領域分割レジスタ生存グラフを用いたレジスタ割付けへの 動的計画法の適用

淺原 英 雄 近 藤 伸 宏竹 古 関 12011 昭††† 濹 彰 小松 秃 深 良

本稿では,命令レベル並列プロセッサ向けの新しいレジスタ割付け手法を提案する.命令レベル並 列性(ILP)を抽出する手法として,我々はシリーズパラレル型レジスタ生存グラフを用いた手法を 提案してきた.しかし,今までの手法はワンパスのヒューリスティクスを用いた手法であったため, プログラム構造によっては十分に最適な解を得ることができないという欠点があった.そこで,解空 間を広く探索し,かつ,動的計画法を用いて計算量が増加するのを抑制できるアルゴリズムを開発し た.本稿では,まずレジスタ割付けに関する問題を整理し,本手法のアルゴリズムと適用例を示し, その評価を行う.

Applying Dynamic Programming Technique to Register Allocation Based on Region Partitioned Register Existence Graph

HIDEO ASAHARA,[†] NOBUHIRO KONDOH,^{††} AKIRA KOSEKI,^{†††} HIDEAKI KOMATSU^{†††} and YOSHIAKI FUKAZAWA[†]

We introduce a new register allocation algorithm for instruction-level parallel processors. We have suggested a method using Series-Parallelized Register Existence Graph in order to extract ILP in a program. However, this method does not always give the optimum result because it only uses one-pass heuristics. Then, we developed a new algorithm which can search the broader possibility of solutions and uses dynamic programming in order to reduce the cost of computation. In this paper, we clarify some problems on register allocation techniques, and give our algorithm with some examples, and its evaluation.

1. はじめに

並列プロセッサに対するレジスタ割付けにおいては, 実レジスタを再利用することで発生する逆依存が問題 視されてきた.そのため,スピルコードの最少化に加 え,並列実行を阻害しない実レジスタの再利用の実現 が目指されてきている^{1)~4)}.

我々はこれまでに,命令レベル並列プロセッサ向けの レジスタ割付け手法として,レジスタの生産と使用の 関係をエッジで表すレジスタ生存グラフを用いて命令 レベル並列性(ILP: Instruction-Level Parallelism) を抽出する手法を提案してきた^{3),4)}.この手法は,あ

††† 日本 IBM 株式会社東京基礎研究所 Tokyo Research Laboratory, IBM Japan, Ltd. るタイミングに生存する仮想レジスタ数が利用可能実 レジスタ数を超えないために,あらかじめ命令の実行 順序に制約を加えておくことで,コードスケジューラ との統合を実現している.しかし,この制約を加える アルゴリズムはアドホックであり,プログラム構造に よっては十分に質の良い解を得ることができないとい う欠点があった.そこで,本稿では動的計画法(DP: Dynamic Programming)を用いて計算量の増加を抑 制しながら解空間をより広く探索するアルゴリズムを 用い,より高い ILP を引き出す新しいレジスタ割付 け手法を提案する.

本稿では,まず,レジスタ割付けにおける問題点を 示し,関連手法を紹介する.そして,本手法の特徴, アルゴリズムを説明し,評価を行う.

研究の目的

レジスタアロケータは,同時に使用するレジスタ数 を利用可能な実レジスタ数以下に低減する機能と,仮

[†] 早稲田大学理工学部

School of Science and Engineering, Waseda University †† 株式会社東芝セミコンダクター社システム LSI 事業部

Semiconductor Company System LSI Division, Toshiba Corp.

想レジスタを実レジスタに割り当てる機能の2つの機 能を持つことが多い.本稿中では,前者が行う作業を 最大干渉度低減操作と呼ぶ.ここで,ある仮想レジス タどうしが干渉しているとは,互いの生存区間が重な ることを指し,両者を同一レジスタに保存できない. また,干渉度とは,あるタイミングにおいて干渉しあ う仮想レジスタの数を示し,最大干渉度とは,レジス タ割付け対象中の最大の干渉度を指す.

ILP を低下させないレジスタ割付けを実行するには, 3つの問題点がある.問題点1は,レジスタの再利用 による逆依存の発生が ILP 抽出の妨げになる点であ る.問題点2は,レジスタアロケータが使用レジスタ 数を減らす向きに働くのに対し,コードスケジューラ は並列度を高め、レジスタプレッシャを高める向きに 働いてしまう点である.ILPを十分に引き出すには, 両者の協調動作が必要である.レジスタ割付けを先に 行うポストパス型は、レジスタアロケータがレジスタ の再利用率を高め,問題点1を強めてしまう結果に なり, ILP の抽出を阻害してしまう. その逆順序のプ リパス型は, コードスケジューラが抽出した ILP を, レジスタアロケータが壊してしまう. $gcc^{5)}$ など,両 者を交互実行により協調させる手法では,計算コスト が著しく高まる.このように,レジスタアロケータと コードスケジューラの協調動作を考慮することは, ILP を抽出するためにきわめて重要な問題である.問題点 3は,レジスタ割付けが NP 完全問題であり,多項式 時間で厳密な最適解を得ることができない点である. そのため,バックトラックのない,一意に解が求まる アルゴリズムを採用し,少ない計算量で比較的良い解 を得るアプローチが有効である.しかし,解を得る際 に解候補の比較や評価などを行わないことから,採用 したアルゴリズムが得意としないプログラム構造に対 しては十分な ILP を保持できないという問題をはら んでいる.

以上の問題を解決するレジスタ割付け手法が望まれ ている.

