スライドウインドウを考慮したレジスタ割付

7B－10

秡川友宏　添野元秀
山下義行　中田育男

筑波大学

1. はじめに

超並列計算機 CP-PACS は，1997 年 1 月時点で “TOP 500 List” に掲載されている計算機では最速を誇る．CP-PACS の高速性はスライドウインドウアーキテクチャ，ハイパークロスバネットワークといった機構によって支えられている．

CP-PACS のマシンクロックでは，メモリアクセスのレーテンシは実に数十 MC 以上にもなる．メモリアクセスのレーテンシを回避する方法としては，ソフトウェアパイプライニングによる先行ロード（**preload** という）が有効である．これを自然に実現するアーキテクチャがスライドウインドウアーキテクチャである[1]．

スライドウインドウアーキテクチャにおける疑似ベクトル処理には，ループ立ち上げ間隔（*II*: Initiation Interval）の算出や命令スケジューリングがごく自然な発想に基づいて素直に実現できるという利点がある．その反面，レジスタ割付のアルゴリズムはレジスタ番号の変化のため従来のものより複雑になると考えられてきた[2]が，筆者らは Spiral Graph と Short Bridge Algorithm からなる自然なフレームワークを提案し，その有効性を実験によって確認した[3]．

現在 CP-PACS 上で稼働中のレジスタ割付器は，従来のレジスタ干渉グラフをスライドウインドウ向けに拡張した有向干渉グラフ（*RID*: Register Interference Digraph）であるが，Spiral Graph ははじめからスライドウインドウを考慮して構成されたフレームワークである．今回，両者の性能比較を数種のベンチマークを用いて行なったので報告する．

2. 基本概念

プロセッサがますます高速になっている現在，メモリは相対的に低速になっている．これは 図1-左 でプロセッサ — メモリ間の距離が遠のき，バケツ (レジスタ) にデータが入ってくるまでの時間が長くなることに相当する．

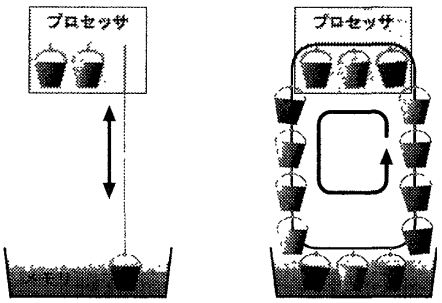

図1　概念図 — 従来のアーキテクチャ（左）と スライドウインドウアーキテクチャ（右）

メモリの低速性を隠蔽するには，**load** 命令を前以って発行（**preload** という）しておけばよい．つまり，演算を行ないつつ，先々必要になるデータに次々と予約をかけていくのである．このような手法は，一般にはソフトウェア・パイプライニングとよばれる．

スライドウインドウアーキテクチャは，このソフトウェア・パイプライニングを効率的に行なうための仕掛として提案された．図1-右 はその概略図である．メモリからデータを引き上げるには時間を要するが，スライドウインドウアーキテクチャでは，次の演算に必要なデータはすぐ隣のバケツに入っている．次の演算を行なうには，観覧車状に連なったバケツ全体を横にスライドさせるだけでよい．このとき，同時に下方のバケツで先々必要になるデータの汲み取りを行なっておくわけである．

3. Spiral Graph によるフレームワーク

ここでは，Spiral Graph によるスライドレジスタ割付のフレームワークについて簡単に説明する．

3.1 表現法 — Spiral Graph

Spiral Graph（*SG*）は横軸にループ内の命令ステップ番号，縦軸にレジスタ番号をとったグラフを用いて表す．横軸がスライド段数の分だけ傾いているのが特徴である．

スライド段数が１のときは１トラックの *SG*になる

Spiral Graph — A Register Allocation Framework for Slide-Window Architecture
Tomohiro HARAIKAWA, Motohide Soeno, Yoshiyuki YAMASHITA, Ikuo NAKATA
University of Tsukuba

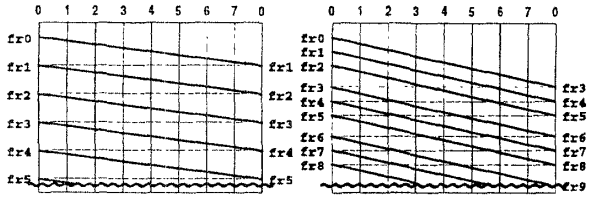

図2 Spiral Graph

（図2-左）．グラフの傾きは，ループを1周するとループ中のどこかにあるslide命令によってレジスタ番号が1つずれる様子を表している．同様に，スライド段数が3のときは3重らせん構造をもつ3トラックのSGとなる（図2-右）．

