計算機アーキテクチャ 136－9
（２０００．１．７）

コンパイラ制御可能なCOMA環境を実現するソフトウェアDSM Laurasiaの実装

上原哲太郎† 齋藤 彰一‡ 城 和貴§ 國枝 義敏‡

†和歌山大学システム情報学センター ‡和歌山大学システム工学部 §奈良女子大学理学部

概 要

自動並列化コンパイラの作成に際し、並列計算機アーキテクチャのノード間通信機構の差異を、コンパイラから制御可能なCOMA(Cache Only Memory Architecture)環境の実現によって隠蔽する方式を提案している。本論文では、同方式によるコンパイラをUNIXベースのワークステーション(WS)クラスタ上に実現するのために作成したソフトウェアDSM(分散共有メモリ)の実現について述べる。共有メモリの弱い整合性制御モデルとしてEntry Consistencyの実現機構を備える。100BASE-TX使用時、4～6MB/sのページ転送性能を達成した。

The implementation of Laurasia : a software DSM system to realize compiler-controlled COMA environment.

Tetsutaro UEHARA† Shoichi SAITO‡ Kazuki JOE§ Yoshitoshi KUNIEDA‡

† Center for Information Science, Wakayama University, Japan.
‡ Faculty of Systems Engineering, Wakayama University, Japan.
§Faculty of Science, Nara Women's University, Japan.

Abstract

We have proposed an approach to bridge a gap among communication mechanisms of various parallel computer architectures by providing COMA (Cache-Only Memory Architecture) environment that can be controlled by automatic parallelizing compilers. This paper presents the implementation of a software DSM (Distributed Shared Memory) system on clusters of workstations running UNIX, which is developed for a compiler based on the approach. The system manages shared memory based on Entry consistency model. The system achieves the transfer rate of approximately 4 to 6MB/s for the shared pages when the network is based on 100BASE-TX.

1 背景

近年、高性能計算機の多くが並列計算機となり、ハードウェアとして広く普及するに至った。また、ワークステーション(WS)やパソコン(PC)はLANに接続されているのが常となり、LANで多数結合されたWS,PC(WS/PCクラスタ)も並列計算機と見なせるので、今や並列計算機はあらゆる所にあると言える。その反面、並列プログラミングはまだ普及途上にあり、多くのプログラマは未だ逐次型プログラミング言語か、せいぜい共有メモリ型並列計算機上でのマルチスレッドプログラミングを行っている程度である。これまで、主記憶分散並列計算機でのプログラミングのためPVM、MPIやHPF[1]といった標準ライブラリ・言語が提供されてはきたが、これらを利用するのは一部のプログラマに限られている。特に科学技術演算の分野におけるソフトウェア資産は未だ多くが逐次型計算機用ないしベクトル型スーパコンピュータ向けに記述されたものであり、これらを分散主記憶型並列計算機に移植する、あるいはアルゴリズムから見直して書き直す作業には、大きな労力が必要である。

この作業の負担を軽減し、並列計算機の可用性を一気に高めることができるのが、自動並列化コンパイラである。既にベクトル型計算機や主記憶共有型並列計算機においてはこのようなコンパイラは多く実用に供されている。しかし、主記憶分散型の並列計算機やWS/PCクラスタにおいては、コンパイラは並列化による速度向上がノード間通信による処理速度の低下を常に上回るよう、タスクとデータの分割方法およびノード間での配置を決定せねばならない。これは一般には非常に困難であるため、これらの計算機用に自動並列化コンパイラを提供することは困難である。HPFなどはこの問題を回避するためデータの分割と配置はプログラマに指定させ、コンパイラは並列化に専念する手法をとっているが、プログラマの負担増は避けがたく、普及の障害となっている。

このような問題へのひとつの解決策として、また多様な並列計算機アーキテクチャに対し並列化コンパイラの研究を行うフレームワークとして、我々はcc-COMAと呼ぶ計算機アーキテクチャモデルを提唱し、その上で自動並列化コンパイラの開発を行うアプローチを提案した[2]。本稿では、このcc-COMAについて述べ、さらにこのcc-COMAモデルのコンパイラ作成のテストベッドとして構築しているWSクラスタ用ソフトウェアDSMシステムLaurasiaの実装について述べる。

