to evaluate a new fast execution scheme of a program with speculative execution on the EM-4 multiprocessor. Conventional Schemes to parallelize programs including conditional branches have some problems. The multi-stage specurative execution scheme enables - (1) solving the side-efects problem, (2) decreaseing the number of processors to execute the specurative execution scheme, (3) jumping multi-stage conditional branches, (4) suiting to general multiprocessor systems. An experimental evaluation shows that the measured speedup ratio is 50% of the theoretical ratio when each task has 10 operations, and it becomes 90 % when each task has 100 operations.

1．まえがき

本報告では，多段の条件分岐に渡る投機的実行手法である多段先行評価方式を，データ駆動機構を持つ分散メモリ型並列処理計算機ＥＭ－４上にインプリメントし，予備評価を行った結果を報告する．

プログラムを並列化する際の問題点は，プログラムが持つ制御依存性，すなわち，条件分岐のために十分な並列性をプログラムから引き出すことができない点にある[1]．条件分岐の結果が実行前に全て既知であるという仮定のもとでプログラムを実行すると，逐次に実行した場合の20～200倍の速度向上が得られる[1][2]．

従来，条件分岐を含むプログラムを並列化する手法がいくつか提案されている．先行評価を用いない手法としては，（1）一般のマルチプロセッサを対象としたタスクの最早実行条件求出法[3][4]，条件分岐１段の先行評価を用いる手法としては，（2）スーパスカラプロセッサやVLIW計算機を対象とした条件分岐１段に限った先行評価方式[5]-[8]，多段の先行評価手法としては，（3）特定のループを対象とした多段の先行評価方式[9][10]，が提案されている．

タスクの最早実行条件求出法では，プログラムをいくつかのタスクに分割し，タスクの終了と分岐方向決定に基づいて，各々のタスクの最早実行条件を求めている．これら２つの条件を用いることにより，制御依存の先行制約を守った上での，最早の実行開始条件を得ることができる．しかし，従来のプログラムは，逐次計算機を対象に書かれているため，制御依存の先行制約を守った場合，並列化の効果は，並列化しない場合を基準とした時の２～３倍に留まる[2][11]．すなわち，タスクの最早実行条件を求めるだけでは，プログラム中から十分な並列性を引き出すことができない．

これに対して，条件分岐１段に限った先行評価方式として，スーパスカラプロセッサやVLIW計算機を対象とした方式が研究されている[5]-[8]．しかし，１段の条件分岐の先行評価では，先行評価しない場合を基準とすると，最大２倍の速度向上しか得られない．

多段の条件分岐に渡る先行評価を用いた実行方式としては，従来，Boolean Recurrence[9]を持つループを対象に研究が行われてきた[1][9][10]．Boolean Recurrenceとは，条件分岐を決定する真理値（Boolean）が，循環参照関係にあることを言う．Boolean Recurrenceを持つループは，条件分岐によって，データ依存関係が変わるため，イテレーション間の並列化ができない．このため，Boolean Recurrenceループを並列化する手法として，制御依存を越えた実行が有効であることが示されている．文献(9)では，専用ハードウェアによる実現方法を提案しているが，対象ループは，ループ内に１つの条件分岐を持つ場合に限定され，多数の条件分岐を持つループに適用できない．

これらの問題に対して，我々は，プログラムをマクロタスクに分割し，マクロタスク間の多段の先行評価方式を一般的な並列処理システム上で提案している．本方式は，従来の先行評価方式が持つ問題を解決する手法であり，(1)先行評価時の副作用問題の解決，(2)必要プロセッサ数増大問題の解決，(3)条件分岐の多段に渡る先行評価の実現，(4)一般的な並列処理システムを対象，という特徴を持つ．

本報告では，多段先行評価方式をデータ駆動機構を持つ分散メモリ型並列計算機EM-4上に実際にインプリメントし，多段先行評価によって生じる制御オーバヘッドを中心に予備評価を行ったので，その結果を報告する．以下，2.において，並列計算機EM-4の概要について述べ，3.で，多段先行評価方式を示す．4.において，EM-4上での予備評価を示し，検討，考察する．

2．並列計算機ＥＭ－４[12]

EM-4プロトタイプは，80台のPEから構成される．PEは，EMC-Rと呼ばれ，ローカルメモリを持ち，サーキュラオメガネットワークで接続されている．

EMC-Rは，RISCアーキテクチャで構成され，パイプラインは逐次の命令実行とネットワークを介した通信とのデータ駆動による同期処理が融合されるように設計されている．ネットワークに対して直接メッセージを送出したり，メッセージによりデスパッチができる．メッセージは，アドレス部とデータ部からなる２ワードの固定長で，これをパケットと呼ぶ．

パケットが到着するとデータ駆動機構により，そのアドレス部で指定されるスレッドがデータ部にある値と共に起動される．そのスレッド処理が終了すると，次のパケットがキューの中から取り出され処理される．

EM-4は，12.5MHzのクロックで動作しており，メモリ参照命令を除く大部分の命令は１クロックで実行される．ネットワークの性能はPEのポート当たり60.9Mbytes/secである．

