# 改良されたメッセージ通信機構を使用した並列化コンパイル技法

4T－8

古川 浩史　　平木 敬

東京大学 大学院理学系研究科 情報科学専攻

## 1 はじめに

最近のメッセージ通信型並列計算機は通信機構へのアクセスをユーザレベルに解放しており、短いメッセージならば非常に低コスト(数命令〜十数命令)で送受信が可能である[1]。また、これらの計算機上でアクティブメッセージ[2]等を用いたリモートアクセス機構により、比較的単純なメモリアクセスパターンをもつ共有メモリプログラムを効率良く実行できることが知られている。しかし、SPLASH[4]などに見られる実際の共有メモリアプリケーションでは、共有変数を用いてプロセッサ間の先行関係を保証することが多く、従来の機構では高性能を達成するのは難しい。

本稿では、共有メモリアクセスが可能なように拡張されたC言語のメッセージ通信型計算機への基本的なコンパイル法を述べ、また従来のリモートアクセス機構の問題点および解決法について述べる。

## 2 対象言語

本稿でコンパイルの対象とする言語は、他のプロセッサのメモリを共有メモリとみなして直接アクセスが可能なようにC言語を拡張した言語である。ユーザは記憶クラス指定子globalを用いて宣言されたデータに対して共有メモリ空間に存在するデータを宣言することができる。コンパイラは共有メモリ空間へのアクセスに対してリモートメモリアクセスを行なうコードを生成する。

## 3 基本的な変換

ある基本ブロックBに関して、リモートリードアクセスの集合GR(B)とリモートライトアクセスの集合GW(B)を考える。GR(B)およびGW(B)のすべての要素がただ一つの同一のプロセッサに対するリモートアクセスであることが静的にわかれば、その基本ブロック全体をアクティブメッセージのハンドラとする。このように変換することにより、複数のリモートメモリアクセスをただ一つのアクティブメッセージの送信に置き換えることができ、メッセージ通信のオーバーヘッドを削減することができる。

以下の例を考える。配列aは各プロセッサにcyclicに割り当てられているとする。

```
struct foo {
  int x, y, z;
} global a[10000];

bar(x, n)
{
  struct foo * global p;
  p = &a[x];
  p->x = p->z + 1;
  p->y = 0;
  p->z = n;
}
```

ここでGR={p->z}, GW={p->x, p->y, p->z}であるが、p = &であるので、GW, GRはすべて同一のプロセッサ(x % NPROC)へのアクセスであり、

```
handler(p, n)
struct foo *p;
{
  p->x = p->z + 1;
  p->y = 0;
  p->z = n;
}

bar(x)
{
  int proc = x % NPROC;
  struct foo *p = &a[x / NPROC];
  send_am(proc, handler, p, n);
}
```

と変換される。ここでNPROCはプロセッサの数であり send_am(proc, handler, args, ...)はhandlerで示されるハンドラを持つアクティブメッセージをプロセッサprocに送信する関数である。args以下はハンドラに対する引数である。

## 4 従来のリモートアクセス機構の問題点

メッセージ通信機構を用いて共有メモリをエミュレートする場合、リモートアクセスのレイテンシが非常に大きくなるために、ノンブロッキングなリモートアクセスが必要である。しかし、従来のリモートアクセス機構ではリモートアクセスの完了をフラグやアクノリッジのカウンタを用いて検出しており、リモートアクセスのorderingを保証するため、先行したリモートアクセスが完了するまでビジーウエイトを行なっていた。これはweak consistencyモデルを実現していることに相当するが、図1のようにweak consistencyモデルではsyncが完了するまで後続のメモリアクセスが発行できず、リモートアクセスのレイテンシを隠蔽することが出来ない。これは共有変数を用いてプロセッサ間の先行関係の保証を行なう時に問題となる。なぜなら、共有変数へのアクセスはsyncアクセスでなければならないが、syncは直前までに発行したすべてのリモートアクセスが完了するまで発行できず、またsync自身が完了するまで後続のメモリアクセスが発行できないため、リモートアクセスレイテンシが大きい場合には著しい性能低下の原因となる。