3.2 割付法 — Short Bridge Algorithm

まず，いくつかの用語を定義する．

開始・終了位置 $v_i = [s_i, e_i)$ において，v_i の始点 s_i をグラフの横幅 II で割った剰余 $s_i \bmod II$ を v_i の**開始位置**という．終点 e_i に対応する**終了位置**も同様に定める．

変数間の遠近 v_i と v_j において，$(s_j - e_i) \bmod II$ を v_i から v_j までの**隙間**という．隙間は非対称であることを注意しておく．隙間が小さければ v_i から v_j までは**近い**といい，大きければ**遠い**という．

簡単のため，まず1トラックのSGから説明する．

アルゴリズム 1 (1-SHORTBRIDGE)

(1) 変数 $v_1, \ldots, v_N$ の中から開始位置のもっとも小さいものを選び，トラック T_1 に巻き付ける．

(2) 残る変数の中から T_1 上に最後に巻き付けられた変数に最も近い変数を選び，続けて巻き付ける．

(3) (2) を，巻き付けていない変数がなくなるまで繰り返す．∎

1-SHORTBRIDGE では，すでに巻き付けられている変数から次に巻き付けるべき変数までの隙間が最小になるように次の変数を選ぶ．これは，すでに巻き付けた変数の終点と次に巻き付ける変数の始点をなるべく短い橋で短絡していくことに相当する（図3-左）．

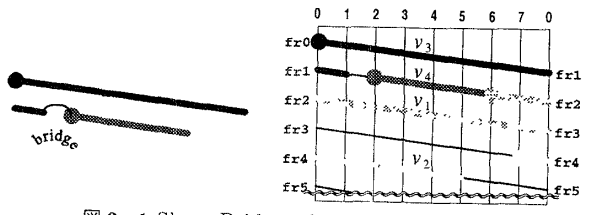

図3 1-Short Bridge Algorithm による割付

変数間の隙間が小さくなるように割り付ければ，全体としても隙間の和は小さくなると考えられる（図3-右）．

これを K トラックに拡張したアルゴリズムを示す．

アルゴリズム 2 (K-SHORTBRIDGE)

(1) 変数 $v_1, \ldots, v_N$ の中から，開始位置の小さい順に K 個の変数を選び，それぞれトラック T_k $(1 \le k \le K)$ に巻き付ける．

(2) 残る変数の中から，T_1 上の最後の変数に最も近い変数 v_1' を選択し，v_1' を T_1 に巻き付けたと仮定する．その状態で $T_1 \sim T_K$ の中から最も長いトラックを探して長さを求め，Max_1 とする．求めたら v_1' を選択前の状態に戻す．

(3) 同様に，$\mathrm{Max}_2, \ldots, \mathrm{Max}_K$ も求める．

(4) Max_k を最小にする k を求め，トラック k に対して実際の巻き付け操作を行なう．

(5) (2) ～ (4) を，巻き付けていない変数がなくなるまで繰り返す．∎

4. 評価とまとめ

SGの性能を評価するため，CP-PACS の最適化コンパイラで Linpack ベンチマークと Livermore Fortran Kernel（LFK）をコンパイルし，パイプライン化に成功したループについてSBAを適用して使用スライドレジスタ数を調べた．比較対象は現在 CP-PACS の最適化コンパイラで使用されているRIDによる割付結果である．なお，LFKはカーネル部 (ker) とソース全体 (all) の双方を示す．

	Linpack	LFK(ker)	LFK(all)
改善	3 (27.2%)	10 (41.7%)	18 (32.7%)
同数	8 (73.8%)	12 (50.0%)	35 (63.6%)
増加	0 (0.0%)	2 (8.3%)	2 (3.7%)
計	11 [loops]	24 [loops]	55 [loops]

SGとSBAからなるフレームワークは，多くのケースでは，必要スライドレジスタを従来と同数あるいはそれより少なくできることがわかった．今後，レジスタ数が増加してしまった2つの不良事例への対応を考えたい．

参 考 文 献

1) H. Nakamura, H. Imori, K. Nakazawa, T. Boku, I. Nakata, Y. Yamashita, H. Wada & Y. Inagami (1993), "A Scaler Architecture for Pseudo Vector Processing based on Slide Windowed Registers", *ACM Proc. Intl. Conf. on Supercomputing* '93, 298–397.

2) 中田育男，山下義行，小柳義夫 (1996), "超並列計算機 **CP-PACS** のソフトウェア", 情報処理 37-1, 29–37.

3) 秋川友宏，添野元秀，山下義行，中田育男 (1996), "スライドウインドウを考慮したレジスタ割付", 日本ソフトウェア科学会 第13回大会論文集, 201-204.