3．多段先行評価方式

多段先行評価方式を以下に示す．

3.1　対象プログラム

対象プログラムは，以下の条件を満たすとする．

(1) フローグラフ[13]が簡約可能[13]である．

(2) 副プログラムはインライン展開されている．

(3) 先行評価により実行時エラーの発生する条件分岐は，予めユーザが指定し，先行評価の対象外とする．

先行評価の単位として，マクロタスク(以下MT)を用いる．MTは，基本タスク(以下BT)から構成される．BTは，プログラムの一部分であり，制御の入点をその先頭に1つのみ持つ．また，BT集合は，その制御フローグラフが非循環有向グラフで表せるとする．MTは，BTを複製及び融合したものであり，制御の入点をその先頭に1つのみ持つ．

3.2　多段先行評価の実行方式

1つのMTはMTの粒度に応じて複数のPEに割り当て，MTを単位として，実行開始・制御決定・実行停止の3条件（以下MT制御条件）により実行制御する．以下に処理内容を示す．

(1) 非実行状態にあるMTは，実行開始条件あるいは制御決定条件が成立した段階で実行を開始する．実行開始条件によって実行を開始した場合を仮実行，制御決定条件によって実行を開始した場合を本実行と呼ぶ．

(2) 仮実行中のMTについて，実行停止条件が成立した段階で，MTの実行を停止する．

実行開始条件は，MTへのデータ依存を保証し，制御決定条件は，MTへの制御依存を保証するための条件である．実行停止条件は，MTへ制御が到達しないことを保証する条件である．

3.3　マクロタスク生成手法

MT生成手順は，以下の6手順である．

(1) HTG[4]による基本タスク作成

(2) 制御フローグラフ作成

(3) 制御依存グラフ作成

(4) データ依存グラフ作成

(5) 副作用回避のための基本タスク複製

(6) 先行評価無効果部分の基本タスク融合

以下に，各手順を具体的に示す．

(1) HTG(Hierarchical Task Graph)[4]による基本タスク生成　プログラムを非循環の複数のグラフに変換する手法として，HTG[4]がある．まず，プログラムを非循環グラフに変換するために，HTGをプログラムから作成する[4]．HTGは，有向非循環グラフHTG=(HV,HE)で表せる．ノードhv∈HVは，(1) 単純ノード[4];サブタスクを持たないタスク，(2) 混合ノード[4];HTGや他のタスクを含むタスク，(3) ループノード[4];ループボディがHTGを構成するタスク，の何れかである．また辺he∈HEは，ノード間の制御フローを表す．例を図1(a)に示す．HTGにおける階層が同一レベルであるノードをその階層における基本タスク（BT）と呼ぶ．プログラム全体をHTGに変換するには，プログラムを上位層から順に階層に分割し，上記のループノードを除き，各階層内の基本タスク間にループが存在しないようにする．ループノードは，ループボディを1つの階層とみなし，他のイテレーションは別の階層であると考え，制御フローグラフを非循環に保つ．そして、以下で述べる方式をHTGの各階層に適用し，階層間の先行評価を実現する．

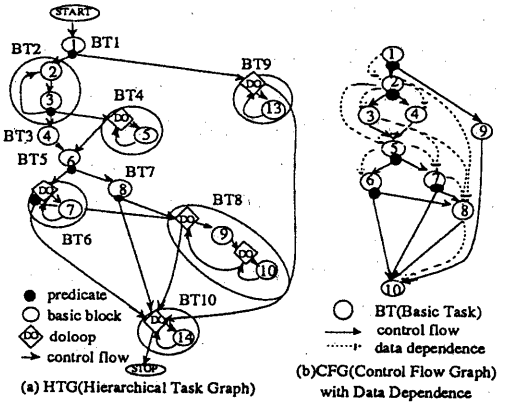


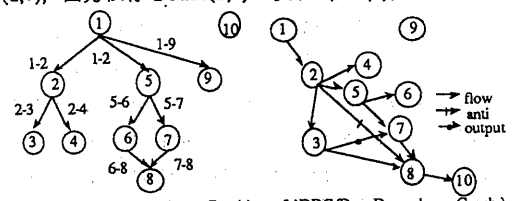
図1　階層タスクグラフと制御フローグラフ

(2) 制御フローグラフ(CFG:Control Flow Graph)作成　制御フローグラフは，HTG[4]から，同一レベルに存在するノードを抜き出したグラフであり，CFG=(V,CFE)で表現する．各頂点v∈VはHTGにおいて同一レベルに存在する基本タスク(BT)を，有向辺cfe=(u,v)∈CFEはBTuからBTvへの制御フローを表す．例を図1(b)に示す．なお，図1(b)では，制御フローの他にデータ依存を点線で示した．