3. 関連研究

最もよく知られたレジスタ割付け技法として,レジ スタ彩色法があげられる⁶⁾.仮想レジスタをノード,干 渉をエッジで表したレジスタ干渉グラフに対し,彩色 問題を解くことで発見的にレジスタ割付けを行う.そ の単純さと強力さから,現在の大部分のコンパイラの 基本アルゴリズムに採用されているが,問題点1が考 慮されていない.これを考慮した後続研究として,並 列化レジスタ干渉グラフを用いたレジスタ彩色法があ

1 if(r2!=0){ 2 r6=r3<<2 3 store(r6,r3) 4 }else{ 5 r7=r4>>2 6 r8=r4+4 7 store(r7,r6) 8 r9=Load(r5) 9 r10=r9+1 10 store(r10,r5) 11 r11=r2+1 12 }	1 cc1=r2&0 2 (!cc1) r6=r3<<2 3 (!cc1) store(r6,r3) 4 (cc1) r7=r4>>2 5 (cc1) r8=r4+4 6 (cc1) store(r7,r8) 7 (cc1) r9=Load(r5) 8 (cc1) r10=r9+1 9 (cc1) store(r10,r5) 10(cc1) r11=r2+1
(i) 変換前 (ii)SSA変換,ガード付加後
図 1 サン	プルコード
Fig. 1 A sa	ample code.
(r3)	r2) M:, (r11)



図 2 レジスタ干渉グラフ Fig. 2 Register interference graph.

る^{1),2)}. これらの手法では, レジスタ干渉グラフに並 列性を保持するためのエッジを加えた並列化レジスタ 干渉グラフを用いて彩色問題を解くことで, 問題点1 を解決している.しかし, 並列実行の可能性に関して 安全な見積りを行うため, 実際には干渉しないノード どうしにもエッジを張ってしまい, 無駄なスピルコー ドが挿入されてしまう.

実際に,図1(i)のサンプルコードにSSA変換⁷⁾ を施し,各命令に実行条件を示すガードを付加した 図1(ii)に対する両手法のグラフを図2に示す.プロ グラムの並列性が高い場合,このように完全グラフに 近くなるため,余分なスピルコードの挿入が頻発して しまう.

これに対して, 我々は GPDG (Guarded Program Dependence Graph)⁸⁾ から生成される, レジスタ生 存グラフを利用するアルゴリズムを提案し, 干渉グラ フを利用した際の問題を解決してきた^{3),4)}.

GPDGとは, PDG⁹⁾を拡張したグラフであり, プ ログラム中の最内ループ中のデータ依存,制御依存の 関係を表現するのに用いられる.最内ループの入口と 出口を示す START ノードと END ノードを持ち,そ の他のノードは各命令を表している.各ノードには, ガードが付加され,ガードとデータに関する生産と使

Sep. 2002



図 3 図 1 に対する GPDG Fig. 3 GPDG for Fig. 1.



図 4 図 3 に対するレジスタ生存グラフ Fig. 4 Register existence graph for Fig. 3.

用の依存関係を,有向エッジを用いて表現する.これ により,GPDG 上では,制御依存をデータ依存と同 等にとらえることができる.図1から得た GPDG を 図3に示す.

レジスタ生存グラフとは,仮想レジスタを表すノードと,その値の生産と使用の関係を表すエッジを持つ グラフである.また,グラフ中にノードの生存時間の 概念が導入され,同時刻に生存している仮想レジスタ どうしは等時刻線と呼ばれる1本の線で貫かれる.同 一の等時刻線が貫くノード数を該当サイクルの干渉度 とする.図3に対するレジスタ生存グラフを図4に 示す.

レジスタ生存グラフを用いると,並列化レジスタ干 渉グラフにおいては事実上干渉しないノードどうしに も張られていたエッジを切ることが可能になる.

問題点2を考慮した手法として,レジスタ生存グラフを参照し,最大干渉度が実レジスタ数を超えているサイクルに生存するノードの中から,クリティカルパス長に影響の少ないノードを選択し,その実行タイミ

ングを遅延させていく手法が提案されている³⁾.この 手法により高い ILP を抽出できたが,レジスタプレッ シャが高い場合に良い結果を出すことができなかった. その理由は,最大干渉度低減操作において,並列度の 高い部分に優先してレジスタ資源を割り付けるのでは なく,クリティカルパスに影響のない仮想レジスタを スピルさせるというヒューリスティクスを用いていた からである.

これに対し,シリーズパラレル化¹⁰⁾したレジスタ 生存グラフを用いることで,この欠点を補う手法も提 案されている⁴⁾.この手法ではコード中の並列度の高 さを数値化し,並列度が高い部分に優先して実レジス タを割り付けるヒューリスティクスを用いることで前 述の欠点を補っている.これにより,レジスタプレッ シャが高い場合にも対応できるアルゴリズムとなった. しかし,バックトラックを行わずにヒューリスティク スにより解を一意に決めてしまう手法であったため, 問題点3は克服されていない.

問題点2を考慮した他の手法に関しては,文献11) で総合的な性能評価が示されているように,数多くの 研究が行われている.

その 1 つとして, Berson らの Resource Spackling¹²⁾ がある.命令をノードとし,あるリソースの再 利用が可能な命令へのエッジを持つ Reuse DAG を 用いる.レジスタの Reuse DAG から必要レジスタが 超過する命令群を解析し,それを回避するためのエッ ジを実行ユニットの Reuse DAG に追加することで, コードスケジューリングとレジスタ割付けの統合を図 る.しかし,再利用が可能な命令を決定するアルゴリ ズムについては記述がなく,この性質によっては不必 要なスピルコードを挿入してしまうことがある.また, この手法でも問題点 3 は克服されていない.