2 cc-COMAランタイムモデル

COMA(Cache Only Memory Architecture)とは、分散共有メモリ(DSM : Distributed Shared Memory)型並列計算機アーキテクチャの一種である。このアーキテクチャでは、全プロセッサが1つのメモリアドレス空間を共有し、各ノード上のローカルメモリはその共有メモリのキャッシュとして動作する。主記憶が各ノードのメモリにどのように割り当てられるかは動的に変化する(すなわち、各アドレスのメモリにとってホームという概念はない)。あるノードにおいてローカルメモリ(キャッシュ)に存在しないメモリ空間への参照が発生した場合には、そのメモリ空間を保持するほかのノードが探し出され、そこからデータが転送される。また、各ノードのメモリはキャッシュであるため、あるメモリ空間の内容が複数のノードに複製されて保持されることがあり、この場合にはそのデータ内容の整合性制御が必要である。

cc-COMA(compiler-controlled COMA)の基本的なアイデアは、分散主記憶型並列計算機を対象として、COMA環境をソフトウェアで実現し、その制御をコンパイラにゆだねようとするものである。このような環境では、タスクを分割し各ノードへ配置すれば、データは実行時に自動的に各ノードに分割配置されてゆく。この際、キャッシュの効果により各データは最もよく参照されるノードに配置されるので、通信は最小化されることが期待できる。これにより、プログラマは(HPF等と違って)データ分割や配置を指定せずにすむ。

しかし、ここで単純なCOMA環境を実装すると、ノード間で共有されるメモリ領域の整合性制御のためデータ転送が頻発し性能が低下するおそれがある。そのため、より弱い整合性(Weak Consistency)制御モデルとして、Entry Consistency[3]モデル(ECモデル)を導入する。ECモデルでは、全ての共有データはプログラム中で明示的に宣言され、それぞれの参照に対し排他制御を行うための同期変数と関連づけられる。共有データへのアクセスが発生するときには、acquireとよぶプリミティブを使ってシステ

ムを呼び出し、対応する同期変数に読み込みロック(アクセスが読み込みのみの時)または排他ロック(アクセスが書き込みを含むとき)をかけ、共有データのうち最新のものがそのノードから参照可能であることを保証する。参照が終わったら、release とよぶプリミティブを使って同期変数を解放するとともに、共有データの書き込みが発生した時はその結果を他のノードに反映させる。

このような実行環境を前提として自動並列化コンパイラを作成することの利点は、以下のものが挙げられる。

- (WS/PC クラスタを含む)分散主記憶並列計算機を実行時環境によって仮想的に主記憶共有型並列計算機として扱えるので、自動並列化コンパイラを作成する際ベクトル計算機やSMP/NUMA 型主記憶共有並列計算機で培われたタスク分割技術が容易に導入できる。
- ノード間のデータ通信を、共有メモリへの参照や定義に置き換えて扱うことができる。これにより、アーキテクチャによって異なるノード間通信インターフェースを単純なメモリ参照に統一して扱える。また、EC モデルの導入により、acquire/release プリミティブによってデータ通信のタイミングが制御でき、通信そのものもブロック化できるためオーバヘッドの削減が可能なる。
- NUMA でなく COMA をエミュレートする[1]ことによって、ノード間のデータの動的再配置は自然に行われる。参照に必要となる通信量も自動的に削減される。複数ノードから参照されるページに関する通信量もキャッシュ間の複製によって削減される。これの通信量は、タスクがノード間を移動しないという仮定の下では自動的に最小になる。

cc-COMA モデル下での自動並列化コンパイラの対象となるアプリケーションとしては、まず主記憶参照が規則的で、かつ十分な並列度が得られるような、大規模密行列上の科学技術計算などを考えている。このようなアプリケーションではプログラム中の主記憶参照が静的に予想しやすいことを利用して、コンパイラは並列化によるデータ転送量の増加とそのコストを静的に予測し、適切な粒度でタスク分割を行うことを目指す。データ配置そのものは実行時環境によって自然に適切になるはずであるが、静的に最適配置が求まる場合にはコンパイラがあらかじめ各ページをノード間で転送・複写しておくためのプリミティブを挿入する。このように、コンパイラが実行時環境と密接な連携を取りながら処理を進めるのが cc-COMA の特徴である。

3 Laurasia の実装

上記で述べた cc-COMA モデルに基づくコンパイラの実行時環境として、LAN で結合されたUNIX ワークステーションのクラスタを対象としたソフトウェア分散共有メモリ(DSM)システム、Laurasia を実装した。

Laurasia が要求するのは、以下のようなごく一般的なワークステーションクラスタ環境である。

- TCP/IP 環境で結合された、パケットロスの少ない高速な LAN(UDP/IP を使用するため)
- UNIX が動作するワークステーション(スレッドには POSIX thread が利用可能なもの)