(3) 制御依存グラフ(CDG:Control Dependence Graph)作成　制御依存グラフ[13]は，有向グラフCDG=(V,CDE)で表現される．CDGの各頂点v∈Vは基本タスク，有向辺cde=(u,v)∈CDEはBTuからBTvへの制御依存を表す．各有向辺cde=(u,v)∈CDEに対して，ラベル

u-aを付ける．ラベルu-aは，CFG内のBTuにおいてBTaが選択されることを示す．例を図2(a)に示す．

(4) データ依存グラフ(DDG:Data Dependence Graph)作成　データ依存グラフは，BT間のデータ依存関係を表したものであり，有向グラフDDG=(V,DDE)で表現する．有向辺dde(u,v)∈DDEはBTuからBTvへのデータ依存を表す．データ依存は，フロー依存（定義→参照），逆依存（参照→定義），出力依存（定義→定義）の3つに分類される[16]．データ依存辺dde(u,v)中，フロー依存辺をfdde(u,v)，逆依存をadde(u,v)，出力依存をodde(u,v)で表す（図2(b)）



(a) CDG(Control Dependence Graph)　(b)DDG(Data Dependence Graph)

図2　図1の制御依存グラフとデータ依存グラフ

(5) 副作用回避のための基本タスク複製　先行評価に伴う副作用を回避するために，BTを複製することにより，BT間のデータ依存を制御依存に関わらず一意に定める．ただし，データ依存中，逆依存と出力依存は，変数名変更[14]（Variable Renaming）により除去できるので，以下では，まず，データ依存としてフロー依存のみを用いてBTの複製を行い，後に，変数名変更によって逆依存と出力依存を除去する．

DDG上のノードv∈Vに対して入力しているフロー依存辺fdde(u,v)のソースノードの集合をFDDS(v)={u | fdde(u,v)∈DDE}，ノードvが実行されると仮定した時に必ず実行されるノード集合をDN(v)とした時，ノードvの複製条件は，FDDS(v)⊆DN(v)となる．これは，ノードvが実行される時，ノードvに入力するフロー依存が一意に定まらないことを表す．ここで，DN(v)は，DN(v)={n | ∃p(n)∈P(n),L(p(n))⊆TL(v)}となる[15]．ノードvがノード集合DN(v)以外からフロー依存を持つ場合，ノードvを複製し，全てのノードvに対して，FDDS(v)⊆DN(v)が成立するようにBTを複製する．以下，ノードvの制御確定条件を表すためのラベル間演算を定義する．原子条件であるラベルa-bは，ノードaでノードb側の分岐が選択されることを示し，ノードbへの分岐が決定した時，真となる．ラベル間の演算は，論理演算子∧（論理積）と∨（論理和）からなる．

ノードvの制御確定条件CC(v)は，∃p(v)∈P(v)のラベル集合L(p(v))に属するラベルが全て真になることである．例えば，図2(a)のノード8の制御確定条件CC(8)は，(1-2∧5-6∧6-8)∨(1-2∧5-7∧7-8)と表せる．同様に，ノードvの制御非到達条件（$\overline{CC(v)}$）をラベルを用いて表す．ノードvの制御非到達条件$\overline{CC(v)}$は，∀p(v)∈P(v)について∃a-b∈L(p(v))が偽になることである．例えば，$\overline{CC(8)}$は，($\overline{1\text{-}2}\vee\overline{5\text{-}6}\vee\overline{6\text{-}8}$)∧($\overline{1\text{-}2}\vee\overline{5\text{-}7}\vee\overline{7\text{-}8}$)と表せる．ここで，$\overline{a\text{-}b}$は，ノードa内の条件分岐評価においてノードbへの分岐が選択されないことを示す．従って，ノードaにおける分岐が2分岐であり，それぞれの分岐先がノードb,cであるとすると，ノードaが制御確定し，かつ，ノードcが制御確定することと同値であり，$\overline{a\text{-}b}$=CC(a)∧a-cとなる．この関係を用いると，$\overline{CC(8)}$=1-9∨(1-2∧5-7∧7-10)∨(1-2∧5-6∧6-10)となる．

表1にこれまでに定義した記号及び以下で用いる記号を示し，基本タスク複製手順を表2に示す．

表1 記号定義

dde(u,v)	DDG上ノードuからノードvへのデータ依存辺
fdde(u,v)	={dde(u,v) | データ依存がフロー依存}
adde(u,v)	={dde(u,v) | データ依存が逆依存}
odde(u,v)	={dde(u,v) | データ依存が出力依存}
VN(dde)	ddeの元となる変数名
cde(u,v)	CDG上ノードuからノードvへの制御依存辺
p(v)	=<a0,a1,...,an=v>,ai∈V(ただしa0は入力に制御依存辺を持たない) CDG上ノードvに至るパス
P(v)	={p(v) | p(v)=<a0,a1,...,an=v>,ai∈V} CDG上ノードvに至るパス集合
L(p(v))	={ai-ci | cde(ai,ai+1)がラベルai-ciを持つ,<ai,ai+1>⊆p(v)} CDG上パスp(v)に属するラベル集合
TL(v)	={ai-ci | ∀p(v)∈P(v), ai-ci∈L(p(v))} CDG上ノードvが選択される時必ず真となるラベル集合
DN(v)	={n | ∃p(n)∈P(n),L(p(n))⊆TL(v)} CDG上ノードvが選択される時必ず実行されるノード集合
DDS(v)	={u | dde(u,v)} DDG上ノードvに対して入力しているデータ依存辺dde(u,v)のソースノード集合
FDDS(v)	={u | fdde(u,v)} DDG上ノードvに対して入力しているフロー依存辺fdde(u,v)のソースノード集合
CC(v)	ノードvの制御確定条件 ∃p(v)∈P(v)のラベル集合L(p(v))に属するラベルが全て真
$\overline{CC(v)}$	ノードvの制御非到達条件 ∀p(v)∈P(v)のp(v)について∃a-b∈L(p(v))が偽