本稿ではこれらの関連研究をふまえ,問題点1~3 の解決を目指し,DPを用いて計算量を抑制しつつ解 空間をより広く探索してILPを抽出するアルゴリズ ムを提案する.

4. 本手法の特徴

4.1 動的計画法の活用

本手法では,NP 完全問題であるレジスタ割付けを, DP を用いて探索コストを抑制しながらより広く解空 間を探索するアルゴリズムを用い,より高い ILP を抽 出する.これにより,問題点3を解決する.DP は最 大干渉度低減操作の際に利用される.レジスタ生存グ ラフを複数の小グラフの集合と見なし,各小グラフを DP のアルゴリズムの下で実レジスタ数とクリティカ ルパス長の制約を加えながら接続していくことで最大 干渉度が実レジスタ数以下に低減されたレジスタ生存 グラフを得る.ここで,接続とは2つの小グラフを1 つのグラフにまとめることをいう.この際,接続後の 最大干渉度が利用可能実レジスタ数を超えない限り, クリティカルパス長が最短となる接続を行う.小グラ フの接続順序によって,得られるレジスタ生存グラフ のクリティカルパス長が変化するため,DPを用いな い場合は全組合せの接続を試行する必要がある.しか し,DPを適用することで,接続結果を複数回再利用 することができる.また,良い解を導くと予想される 接続結果を優先して用いることができるため,大幅に 枝刈りされる.DPによる枝狩りの例は6章で,また, 枝狩りによる計算量の抑制度は8章で述べる.

4.2 論理排他性の考慮

本手法では,実レジスタの有効な再利用のため論理 排他性を利用する.ここで,命令 A, B が論理排他 性を持つとは,両者の実行結果がともに有効になる実 行条件が存在しないことを意味する.この条件下で, 命令 A, B の目的レジスタの生存範囲が重なる場合, レジスタを共有することができる.書き込みの際には, ガードの値から実行結果が有効にされる命令を選択 し,その値を書き込めばよい.プレディケートなどを サポートするアーキテクチャの場合,論理排他性を利 用したレジスタの有効活用を図ることができる.本手 法では,ガードをサポートするアーキテクチャを対象 にしており,プレディケートでは難しい複雑な論理排 他性の存在も効率的に行われる.

4.3 領域分割レジスタ生存グラフの利用

本手法では,最大干渉度低減操作において領域分割 レジスタ生存グラフを用いる.これにより,問題点1 が解決され,また問題点2を克服するアルゴリズムの 適用が可能になる.ここで,領域分割レジスタ生存グ ラフとは,以下の条件を満たすレジスタ生存グラフで あると定義する.

- (1) 各ノードは生存区間の終わり以外からのエッジ を持たない.
- (2) ガードが生成されたサイクルから次のサイクル にまたがって生存するノードが存在しない.

各ノードは生存時間の分だけ等時刻線と交わる.ま た,ガードが生成されたサイクルの直後に条件境界を 持ち,Top ノードの直後とBottom ノードの直前に もそれぞれTop ノード境界,Bottom ノード境界を 持つ.各境界はノード,境界,等時刻線とは交わらず, エッジのみと交わる.また,接続に用いる小グラフを 解要素領域と呼び,最大干渉度低減操作時には解要素 領域のパラメータを用いて作業を行う. グラフ中にお いて条件境界領域とは,同一の組の条件境界に上下を 挟まれたノードの集合である.解要素領域は,条件境 界領域中のノードで,直接,もしくは間接的にデータ 依存関係を持つノードの集合である.

このような領域分割レジスタ生存グラフの条件は, 通常満たされないことが多く,本手法では条件を満た すようにグラフトポロジを変形させている.前述の条 件(1)を満たすため,図4のレジスタr2を,また, 条件(2)を満たすため,レジスタr3,r5を2つに分 割した.図6に示すように,分割後のノードどうし には,同じ値を保持することを示すセイムエッジを張 る.分割された2つのノードは,必ず,同一解要素領 域に属するかもしくは別の条件境界領域に属する.そ のため,本手法を適用する場合,領域の接続時に複製 されたノードどうしが干渉度を不必要に高める心配は なく,付加されたノード・セイムエッジに対して特別 な考慮は必要ない.

このような領域分割レジスタ生存グラフを用いる利 点として,次の2つがあげられる.1つ目の利点は, 探索木の枝狩りによる計算コストの低減である.同一 解要素領域にはデータ依存関係を持つノードどうしが 集められるため,ある仮想レジスタの値が定義される と,多くの場合,同一解要素領域に属する他の仮想レ ジスタの依存関係が満たされ、命令が実行可能な状態 になる.つまり,同一解要素領域内の仮想レジスタを 連続して定義することで,仮想レジスタの生存区間を 短く保つことができる.このため,コードの持つ ILP に対して利用可能な実レジスタ数が不足する場合に も,十分な ILP を保つことが可能である.例として, 図 6 中の条件境界領域 G を用いると, r7, r8 が定義 されてから store 命令を実行せずにいると,両者の生 存区間が伸長し、レジスタ資源を逼迫することは自明 である.そこで,領域分割レジスタ生存グラフを用い ると, store 命令の実行を遅らせるような解の探索を せず,探索コストを低減することができる.