現在の実装では、UNIX カーネルにも特殊な機能は必要とせず、通常の LAN アダプタとドライバで動作している。

Laurasia の構造を、図 1 に示す。ユーザプログラムはコンパイラによって共有メモリ型並列計算機上の並列プログラムに変換される。実行時は、各ノードに UNIX の 1 プロセスを割り当て、それぞれ 1 ないし複数のユーザスレッドを起動して実行する。このほかに、キャッシュの管理およびノード間の通信を行うためのスレッドが同じプロセスで生成される。これらスレッドは全てPOSIX threads(pthreads)インターフェースで生成管理されている。通信に関しては TCP/IP ではなく UDP/IP を利用している。これは、Laurasia が対象とするのが LAN で接続されたPC/WS であるため転送データはごく低い確立でしか失われないと仮定して、安全性より性能をとったためである。

Laurasia システムそのものはライブラリとして実現されている。これらのユーザプロセスからの

[1] ここで NUMA をエミュレートした方が実装は簡単になる。しかし COMA をあえて採用したもう一つの理由は、我々が対象とする分散主記憶型並列計算機では、一般にあるページがどのノードにあるか検索するコストがデータ転送自体のコストに比べて十分小さいであろうという予想に基づいている。

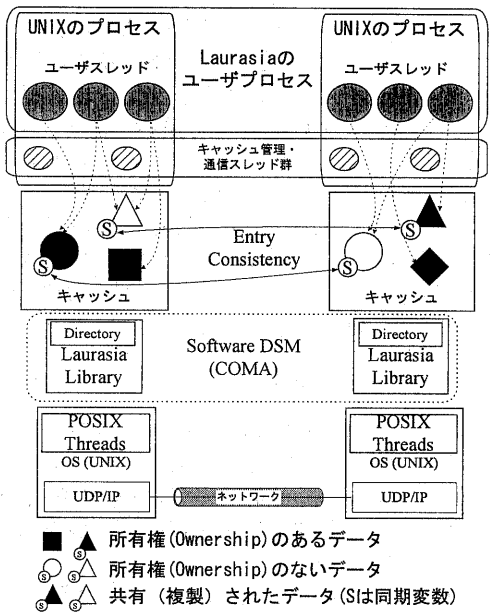


図 1 Laurasia の構造

要求に応じて共有メモリを確保、解放、転送、複製する COMA 関連のシステムコールと、EC モデルでのメモリ整合性制御を実現するための同期変数の実現用システムコール、あとプログラミングのためスレッド間のバリア同期などのシステムコールを備えている。これらのシステムコールはいずれもコンパイラが自動的にプログラム中に埋め込んで用いる。

なお、Laurasia は、UNIX そのものには変更を加えないため、他のノードにコンテキストを保存したまま新たなスレッドを起動したり、ノード間でスレッド移送したりする機能は備えていない。しかし、cc-COMA のモデルではスレッド生成や移送もコンパイラが生成したコード内で行われ、外部からのスレッド移送要求などは発生しない。このため、あらかじめスレッド移送と同等の効果があるコードを生成してプログラム中に埋め込むことで実現できる。例えばスレッドを移送する際は、各ノードのプロセスが同期をとりつつ、送り元のスレッドを消すと同時に送り先でスレッドを生成し、必要なデータは共有メモリを通じて授受させるようコンパイラにコード生成させる。

この他、Laurasia の実装技術の多くは他のソフトウェア DSM で使われているものとかわらない。以下にいくつかの実装について述べる。

3. 1 キャッシュの管理

cc-COMA モデルでは本来キャッシュは共有変数という不定長・不定形のオブジェクト単位で管理されるが、Laurasia では実装を容易にするため、固定長ページ単位で管理している。

キャッシュ内の各ページは Home, Exclusive, MasterShared, Shared の 4 つの状態を持つ。各ページはそれぞれホームノードが決まっており、Home はそのページがホームノードにあり、かつ他のノードにはそのページの複製がないことを示している。Exclusive は、Home と同様だが但しそのページがホームノード以外にあることを示す。他のノードがそのページの読み込みを行うと、Home または Exclusive を持っていたノードではそのページは MasterShared に移行し、読み込みを行ったノードでは転送されてきたページに Shared という状態が与えられる。

ホームノードの役割は、そのページの実体およびコピーがどのノードに存在しているか把握する、すなわちキャッシュディレクトリを管理することである。あるスレッドが、そのノード内のキャッシュに存在しないページを読み出そうとする際は、まずそのページのホームノードに、データを持つノードを要求する。ホームノードは、そのページが自己ノードにあり、Home または MasterShared 状態であればそのページを即座に転送する。そうでなければ、キャッシュディレクトリを見て当該ページを Exclusive または MasterShared 状態で持つノードを探し出し、そのノードに要求ノードへページを転送するよう依頼する。いずれの場合も要求ノードが Shared 状態でそのページを持つことになるので、ホームノードはそれをキャッシュおよびディレクトリに記録する。