表2　基本タスク複製手順

手順（1）　FDDS(v)⊈DN(v)となる∃vについて以下の処理を行う．FDDS(v)⊈DN(v)となるノードが存在しない場合には，手順（8）へ．

手順（2）　ノードvが実行される時必ずしも実行されるとは限らないDDG上のノードvへのソースノード集合NDS(v)=FDDS(v)−{DN(v)∩FDDS(v)}を求め，∃u∈NDS(v)について以下の処理を行う．

手順（3）　CC(u)，$\overline{CC(u)}$，CC(v)を求める．

手順（4）　ノードvをCDG及びDDGから削除し，ノードvの複製をCDG及びDDG上に2つ作成〈v',v''〉する．

手順（5）　ノードv'は，ノードuからのデータ依存が常に存在するようにするためにCC(v')=CC(v)∧CC(u)，ノードv''は，ノードuからのデータ依存が全く存在しないようにするためにCC(v'')=CC(v)∧$\overline{CC(u)}$となる制御依存辺を以下の制御依存

辺付加手順によりCDG上に付加する．

手順（6） DDG上のノードv',v''に対して，ddeを付加する．元々のノードvに入力していた$\forall u \in DDS(v)$について，$CC(u) \wedge CC(v')$が恒常的に偽となる場合（相反する条件が存在する場合）を除いてdde(u,v')を付加する．ノードv''に対しても$\forall u \in DDS(v)$について，$CC(u) \wedge CC(v'')$が恒常的に偽となる場合を除いてdde(u,v'')を付加する．最後に，$\forall w \in \{w \mid dde(v,w) \in DDE\}$について，dde(v',w)及びdde(v'',w)を付加する．

手順（7） 手順（1）に戻る．

手順（8） 非到達fdde削除 $\forall fdde \in \{fdde(u,v) \mid \exists w \in V, w \neq v, VN(fdde(u,v))=VN(fdde(w,v))=VN(odde(w,u))\}$，つまり，データの2重定義によりノードvにデータ依存が到達しないfddeをDDG上から削除する．

手順（9） 重複ノード削除 $\forall v \in \{v \mid \exists v' \in V, v' \neq v, DDS(v)=DDS(v')$, v'はvを複製したノード$\}$をDDG及びCDG上から削除する．つまり，ノードvに入力するデータ依存がノードvを複製したノードv'と等しい場合，ノードv'を削除する．また，$\forall w \in \{w \mid dde(v,w)\}$について，dde(v',w)をDDG上に付加し，$CC(v')=CC(v) \vee CC(v')$となる制御依存辺を以下の制御依存辺付加手順によりCDG上に付加する．削除対象ノードが無くなるまで手順（9）を繰り返す．

手順（10） 保証不用dde削除 $\forall dde \in \{dde(u,v) \mid u \neq v, fdde(u,*)fdde(*,v)$が存在$\}$をDDGから削除する．但し，fdde(u,*)fdde(*,v)は，ノードuからvに1個以上のノードを経由したフロー依存の存在を示す．dde(u,v)はfdde(u,*)fdde(*,v)により保証できるので削除する．

手順（11） 変数名変更 $\forall v \in \{v \mid \exists w \in V, w \neq v, adde(w,v)\}$においてVN(adde(w,v))を，$\forall v \in \{v \mid \exists w \in V, w \neq v, odde(w,v)\}$においてVN(odde(w,v))を変数名変更し，全てのadde及びoddeを削除する．また，複製されたノード内で定義される変数についても変数名変更する．これにより2重定義に伴う副作用を回避する．