2つ目の利点は,論理排他性の有効活用である.最 大干渉度低減時にグラフトポロジの無制約な変更を許 せば,探索を行う解空間の範囲は大きくなり,探索コ ストの増大と引き換えに,最良の解を発見する可能性 が高まるように見える.しかし,論理排他性を持つ命 令どうしの実行サイクルがずれてしまい,レジスタを 共有しにくくなる恐れがあるため,必ずしもそうであ るとはいえない.そこで,領域分割レジスタ生存グラ フを用いればグラフトポロジの変更は条件境界領域内 にとどめられるため,明示的に論理排他性を利用しや



すい状態に置くことができる.図6を例にすると,条 件境界2中のr3,r6は,論理排他性の関係にあるノー ドと同じ条件境界2に属している.よって,必ず生存 区間が重なるため,論理排他性を利用できる.しかし, 図4を用いる文献3),4)の手法では,必ずしも論理 排他性を利用できる形でグラフ変形が行われるとは限 らない.このことから,領域分割レジスタ生存グラフ を用いることで論理排他性を有効に活用できることが 分かる.

5. 本手法の流れ

図5に全体の流れを示した.

まず,入力となる GPDG からレジスタ生存グラフ を生成し,領域分割レジスタ生存グラフの定義に適合 させる変形を加える.変形後,グラフ中から領域の抽 出とパラメータ設定の計算を行い,これを用いて DP を用いた最大干渉度低減操作を行う.この操作によっ て,グラフ中にスピルコードの挿入や命令の実行順序 の変更による仮想レジスタの生存区間の関係を保つた めの仮想エッジを GPDG に追加することで,最大干 渉度低減操作の結果が反映された GPDG を得る.そ の後,仮想レジスタを実レジスタへ割付けるとともに コードスケジューリングを行う.

また,レジスタ生存グラフ中において最大干渉度が 利用可能実レジスタ数以下であれば,実レジスタは不 足しないため,低減操作は行わない.このようにして, レジスタ割付け,およびコードスケジューリングを完 成させる.

6. サンプルへのスケジューリング

図5に従い,図1のサンプルコードに対して本手 法を適用した結果を示す.

まず,図1(ii)に対するGPDGが図3,ここから導 くレジスタ生存グラフが図4,領域分割レジスタ生存



図 6 図 4 に対する領域分割レジスタ生存グラフ

Fig. 6 Region partitioned register existence graph for Fig. 4.

解要素領域E	解要素領域F	解要素領域G	解要素領域H		EGH	٦					
width[3]=1	width[3]=2	width[3]=2	width[3]=2	1 60 1	FC CH						
width[4]=1	width[4]=0	width[4]=0	width[4]=2	IFXH	EO OR						
width[5]=1	width[5]=0	width[5]=0	width[5]=2	EC	CH /						
width[6]=1	width[6]=0	width[6]=0	width[6]=0	EG	UH.						
depth=4	depth=4	depth=4	depth=4	а Ц	E						
free=3	free=2	free=2	free=0	п	7						
pre edge=1	pre edge=1	pre edge=1	pre edge=1		<u>Ý Ý</u>						
cc=cc1	cc= !cc1	cc=cc1	cc=cc1	2段目	EGH						
(i) < 条	佐倍肉頌博 (C由	の各解亜麦領情	の情報~	(ii) <cc=< td=""><td>cc1に対するD</td><td>Pの表</td></cc=<>	cc1に対するD	Pの表					
(1) <末件境外領域C中の旨解安系領域の情報> (1) いい いいには、 (1)											
EG	GH	解要素領	镇(GH)E	解	要素領域(EG)	Н					
[ノーマル接続]	[ノーマル接続	[] [Eを2サ	イクルスピル]	(H	を1サイクルス	ピル]					
width[3]=3	width[3]=4	width	31=4		width[3]=4	1					
width[4]=1	width[4]=2	width	41=3		width[4]=3						
		width	depth	ıの値を	width[5]-3						



(iii) < (ii) の表のために必要な接続結果 >

図 7 最大干涉度低減操作

Fig. 7 Maximum interference degree reduction.

グラフが図6である.

領域を抽出後,図6の条件境界領域2に対して最 大干渉度低減操作を行った際の様子を図7に示す.

以下,Gを解要素領域とし,次のようにGの各パ ラメータを表す.

 $G_{width[i]}: i$ サイクルの干渉度

 $G_{depth}: クリティカルパス長$

 G_{free} :自由度

*G*_{pre-edge}:入力エッジの本数

 $G_{edge[i]}: i$ サイクルの入力エッジの本数

 $G_{spill-in}$:挿入されたスピルインノードのサイクル

 $G_{spill-out}$:挿入されたスピルアウトノードのサイクル

ここで解要素領域の自由度とは,該当する解要素領 域に属する全ノードに関し,解要素領域が属する条件 境界領域のクリティカルパス長に影響なく実行を遅延



図 8 変更後の GPDG

Fig. 8 GPDG after reduction of interference degree.

clock	ALU1	ALU2	レジスタ1	レジスタ2	レジスタ3	レジスタ4
1	cc1=r2&0	spill_out(r2)	r2	r3	r4	r5
2	r9=Load(r5)	r8=r4+4	empty	r3	r4	r5
3	r6=r3<<2	r7=r4>>2	(r9)	r3/r8	r4	r5
4	store(r7,r8)	spill_in(r2)	r9	r3/r8	r6/r7	r5
5	r10=r9+1	store(r6,r3)	r9	r3/(r2)	r6	r5
6	store(r10,r5)	r11=r2+1	r10	r2	empty	r5
7			empty	r11	empty	empty

図 9 スケジューリング結果 Fig.9 Result of scheduling by our method.

することができるサイクル数を意味する.

条件境界領域 2 中には,図 7 (i) のように, *E*~*H* の解要素領域が存在する.このうち,解要素領域 F は !*cc*1 というガードが付加されており,他の領域と論理 排他性を持つため, E, G, H を優先して接続する.