書き込みは、Home または Exclusive 状態のページにのみ許されている。そうでないページへの書き込みが必要となる際には、当該ノードはホームノードへ書き込み権限を要求する。ホームノードはそのページの複製を持つ他のノード上のページを無効にした後、要求があったノードが当該ページを Exclusive 状態で持つことをディレクトリに記録して、書き込み許可を出す。すなわち、Write-invalidate 方式で整合性を制御している。

3. 2 ロックとバリア

同期変数のロックも、キャッシュと同様に変数ごとにホームノードが決定され、そのノードで状

態が管理される。ロックには READ_WRITE モード(読み書き可能権の取得または排他的ロック)と READ モード(読み込み可能権の取得または共有ロック)の 2 つのモードがあるが、ホームノードはキャッシュのページ管理と同様の手法で READ_WRITE 権を持つスレッドが存在するノードを常に管理している。ロックを要求したスレッドが存在するノードに READ_WRITE 権を持つスレッドがなければ、要求はホームノードに転送され、ホームノードもしくは READ_WRITE 権を持っているノードがその要求に応答する。ページ管理との違いは、所有者の単位がノードではなくスレッドである点にある。

これに対しバリアは、コーディネータノードと呼ぶ特定のノードで一括管理される。バリア要求まず各ノード内で処理され、そのノード内の全スレッドがバリアに到達するとコーディネータノードに通知される。コーディネータノードは全ノードからの通知が揃うのを待って、各ノードに継続許可を通知する。

3.3 EC モデルと cc-COMA

多くのソフトウェア DSM システムでは、ページフォルトなどをきっかけにしてキャッシュ制御機構が動作し、整合性を保つ。ところが、Laurasia は、EC モデルに基づく整合性制御用のコードを全てコンパイラが生成してユーザプロセスに埋め込む必要がある。つまり共有変数の参照に際し、acquire と release プリミティブでの宣言、具体的には同期変数へのロックを行う。Laurasia ではこれを支援するため、キャッシュのページと同期変数の関連を宣言することによってシステム側で記憶しておくことができる。この機構により、システムコールによって、あるページに関連づけられた同期変数、またはある同期変数に関連づけられたページ群を得ることができる。それと同時に、キャッシュ制御スレッドは他のノードからのページの複製要求に対し、当該ページが READ_WRITE モードでロックされている場合には同期変数が解放されるまで転送を遅延させ、誤ったデータの転送を防ぐ。

EC モデルに基づく一連の共有変数への参照は次のようになる。

- Acquire 発行。共有変数に対応する同期変数に READ_WRITE(書き込み時)または READ(読み込み時)ロックをかける。
- 共有変数のうち自ノードで invalid なページの読み込み要求を発行する。
- クリティカルセクションの実行
- 共有変数の各ページのうち Shared または MasterShared を状態に持つものは書きかわった可能性があるので、Exclusive(または Home)にして他ノードのものは無効化する。
- release 発行。同期変数を解放する。

整合性制御はクリティカルセクションの前後に集中しているのでノード間のメッセージが効率よく転送される。また、共有変数の更新を遅延させているため、他ノードがこの間旧データで実行を続行でき、実行効率が高まる。

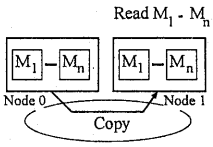
4 評価

Laurasia の基本的なデータ転送性能を、次のような実験環境で評価した。

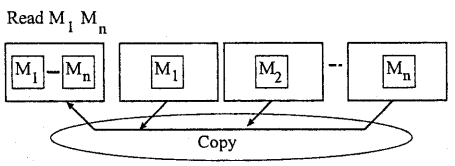
- SGI O2 R5000SC 180MHz 最大 17 台
  (主記憶 128MB, OS IRIX 6.3)
- ページサイズ 8kbytes, Write-invalidate 使用
- 100BASE-TX シェアードハブで結合

この環境で、3 つのプログラムにより評価した。

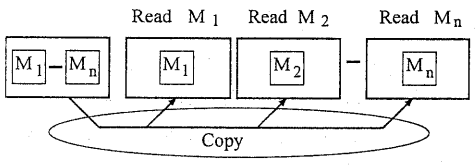
(a) 2 ノード用意し、一方で共有変数を宣言した後、他方で読み出す。



(b) 多数のノードでそれぞれ宣言した共有変数を、1 つのノードで読み出す。



(c) 1 つのノードで宣言した多数の共有変数を、多数のノードから読み出す。



全てのテストについて、cc-COMA の場合の他、ページフォルトハンドラによってキャッシュ制御ルーチンを動かすモード(COMA モードと呼ぶ)と比較した。結果を以下の表に示す。実験(b)(c)に関しては、ノード数は並列の読み出し/書き込みに利用したノード数で表す(よって総ノード数は 1 多い)。