制御依存辺付加手順 ノードv'の制御確定条件を原子条件であるラベルの積和で表したものを$CC(v')=(a_{11}\text{-}b_{11} \wedge a_{12}\text{-}b_{12} \wedge \cdots \wedge a_{1(n1)}\text{-}b_{1(n1)}) \vee (a_{21}\text{-}b_{21} \wedge \cdots \wedge a_{2(n2)}\text{-}b_{2(n2)}) \vee \cdots \vee (a_{m1}\text{-}b_{m1} \wedge \cdots \wedge a_{m(nm)}\text{-}b_{m(nm)})$とする．ここで，積和は和間の項数が最小になるように簡略化されているものとする．制御確定条件の定義より各々の和項は，CDG上のノードv'へのある1本のパスを示し，積項のラベルは，該当パス上に存在するラベルを示す．従って，m個の積項からなる場合は，ノードv'にm個のパス$(p_1(v'), p_2(v'), \ldots p_m(v'))$が存在し，i番目のパスのラベル集合は，$L(p_i(v'))=\{a_{i1}\text{-}b_{i1}, a_{i2}\text{-}b_{i2}, \ldots, a_{i(ni)}\text{-}b_{i(ni)}\}$となる．以上より，CDG上のノードv'に対して，m個のパスを付加し，各々のパスに対してラベル$L(p_i(v'))=\{a_{i1}\text{-}b_{i1}, a_{i2}\text{-}b_{i2}, \ldots, a_{i(ni)}\text{-}b_{i(ni)}\}$を付加する．この時，ラベルa-bを持つ制御依存辺がノードaから出力されるようにノードaをダミーノードとして付加する．このダミーノードは，ノードv'の制御確定条件を示すためにだけ用いる．同様にノードv''についても制御依存辺を付加する．

図2のプログラムに適用した場合の例を示す．手順(1)に従いFDDS(v)とDN(v)を全ノードについて求める．$FDDS(v) \subseteq DN(v)$となるvは{8,10}である．まずv=8について複製処理を行う．手順(2)に従い，NDS(v)を求めると，NDS(8)={3,7}となる．すなわち，ノード8へ入力するノード3及び7からのデータ依存が制御によって一意に定まらないことを示す．ここでは，まずu=3について手順(3)以下の処理を行う．CC(3)=1-2∧2-3，$\overline{CC(3)}$=1-9∨(1-2∧2-4)，CC(8)=(1-2∧5-6∧6-8)∨(1-2∧5-7∧7-8)となり，手順(4)(5)より，複製されるノードを8',8''とすると，CC(8)=CC(8)∧CC(3)=(1-2∧2-3∧5-6∧6-8)∨(1-2∧2-3∧5-7∧7-8)，CC(8')=CC(8)∧$\overline{CC(3)}$=(1-2∧2-4∧5-6∧6-8)∨(1-2∧2-4∧5-7∧7-8)となる．ノード8'，8''に制御依存辺を追加し，手順(6)によりデータ依存辺を付加する．以上の手順(1)～(7)を$FDDS(v) \subseteq DN(v)$となるvが存在しなくなるまで繰り返すと，図2のプログラムは，図3のように複製される．

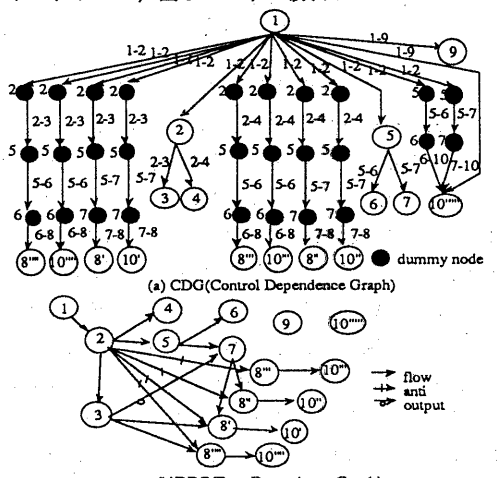


図3 基本タスク複製手順(1)-(7)適用後のCDGとDDG

次に，データの2重定義によりデータ依存が到達しないフロー依存を手順(8)により削除する．図3においてfdde(3,8)，fdde(7,8)，odde(3,7)が同一変数による依存であるとすると，ノード3で定義された後，再びノード7で定義され，ノード3からのフロー依存はノード8'に到達しないのでfdde(3,8)を削除する．この削除により，DDS(8)とDDS(8')が等しくなるので，手順(9)によりノード8'と8''を融合する．同様にノード10'と10''を融合する．融合後のノードを8'，10'で表す．手順(10)では，保証する必要のないデータ依存を削除する．これによりadde(2,8')，adde(2,8''')が削除される．最後に，手順(11)の変数名変更により，odde(3,7)とadde(2,8''')を削除する．さらに，複製されたノード内(8',8'',8''',10',10'',10''',10'''')で定義される変数名も変更し先行評価時の2重定義による副作用を回避する．以上の結果得られるCDGとDDGを図4に示す．

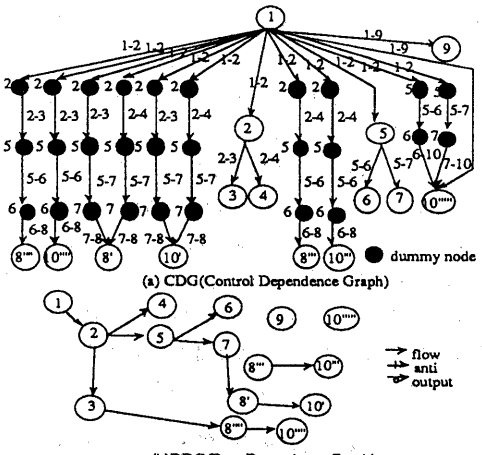


図4　基本タスク複製手順(11)適用後のCDGとDDG

(6) 先行評価無効果部分の基本タスク融合

データ依存と制御依存の関係を用いて，BTを融合した場合も先行評価の効果を失わないように，BTを融合しMTを構成する．2つの基本タスクBTaとBTbを融合した時，先行評価の効果を失わないための必要十分条件は，実行開始条件，すなわちデータ依存の解決する時刻，が融合の前後で変わらないことである．これは，融合によってデータ依存の解決する時刻が遅れた場合，先行評価の効果が小さくなることから明らかである．また，3.1で定義したように，MTは，その先頭に1つの制御の入点を持たなければならない．