E,G,Hの最適な組合せを探索するため,図7(ii) の表を作成する.2段目のE,G,Hの接続は,Eと Gの接続を先に行う(EG)Hと,GとHの接続を先 に行うE(GH)の2種類の接続が考えられるため,両 者の接続結果のうち, G_{depth} が最短の接続方法を用い る.ここでは前者の接続の結果が5,後者の結果が4 となるため,後者の接続方法を採用する.その様子が, 図7(iii)に示されている.図中,括弧で括られた解要 素領域は,その接続を優先して行ったことを示す.解 要素領域の数rが増すに連れて DPの表の深さも増 すが,計算量は $O(r^2)$ 以下に抑えられるのが特徴で ある.

このようにして,最大干渉度が実レジスタ数以下に 低減され,かつクリティカルパス長が最短となる条件 境界領域が得られる.すべての条件境界領域の最大干 渉度を低減し,ここで行った実レジスタ数制約を満た すための変更点を,図8のように GPDG に依存エッ ジを加えることで反映させる.

図 8 に対して 2 つの ALU, 4 つのレジスタを持つ ハードウェアを想定してスケジューリングすると,図9 に示すように 7 サイクル必要である.文献 6)の手法 では 10 サイクル,文献 3)の手法では 8 サイクル,文 献 4)の手法では 8 サイクル必要であった.

7. アルゴリズム

7.1 領域分割レジスタ生存グラフへの変形手順 [入力:レジスタ生存グラフ]

- Top ノードの直後, Bottom ノードの直前に, それぞれ Top ノード境界と Bottom ノード境 界を設置.
- (2) 全ガードノードに対し,その値が生成された直後に条件境界を設置.
- (3) 全条件境界に対し,境界を越えて生存する全 ノードを境界を境に分割し,ノード間にセイム エッジを設置.
- (4) 生存区間の終わり以外のサイクルで値参照関係のある全ノードに対し,値参照のあるサイクルでノードを分割し,値の引継ぎを示すセイムエッジを設置.

[出力:領域分割レジスタ生存グラフ]

7.2 各パラメータの設定

[入力:領域分割レジスタ生存グラフ]

- (1) 全境界をサイクルの降順に境界リストへ追加.
- (2) 最も近い境界どうしに挟まれた全ノードを同じ
 条件境界領域 R に属するとし, i サイクルの等
 時刻線の干渉度を R_{width[i]} に設定.
- (3) 全ノードをノードリスト1に追加し、空になる まで以下を実行.
 - (a) ノードリスト1から先頭ノード n を取
 り出し,新しい解要素領域 r に属すると
 設定.
 - (b) r が, n が属する条件境界領域 R に属
 すると設定.
 - (c) nからエッジをたどって到達可能なノードのうち, Rに属するノードをノードリスト1から取り出し, rに属すると設定.ただし, Top ノード, Bottom ノードを介してはたどれず, Rに属さないノードは連続して1度しかたどれない.
 - (d) rに属するノードからなるレジスタ生存
 グラフのiサイクルの干渉度を,r_{Width[i]}
 として設定.
 - (e) rからなるレジスタ生存グラフの自由度
 を r_{free} に設定.
 - (f) Rが挟まれる境界どうしの距離から r_{free}
 を引いた値を r_{depth} に設定.

(g) *n* に付加されたガードを r_{cc} に設定.

- (4) 全解要素領域 r に対し, r_{cc} と異なるガードが 付加されたノードが存在すれば,論理排他性の ある実行条件が付加されたノードの組に対して, i サイクルのノードの組の数を r_{Width[i]} から減 算し,ガードの共通部分を r_{cc} として再設定.
- (5) 最大干渉度が利用可能な実レジスタ数を超過する解要素領域に対し,文献3)のヒューリスティクスによる最大干渉度低減操作を適用.
- (6) 解要素領域のクリティカルパスが伸びた場合,
 同一条件境界領域に属する全解要素領域 r の
 r_{depth} を同じだけ延長.
- (7) 元のグラフにおけるノードへのエッジを,その ノードが属する解要素領域へのエッジに追加.
- [出力:パラメータ抽出後の領域分割レジスタ生存グラフ]7.3 最大干渉度低減操作

[入力:領域分割レジスタ生存グラフ]

- (1) すべての条件境界領域を条件境界領域リストに追加.
- (2) 以下の作業を条件境界領域リストが空になるま で繰り返す.
 - (a) 条件境界領域リストの先頭 R を取り出し、全干渉度が実レジスタ数を超過していなければ、(2)へ移動.
 - (b) R に属する解要素領域中から論理排他性 を持つ組をガード番号が若い順に検出し, 後述する接続アルゴリズムに従って同一 ガードごとに接続したあとに,論理排他 性を持つ解要素領域どうしを接続.
 - (c) (b)の接続結果と残る解要素領域を接続.
 - (d) (c)の接続内容を改めて R として設定.
- (3) すべての条件境界領域をあわせ,領域分割レジ スタ生存グラフを再構成し,レジスタ制約が満 たす限り,スピルコードを除去.
- [出力:最大干渉度低減後の領域分割レジスタ生存グラフ]

接続アルゴリズム

[入力:解要素領域リスト]

- (1) **カレントレベルを**1に設定.
- (2) カレントレベルが R_{no} になるまで繰り返し.
 - (a) DP の表の[カレントレベル]段に[カ レントレベル+1]個の隣り合う解要素
 領域を接続する組合せを挿入.
 - (b) (a)で得られた全組合せに対し,組合せを2分割する全組合せに対し解要素領域どうしの接続を行い,クリティカルパ

ス長が最短になる接続結果をカレントグ ループの接続結果として登録.この際, すでに表中に登録済みの接続結果を用い て接続を実行.