実験(a)	転送総バイト数					
	64kB	256kB	1MB	2MB	4MB	16MB
cc-COMA	3.68	3.13	4.76	4.00	4.08	4.18
COMA	2.00	2.00	2.00	2.00	2.00	2.00
※単位 Mbyte/秒						

実験(b)	1 ノードあたりの共有メモリバイト数					
	64kB	256kB	1MB	2MB	4MB	16MB
cc-COMA						
2 ノード	3.13	5.56	6.06	6.15	6.20	6.46
4 ノード	6.25	0.89	3.05	5.03	6.11	6.35
8 ノード	0.47	1.71	3.27	4.26	5.84	未計測
16 ノード	0.48	0.95	2.96	5.38	5.22	未計測
COMA						
2 ノード	1.60	2.17	2.17	2.17	2.17	2.18
4 ノード	1.81	2.17	2.17	2.17	2.17	未計測
8 ノード	1.81	2.20	2.16	2.17	2.16	未計測
16 ノード	2.17	2.17	2.17	2.17	未計測	未計測
※単位 Mbyte/秒						

実験(c)	1 ノードあたりの共有メモリバイト数					
	64kB	256kB	1MB	2MB	4MB	16MB
cc-COMA						
2 ノード	3.13	4.17	4.17	4.17	4.17	4.16
4 ノード	4.17	5.56	4.65	4.08	4.35	4.23
8 ノード	5.00	4.76	4.21	4.12	3.62	未計測
16 ノード	5.00	4.35	4.08	4.08	3.80	未計測
COMA						
2 ノード	2.00	2.63	2.41	2.40	2.37	2.39
4 ノード	2.22	2.44	2.44	2.42	2.39	2.39
8 ノード	2.94	2.56	2.42	2.38	2.41	未計測
16 ノード	2.78	2.48	2.43	2.40	2.34	未計測
※単位 Mbyte/秒						

なお、一部ノード数が多くなると主記憶不足により計測できなかった。また、実験(b)で一部異常に性能が低いのは、パケットロス発生時の再送アルゴリズムに問題があるとみている。

これらの結果から、以下のような読みとれる。

- cc-COMA の場合、単一ノードからのデータ送り出し性能は 4〜5MB/s であり、ネットワークの理論最大値の 4 割程度である。
- COMA モードの場合、ページフォルトのオーバヘッドにより性能は半減する。
- 実験(b)より、cc-COMA 時の単一ノードのデータ受け取り性能は 6MB/s 以上になる。
- 実験(c)の COMA の結果より、読みとりが並列になるとページフォルトが並列に発生するため性能が向上する。cc-COMA の結果は送出性能で律速していると考えられる。

5 おわりに

本研究で示した実験結果により、コンパイラによるメモリ整合性制御タイミングの指定によって通信速度が倍増できること示された。しかし絶対性能がネットワーク性能の理論最大値に比して低く、コンパイラ開発の土台とするためにさらなる改良が必要であると判断している。そこで現在、新しいソフトウェア DSM の構築を開始している。特に通信の性能を向上させるため、現在 Linux カーネル内に DSM 専用の通信スタックを作成中である。現在までに、システムコール呼び出しオーバヘッドが半減するなどの結果を得ている。これを用いて通信路を使い切る性能を狙う予定である。

謝辞

本研究の一部は、日本学術振興会 未来開拓学術研究推進事業「知能情報・高度情報処理」第 5 プロジェクト「分散・並列スーパーコンピューティングのソフトウェアの研究」および文部省科学研究費基盤研究(C)課題番号 09680340 の支援による。

[1] High Performance Fortran Forum : High Performance Fortran 2.0 公式マニュアル, シュプリンガー・フェアラーク東京, 1999

[2] T. Uehara , S. Saito, K. Joe and Y. Kunieda : The Design and Implementation of cc-COMA as a Platform for Distributed-Memory Machines, In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99),* June 28 – July 1 1999. Las Vegas, Nevada, U.S.A.

[3] B.N. Bershad, M.J. Zekauskas and W.A. Sawdon : The Midway distributed shared memory system., *CMU Technical Report CMU-CS-91-170,* September 1991.