まず，実行開始条件が融合前後で変らないための条件を示す．BTa及びBTbへ入力するデータ依存辺が次の何れかの条件を満たせばよい．

$$DDS(BTa)=DDS(BTb) \quad (1)$$

$$\exists dde(BTa,BTb),\ \forall BTx,\ BTx \in \{BT \mid BT \neq BTa,\ BT \in DDS(BTb)\} \rightarrow dde(BTx,*)dde(*,BTa) \quad (2)$$

但し，式（2）において，dde(BTx,*)dde(*,BTa)は，BTxからBTaへのデータ依存がBT*（*は1個以上の任意BT）を経由して存在することを示す．式（1）は，BTa及びBTbへ入力するデータ依存が全く等しい場合を示す．式（2）は，BTaの実行終了によりBTbへ入力するデータ依存を保証できる場合を示し，融合後のMTの実行開始条件はBTaの実行開始条件となる．

次に，MTがその先頭に1つの制御の入点を持つためには，BTa及びBTbへ入力する制御依存辺が次の何れかの条件を満たせばよい．

$$CC(BTa)=CC(BTb) \quad (3)$$

$$CC(BTa)\wedge\{a\text{-}x\}=CC(BTb) \quad (4)$$

但し，式（4）において，{a-x}はBTa内で解決できる制御依存のラベル集合の部分集合を表す．式（3）は，BTa及びBTbの制御依存が等しいことを示し，融合されたMTは唯一の制御の入点を持つ．式（4）は，BTaの制御条件及びBTa内で解決される制御条件によりBTbの制御依存を保証できる場合を示す．すなわち，BTa終了時にBTbの制御条件が確定するので，融合されたMTはCC(BTa)を制御条件として持てばよい．

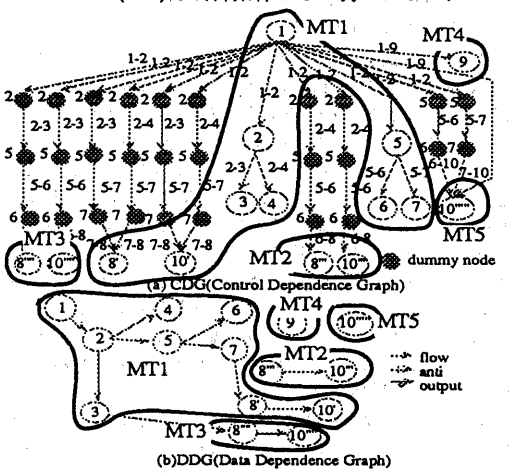


図5　融合手順適用後融合されたマクロタスク

以上より，融合条件は，式(1)(2)の何れかが真になり，かつ，式(3)(4)の何れかが真となる4条件となる．本条件を融合できるBT対がなくなるまで適用し，得られたブロックをマクロタスクと呼ぶ．本融合条件を図4の複製後の基本タスク群に適用すると，図5に示すように，5つのマクロタスクに融合される．

3.4 マクロタスク制御条件

制御条件は，実行開始，制御決定，実行停止の3条件である．各制御条件は，制御の確定したBTの実行終了及び制御の確定したBT内での条件分岐決定の2種類の原子条件により表現する．

マクロタスクMTiの実行開始条件　各MTへ入力するデータ依存は，制御によらず一意である．従って，実行開始条件は，MTiに入力するデータ定義終了，すなわち，DDS(MTi)に含まれる全てのBTの終了のみを保証すればよい．原子条件として基本タスク番号を用いる．

マクロタスクMTiの制御確定条件　MTiに対する制御確定条件は，CC(MTi)である．すなわち，$\exists p(MTi) \in P(MTi)$についてL(p(MTi))に含まれるラベルが全て真になることが条件となる．

マクロタスクMTiの実行停止条件　MTiに対する制御非到達条件は，$\overline{CC(MTi)}$である．すなわち，$\forall p(MTi) \in P(MTi)$についてL(p(MTi))に含まれる1つ以上のラベルが偽になることが条件となる．

制御確定条件及び実行停止条件の算出において，例えば，本例における1-2∧2-3のように2に至る制御依存辺が1本であり，2が制御確定する時には必ず1-2が真になるような場合，1-2を条件から省略する．これは，2-3の条件が2が制御確定していることを前提としているからである．つまり，a-b∧c-dにおいて，∀n∈{c及びcを複製したノード}に対してTL(n)∋a-bであるとき，abを省略する．表3に，図5のMTの制御条件を示す．