[出力:接続後の解要素領域]

G' と G'' を接続して G を得るとした際の接続に 用いる計算式

論理排他性の利用

 $\begin{aligned} G_{width[i]} &= Max[G'_{width[i]}, G''_{width[i]}] \\ G_{depth} &= Max[G'_{depth}, G''_{depth}] \end{aligned}$

- ノーマル接続 $G_{width[i]} = G'_{width[i]} + G''_{width[i]}$ $G_{depth} = Max[G'_{depth}, G''_{depth}]$
- G' にスピルコードを挿入 スピル中のサイクル,スピルイン命令が挿入されて いるサイクル,その他のサイクルの順に G_{Width[i]} を求める計算式を示す.ただし,αとは,対象アー キテクチャにおけるスピルイン命令に必要なサイ クルコストを示すものとする.

 $\begin{aligned} G_{width[i]} &= G''_{width[i]} \\ G_{width[i]} &= G''_{width[i]} + G'_{edge[G'_{spill-out}]} \\ G_{width[i]} &= G'_{width[i]} + G''_{width[i]} \\ G_{depth} &= Max[G'_{depth} + (G'_{spill-in} - G'_{spill-out}) + \\ \alpha - G'_{free}, G''_{depth}] \end{aligned}$

8.実 験

8.1 性 能 評 価 想定するハードウェアを以下に示す .

- 2 命令・4 命令並列 VLIW アーキテクチャ.
- レジスタ数 8・32.
- キャッシュミスが起きない.
- プレディケートをサポート.
- 命令レイテンシは,整数演算を1,メモリ演算を2,浮動小数演算を5,浮動小数の乗算を7,浮動小数の除算を23,スループットをすべて1と設定.

キャッシュミスが起きないと設定しているのは,レ ジスタ割付けに対する定常的な評価を示すためである. 各命令のレイテンシは,広く使用されている Pentium4 の x86 アーキテクチャに準拠した.

ベンチマークプログラムは, Stanford-Integer Benchmark¹³⁾中の特徴的なループを用いた.また,浮 動小数点演算中心のプログラムのデータとして, Livermore Fortran Kernel(LFK)¹⁴⁾の "DISCRETE ORDINATES TRANSPORT" カーネルに対しても,

レジスタ数	レジスタ数 8								レジスタ数 32							
並列度	2				4			2				4				
手法	手法 1	手法 2	手法 3	手法 4	手法 1	手法 2	手法 3	手法 4	手法 1	手法 2	手法 3	手法 4	手法 1	手法 2	手法 3	手法 4
Bubble	38	32	24	22	38	19	16	15	27	24	24	24	25	15	15	15
Exptab	92	49	45	44	92	43	25	24	39	39	39	39	34	20	20	20
Fit	77	149	77	73	75	147	75	72	73	73	73	73	72	72	72	72
Initarr	96	83	49	45	96	55	33	33	51	41	41	41	49	33	33	33
Initmat	111	77	58	61	111	74	35	39	63	52	52	52	59	31	31	31
Permute	26	40	21	18	25	29	18	18	26	21	21	21	24	18	18	18
Qsort	35	45	35	32	35	31	25	23	29	29	29	29	24	23	23	23
Trans	153	155	135	135	153	155	135	135	136	135	135	135	136	135	135	135

表 1 クリティカルパス長の比較 Table 1 Comparisons of critical path length.

試験的に同様の実験を行った.プログラムから最内 ループを取り出し,それに対するレジスタ割付けを行 い,クリティカルパスリストスケジューリング¹⁵⁾に よるコードスケジューリングを行った.比較した手法 は,レジスタ彩色法(手法1),並列化レジスタ干渉 グラフを用いた手法(手法2),シリーズパラレル型 レジスタ生存グラフを用いた手法(手法3),そして, 本手法の4種類である.評価は,生成されたコードの クリティカルパス長によって行った.クリティカルパ スの長さはプログラムの実行時間に相当するため¹⁶⁾, この数値の比較によりプログラム速度の比較を行うこ とができる.結果を表1に示す.

手法1は,並列性を考慮していない手法であるため, プロセッサの並列度が上がっても,顕著な性能の向上 が見られない.

手法2は,コードが保持するILP に対しレジスタ 資源が潤沢でない際,つまりレジスタプレッシャが高 い場合には,並列性を保持するためのエッジが逆効果 となり,余分なスピルコードが出てしまうことがある. このため,Fit に見られるように下回る性能を示す可 能性がある.

手法3は,並列度の高い部分に優先してレジスタを 割り当てるため,手法2と比較して余分なスピルコー ドの発行を抑制しており,高いILPの抽出に成功し ている.しかし,Fitに見られるようにコードの持つ ILPが低い場合やレジスタ数32における,レジスタ プレッシャが高くない状況の実験においては,従来手 法との性能差が見られない.