## 5 Release Consistency モデルの実装

Release consistencyモデル[3]ではreleaseに後続するacquireや通常のメモリアクセスはreleaseの完了を待たずに発行

---

Compilation Techniques for Improved Messaging Facilities

FURUKAWA Hiroshi and HIRAKI Kei

Department of Information Science, Faculty of Science, the University of Tokyo

{furukawa,hiraki}@is.s.u-tokyo.ac.jp

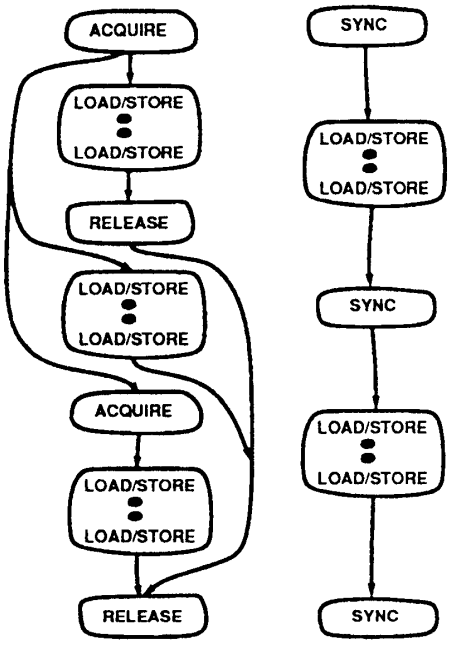


図 1: Release and Weak Consistency

が可能であり、リモートアクセスのレイテンシを隠蔽する機会が大幅に増える。Release 自身の発行は直前までに発行したリモートアクセスの完了を待たねばならないが、アクノリッジの到着をトリガとして発行するようにキューイングしておけばよい。Weak consistency モデルにおいてもキューイングの技法が使えるが、sync 以降のリモートアクセスをリードアクセスを含めすべてキューイングしなければならず、性能向上は期待できない。

以下で、release consistency モデルを実装するリモートアクセスプリミティブの実装について述べる。

各プロセッサは世代カウンタを持ち、それぞれのリモートメモリアクセスを、それを発行した時点での世代カウンタの値で世代づけする。また、各プロセッサは各世代に対応した発行カウンタと回収カウンタを持ち、リモートアクセスの発行、アクノリッジの回収時にそれぞれのカウンタをインクリメントする。二つのカウンタの値を比較することにより、プロセッサはそれぞれの世代についてすべてのアクセスが完了したか否かを知ることが出来る。

プロセッサが release を発行すると、まず、メモリアクセスの世代カウンタをインクリメントし、新しい世代の発行カウンタと回収カウンタをクリアする。次に、前の世代のメモリアクセスがすべて完了したか否かをチェックする。すべてのリモートアクセスが完了していれば、release すべきアドレスに対してリモートストアを新しい世代で発行する。まだ完了していないリモートアクセスがあれば、release すべきリモートアクセスリクエストを新しい世代で世代づけし、キューに入れると同時にリモートアクセスの発行カウンタをインクリメントする。

プロセッサはアクノリッジを受けとるとハンドラルーチンにディスパッチする。ある世代のリモートアクセスのアクノリッジをすべて受けとると、ハンドラはキューからリモートアクセスリクエストを (存在すれば) 取り出し、ネットワークに対して発行する。このとき発行カウンタはキューに入れた時点でインクリメントしているために実際にリクエストを発行する際にはカウンタのインクリメントは行なわない。

通常のリモートアクセスを発行する場合には、単に発行カウンタをインクリメントしてリクエストメッセージをネットワークに対して送出すればよい。

acquire アクセスは単純にアクノリッジが帰ってくるまでビジーウエイトする。

## 6 不可分操作

ロックで排他制御を行なう領域について考える。Release consistency モデルでは、ロックの獲得は acquire アクセス、ロックの解放は release アクセスで行ない、排他的に実行される領域内のリモートアクセスは通常の load/store アクセスで行なう。しかし、acquire アクセスは blocking なリモートアクセスであるため、アクセスレイテンシの隠蔽は不可能であり、また release によるロックの解放も、実際にネットワークにメッセージを送信するのは領域内のリモートアクセスの完了を待たねばならず、必要以上に長い間領域がロックされる可能性がある。実際のアプリケーションにおいては、ロックの獲得／解放のような低レベルな操作はカウンタの排他的更新やキューイング／デキューイングなどより抽象的なメモリの不可分操作を実現するために用いられていることが多く、これらの場合には高度な操作を直接実装することが出来れば、送信すべきメッセージは一つで済み、また送信側のプロセッサは領域内のメモリアクセスの完了を待たずに処理を続行することが出来るため、大幅な性能向上を達成することが可能である。実際には、ロックへのアクセスを含めて領域内のリモートアクセスがすべて同一のプロセッサに対するものであれば、基本ブロックの変換と同様に領域全体をアクティブメッセージのハンドラとして実現することができる。ハンドラ内では、別のメッセージが到着した場合に preemption を起こさないとすれば領域内の実行の排他性も保証される。

## 7 おわりに

本稿では、共有メモリアクセスが可能なように拡張された C 言語をメッセージ通信型計算機へとコンパイルする方法について述べた。またメッセージ通信型計算機での共有メモリの従来の実装法の問題点を指摘し、解決法について述べた。今後は、以上に述べた技法に基づくコンパイラを実装する予定である。

## 参考文献

[1] T. Horie, et al. Improving AP1000 Parallel Computer Performance with Message Communication. *Proc. 20th Int. Symp. on Computer Architecture*, pp.314-325, 1993

[2] T. von Eicken, et al. Active Messages: a Mechanism for Integrated Communication and Computation. *Proc. 19th Int. Symp. on Computer Architecture*, pp.256-266, 1992

[3] K. Gharachorloo, et al. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. *Proc. 17th Int. Symp. on Computer Architecture*, pp.15-26, 1990

[4] J.P. Singh, et al. SPLASH: Stanford Parallel Applications for Shared memory. Techincal Report CSL-TR-92-205, Stanford University, 1991