表3　マクロタスク制御条件

MT	実行開始条件	制御決定条件	実行停止条件
1	True	True	False
2	True	2-4∧6-8	1-9∨2-3 ∨(2-4∧5-7) ∨(2-4∧6-10)
3	3	2-3∧6-8	1-9∨2-4 ∨(2-3∧5-7) ∨(2-3∧6-10)
4	True	1-9	1-2
5	True	1-9∨6-10 ∨7-10	6-8∨7-8

4．EM－4上での予備評価

多段先行評価方式を実機上で実行する際の問題点は，(1)多段先行評価を行うための制御オーバヘッドの増大，(2)先行評価に伴うメモリバンド幅の増大である．今回は，これらの問題点の内，(1)の制御オーバヘッドの先行評価の効果に与える影響を評価する．(2)のメモリバンド幅の増大は，アプリケーションに大きく左右されることが予想されるため，今後，実際のアプリケーションによる評価を行う．

4.1　EM-4上への多段先行評価方式実装方法

3.4で示したマクロタスク制御条件に基づいてマクロタスクの実行開始・制御決定・実行停止を行う．プログラミングは，EM-4用のプログラミング言語の一つであるEM-C[17]を用い，マクロタスクの制御に伴うオーバヘッドを削減するため，図6に示すように，マクロタスクの実行開始・制御決定・実行停止を異なるスレッドとし，別々のプロセッサに割り当てる．さらに，各制御条件の判定を高速に行うため，各タスク制御条件から，ワーキングセットを抽出し，そのワーキングセット内で各制御条件の判定を行う．ワーキングセットとは，残り1つの原子条件により成立するか否かを判断できる制御条件集合である．例えば，表3のMT2の制御決定条件は，原子条件2-4が成立し，残りの条件が原子条件6-8のみになった時にワーキングセットに入る．また，マクロタスクのPEへのスケジューリングは，予め静的に定義し，スケジューリングに伴う影響を本評価から排除してある．

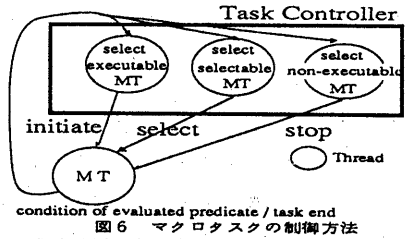


図6　マクロタスクの制御方法

4.2　評価対象プログラムと評価方法

評価対象プログラムとして，非ループ型とループ型の2つのトイプログラムを用いた．図5に示したプログラムと，図7に示すBoolean Recurrenceループである．何れのプログラムにおいても，BTは，for (i=0; i < TASK_SIZE; a(i) = a(i) ×b(i), i++)で構成し，タスクサイズを変えて評価を行った．また，データ通信に伴う影響を排除するため，各配列は自PE内のものを用い，図5，図7中データ依存がある場合には1データのみを通信すると仮定した．

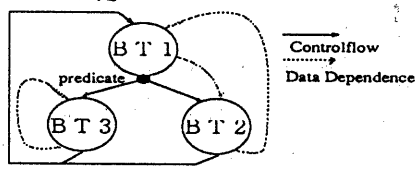


図7　評価に用いたBoolean Recurrenceループ

4.3　評価結果

図5のプログラムを先行評価した際のオーバヘッドを考えない理論的な性能向上に対する実際の性能向上率を図8に示す．先行評価では，選択されるパスにより評価結果が異なるため，図8では全てのパスの平均値を示した．図8より，タスクサイズが10命令時，理論的な速度向上の50%，タスクサイズが100命令時，90%の効果を得ることが可能であることがわかる．しかし，タスクサイズが1命令である場合，逆に5%の性能低下がみられる．

次に，図7のプログラム(繰返し回数=10)を先行評価した際の速度向上率，及び，各タスクサイズにおいて実際に先行評価された先行評価段数を示した．先行評価段数とは，制御が決定するまでに起動することのできたMT内に含まれる条件分岐段数を表す．ループ型のプログラムでは，3.4に示したMT制御条件を動的に生成するため，先行評価される段数は，動的に決定

される．また，条件分岐は50%:50%の確率でランダムにBT2:BT3が選択されるものとした．図9より，タスクサイズが大きいほど，多数段の先行評価ができ，速度向上率も上がることがわかる．ループ型のプログラムの場合，タスク制御条件を動的に生成するため，同等の効果を得るためには，MTの粒度が非ループ型に比べて大きくなくてはならない．

本予備評価により，非ループ型のプログラムでは，タスクサイズが10命令程度で理論性能の50%程度を引き出すことができ，実際のアプリケーションを対象とした場合も，効果が期待できることがわかった．しかし，ループ型のプログラムの場合，タスクサイズをある程度大きくしなければ，多段の先行評価が期待できない．解決策としてループアンローリングが考えられが，これは，今後の課題である．