手法4は、より広く解空間を探索するアルゴリズムに より、全体として従来手法以上のILPを抽出している. しかし、レジスタ数:8、実験プログラム:Initmatの 場合に見られるように、利用可能実レジスタ数と同じ くらい大きな干渉度を持つ解要素領域が、同一条件境 界領域内に複数存在する場合、手法3の性能を下回る 可能性がある.これは、解要素領域の再接続において スピルコード挿入が必要になるが、その際スピルコー ドが頻発することによってクリティカルパス長の伸長 が起こるためである.また,手法3はこういった状態 に特に強いヒューリスティックアルゴリズムを採用し ているため,このような結果が出たと思われる.この 状況は,領域分割レジスタ生存グラフから予測可能で あり,別手法に切り替えるなど,様々なアプローチで の克服が可能である.レジスタ数32における実験に おいては,従来手法との性能差が見られないが,Fit に見られるようにコードの持つILPが低い場合にも, 従来手法を上回る性能を示した.また,本手法のレジ スタ数8,32の場合における性能差が少ないないこと から,少ないレジスタ資源の再利用が並列実行を阻害 する逆依存を発生させないよう,コードスケジューラ とレジスタアロケータの統合アルゴリズムが効果的に 働いたといえる.

この実験により,従来手法が性能を発揮しやすい場 合や ILP の抽出が困難な場合にも,従来手法と同程 度の性能を発揮することができ,特にレジスタプレッ シャが高い場合に,より高い ILP を抽出できること が確認された.

8.2 計算量評価

本手法は,アルゴリズムに DP を適用することで, 計算量を抑制しつつ広い探索を試みている.そこで, DP による枝狩りの効果,領域分割レジスタ生存グラ フを用いた効果について評価を行う.計算量の比較対 象は次のとおりである.

- 手法1:レジスタ生存グラフを用いた全探索手法
- 手法2:領域分割レジスタ生存グラフを用い,解 要素領域の組合せを全探索する手法
- 手法3:本手法(論理排他性検出時)
- 手法4:本手法(論理排他性未検出時)

手法1は,探索木中のノードを設定する操作が最内 ループとなり,レジスタ生存グラフ中のノード数を *n* とすると,作成する探索木のノード数は *n*!である.ス ピルコード挿入時は,グラフトポロジの変形に際し, たかだか *n* 回の値比較を行うため,手法1の計算量 $lactriangle O(n^n)$ である.

手法 2 は,解要素領域どうしの接続が最内ループと なり,条件境界領域数を NUM_R ,ある条件境界領域 Rに属する解要素領域数を $NUM_{r[R]}$ としたときの接 続回数は, $NUM_R \cdot \sum_{i=1}^{NUM_{r[R]}} NUM_{r[R]} P_i$ と表される. また,接続の計算量は,スピルコードが挿入される場 合 $NUM_{r[R]} \cdot r_{depth}$ に比例する.これより,計算量は $NUM_R \cdot NUM_{r[R]} \cdot r_{depth} \cdot \sum_{i=1}^{NUM_{r[R]}} NUM_{r[R]} P_i$ に 比例することとなり,手法 2 の計算量は, $O(NUM_R \cdot r_{depth} \cdot NUM_{r[R]})$ であると分かる.

手法 3 に関しては,最大干渉度低減操作手順の接続 アルゴリズム内,手順(2)(c)が最内ループとなる. 論理排他性を持つ組が k 個存在した場合,k 組目の 解要素領域数 $NUM_{r[R][k]}$ は平均して $NUM_{r[R][k]} \leq \frac{2 \cdot NUM_{r[R]}}{k}$ を満たす.この場合の接続回数は,2k 回 であり,解要素領域数 NUM_r 個に対する計算量は NUM_r^2 に比例する.また,接続の計算のコストは, スピルコードを挿入する場合 $NUM_{r[R]} \cdot r_{depth}$ に比 例するため,本手法の計算量は, $O((\frac{NUM_{r[R]}}{k})^2 \cdot k \cdot NUM_{r[R]} \cdot r_{depth}) = O(\frac{r_{depth}}{k} \cdot NUM_{r[R]}^3)$ となる.

本手法に関して上記に示した計算量のデータは, 論理排他性を持つ解要素領域が含まれない場合,ゼ 口除算が発生し,意味をなさない.手法4,つまり k = 0の際,スピルコード挿入時を基準に起算すると, $O(r_{depth} \cdot NUM_{r[R]}^{3})$,となる.

手法 1 の計算量を基準として,どれほど計算コス トを低減できているかを考察する.手法 2 におい て, $NUM_R \cdot NUM_{r[R]} \le n$ がつねに成り立つため, $NUM_R \cdot r_{depth} \cdot NUM_{r[R]}^{NUM_{r[R]}+1} \le n \cdot r_{depth} \cdot$ $NUM_{r[R]}^{NUM_{r[R]}}$ である.また, $r_{depth} \ll n$ かつ $NUM_{r[R]} \ll n$ が多くの場合成り立つことは自明であ る.これらの条件から,見た目のオーダは,手法 2 が手 法1以上に高まっているものの,計算コストそのもの は大幅に低減されていると分かる.しかし, $NUM_{r[R]}$ によるとはいえ,計算コストは大きく発散しており, 現実的な計算量であるとはいえない.

次に,手法3において, $k \ge r_{depth}$ はnに依存する ため, $O(NUM_{r[R]}^3)$ が成り立つ.ここで,先ほど述 べたように, $NUM_{r[R]} \ll n$ が多くの場合成り立つた め, $O(NUM_{r[R]}^3) \doteq n^2$ であるといえる.以上から, 手法1と比較し,計算量をきわめて大きく低減してい ることが分かる.また,文献6)の手法などが,スピ ルコードが頻発した場合に $O(n^3)$ 程度の計算コスト を必要とすることなどから,手法3の計算量は,十分 許容範囲内であることが結論付けられる.