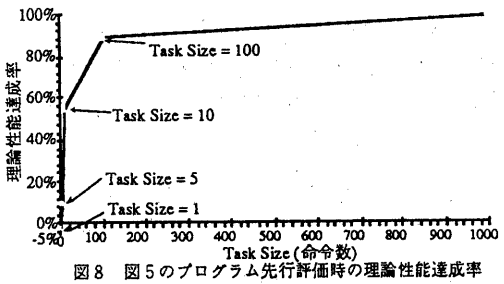


図8　図5のプログラム先行評価時の理論性能達成率

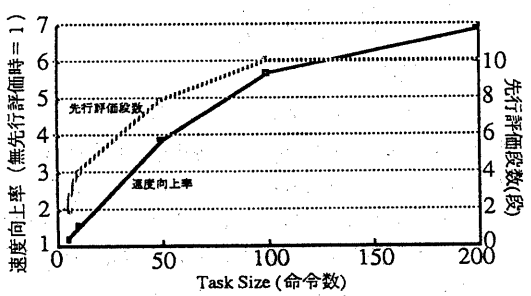


図9　図7のプログラム先行評価時の速度向上率と先行評価段数

5．むすび

本報告では，多段先行評価方式を並列処理計算機EM-4上にインプリメントし，予備評価を行った結果を示した．この結果，タスクサイズが10命令程度以上であれば，理論的な速度向上の50%の効果を得ることができ，タスクサイズが100命令になれば，90%の効果を得ることが可能であることがわかった．

今回のインプリメントでは，タスクの制御を集中して行っており，タスクサイズが小さい場合，タスク制御のオーバヘッドの影響が大きい．しかし，タスク制御を分散して行うことができれば，制御のオーバヘッドをさらに削減できる．今後は，このようなタスク制御の分散化，タスクの動的スケジューリング，実際のアプリケーションを用いた評価，多段先行評価を自動的に行うコンパイラの開発を行っていく予定である．

謝辞

本研究を遂行するにあたり御指導，御討論いただいた太田情報アーキテクチャ部部長ならびに計算機方式研究室の同僚諸氏に感謝いたします．

文献

(1) E.Riseman,C.Foster:"The Inhibitation of Potential Parallelism by Conditional Jumps", IEEE Trans. Comput., C-21,12,pp.1405-1411 (1972).

(2) M.S.Lam,R.P.Wilson:"Limits of Control Flow on Parallelism", Proc. of 19th Int. Symp. on Computer Architecture, pp.46-57 (1992).

(3) 本多,岩田,笠原:"Fortran プログラム粗粒度タスク間の並列性検出手法",信学論(D-I),J73-D-I,12,pp.951-960 (1990).

(4) Milinf Girkar,C.D.Polychronopolulos: "Automatic Extraction of Functional Parallelism from Ordinary Programs", IEEE Trans. Parallel & Distributed Syst.,3, 2, pp.166-178 (1992).

(5) R.M.Tomasulo:"An Efficient Algorithm for Exploiting Multiple Arithmetic Units",IBM J. Res. & Dev., 11,1,pp.25-33 (1967).

(6) A.Aiken,A.Nicolau:"A Development Environment for Horizontal Microcode",IEEE Trans. Soft. Eng., 14,5,pp.584-594 (1988).

(7) 原,納富,久我,村上,富田:"SIMP(単一命令流/多重命令パイプライン)方式に基づく改良版スーパースカラ・プロセッサの構成と処理",信学技報,CPSY-90-55,pp.103-108 (1990).

(8) 丸島,西,大沢,中崎:"パイプライン計算機における分岐仮実行アーキテクチャ",情処学研報,ARC-87-2,pp.1-8(1991).

(9) Utpal Banerjee,D.D.Gajski:"Fast Execution of Loop with IF statements",IEEE Trans. on Comput.,C-33,11,pp.1030-1033 (1984).

(10) A.K.Uht: " Requirements for Optimal execution of Loops with Tests", Proc. of ACM Int. Conf. on Supercomputing'88, pp.230-237 (1988).

(11) 山名,石崎,安江,村岡:"並列処理システム－晴－における条件分岐の先行評価制御方式", 情処学研報,ARC-89-19, pp.135-142 (1991).

(12) S.Sakai,Y.Yamaguchi,K.Hiraki,Y.Kodama,T.Yuba:"An Architecture of a Dataflow Single Chip Processor", Proc. of 16th Ann. Symp. on Comp. Arch., pp.46-53(1989)

(13) A.V.Aho,R.Sethi,J.D.Ullman: "Compilers: Principles, Techiniques and Tools", Addision-Wesley (1986).

(14) D.A.Padua,M.J.Wolfe: "Advanced Compiler Optimizations for Supercomputers", Communications of the ACM, 29, 12, pp.1184-1201 (1986).

(15) 山名,安江,石井,村岡:"先行評価に適したマクロタスク生成手法",情処全大,8L-5(1993)

(17) 佐藤,児玉,坂井,山口:"並列計算機EM-4 上での共有メモリベンチマークの実行",信学技報,CPSY92-61 ,pp.49-56(1992)