最後に,手法4において, $r_{depth} \leq n$ がつねに成

り立ち, $NUM_{r[R]} \ll n$ が多くの場合成り立つことから, $O(r_{depth} \cdot NUM_{r[R}^3) \doteq n^3$ であるといえる.以上から,手法1の計算コストと相対比較し,計算量をきわめて大きく低減していることが分かる.しかし,手法3, レジスタ干渉グラフを用いたレジスタ彩色法と比較すると,計算量は著しく大きい. n^3 という値から,動的コンパイルにおけるレジスタ割付けへの適用は困難であると予想されるが,静的コンパイルへの適用を考慮すると,十分に計算量を抑制しているといえる.

9. おわりに

本稿では,命令レベル並列プロセッサ向けにコード を最適化するレジスタ割付け手法を提案した.DPを 用いたアルゴリズムにより,計算コストを低減しつつ, 広く解空間を探索するヒューリスティクスを考案し, より高い ILP を抽出する新しいアプローチのレジス タ割付けを行った.本稿により,本手法がレジスタプ レッシャに高い場合に良い性能を発揮することが確か められた.今後,より効率的なアルゴリズムの考案を 進める予定である.

参考文献

- Norris, C. and Pollock, L.L.: A Scheduler-Sensitive Global Register Allocator, *Proc. ACM SIGPLAN '93 Conf. on Supercomputing*, pp.804–813 (1993).
- Pinterm, S.S.: Register Allocation with Instruction Scheduling: A New Approach, Proc. ACM SIGPLAN '93 Conf. on Programming Languages Design and Implementation, pp.248-257 (1993).
- 3) 古関 聰,小松秀昭,百瀬浩之,深澤良彰:命 令レベル並列アーキテクチャのためのコードス ケジューラ及びレジスタアロケータの協調技法, 情報処理学会論文誌, Vol.38, No.3, pp.584–594 (1997).
- 4) 近藤伸宏,小松秀昭,古関 聰,深澤良彰:シ リーズパラレル型レジスタ生存グラフを用いたレ ジスタ割付け技法とその評価,情報処理学会論文 誌, Vol.41, No.11, pp.3122–3132 (2000).
- 5) http://gcc.gnu.org/
- Chaitin, G.J., et al.: Register Allocation via Coloring, *Computer Languages*, Vol.6, pp.47– 57 (1981).
- 7) Cytron, R., et al.: An Efficient Method of Computing Static Single Assignment Form, Conf. Record of the 16th ACM Symposium on the Principles of Programming Languages, pp.25–35 (1989).

Vol. 43 No. SIG 6(HPS 5)

- 8) 小松秀昭, 古関 聰, 深澤良彰: 命令レベル並列 アーキテクチャのための大域的コードスケジューリ ング技法,情報処理学会論文誌, Vol.6, pp.1149– 1161 (1996).
- 9) Ferrante, J., Ottenstein, K.J. and Warren, J.D.: The Program Dependence Graph and Its Use in Optimization, ACM Trans. Prog. Lang. Syst., Vol.9, No.3, pp.319–349 (1987).
- Duffin, R.J.: Topology of Series-Parallel Networks, J. Math. Applic. 10, pp.303–318 (1965).
- Norris, C., et al.: An Experimental Study of Several Cooperative Register Allocation and Instruction Scheduling Strategies, *Proc. 28th International Symposium on MICRO* (Nov. 1995).
- 12) Berson, D.A., et al.: Resource Spackling: A Framework for Integrating Register Allocation in Local and Global Schedulers, *PACT* (Aug. 1994).
- Hennessy, J., et al.: Stanford Integer Benchmarks, Stanford Univ. Personal Communication (1982).
- 14) http://www.llnl.gov/asci_benchmarks/asci/ limited/lfk/
- Muchnick, S.S.: Advanced Compiler Design Implementation, pp.535–543, Morgan Kaufmann Publichers, Inc. (1997).
- 16) Fernandez, E.B. and Bussel, B.: Bounds the Number of Processors and Time for Multiprocessor Optimal Schedules, *IEEE Trans. Comput.*, Vol.C22, No.8, pp.745–751 (1973).

(平成 14 年 1 月 29 日受付)(平成 14 年 5 月 4 日採録)



淺原 英雄

1977年生.2000年早稲田大学理 工学部情報学科卒業.2002年同大 学大学院理工学研究科修士課程修了. 同年,キヤノン(株)入社.現在,同 社にてソフトウェア開発に従事. 600

近藤 伸宏

1974年生.1997年早稲田大学理 工学部情報学科卒業.1999年同大学 大学院理工学研究科修士課程修了. 同年(株)東芝入社.現在(株)東 芝セミコンダクター社システムLSI

事業部所属.



古関 聰(正会員)

1969年生.1994年早稲田大学大 学院理工学研究科電気工学専攻修士 課程修了.1998年同大学院理工学研 究科電気工学専攻博士課程修了.同 年日本 IBM(株)入社.以来,同社

東京基礎研究所において, Java just-in-time コンパイ ラの開発に従事.工学博士.



小松 秀昭(正会員) 1960年生.1985年早稲田大学大 学院理工学研究科電気工学専攻修了. 同年,日本 IBM(株)東京基礎研 究所入社.コンパイラ,アーキテク チャ,並列処理の研究に従事.博士

(情報科学).



深澤 良彰(正会員)

1976年早稲田大学理工学部電気工 学科卒業.1983年同大学大学院博士 課程中退.同年相模工業大学工学部 情報工学科専任講師.1987念早稲田 大学理工学部助教授.1992年同教授.

工学博士 . ソフトウェア工学 , コンピュータアーキテ クチャ等の研究に従事 . ソフトウェア科学会 , IEEE , ACM 各会員 .