1K−6

並列プログラムのデバッグのためのトレース手法

井上 和紀　矢向 高弘　天野 英晴　安西 祐一郎

慶應義塾大学

1 はじめに

一般に、並列（あるいは並行）に動作する複数のプロセスからなるプログラムは非決定的であり、同一の入力に対して同じ動作を行なう保証がない。このため並列プログラムをデバッグするためには、過去の動作を何らかの形で再現する必要があり、再現に必要な情報を記録するための並列プログラムトレーサが重要な役割を持つ。本研究では、共有メモリ型並列計算機上で、実際のプログラムの動作履歴を効率良く記録できるトレース手法を提案し、実装する。

2 デバッガの開発方針

本研究が目指すのは、バス結合共有メモリ型並列計算機における実用的なデバッグ環境の構築である。これは並列プログラムのトレースと再生に基づく、システムプログラマ向けのデバッグ・ツールであり、ハードウェアに近いレベルでプロセッサ間のプリミティブなインタラクションを把握・検証することを目的とする。

アプローチとしては、まずデバッガのシステムを、プログラムを実行しながら履歴を取る部分(レコーダ)と、繰り返し再生して動作を観察する部分(プレーヤ)に分離する。また、PU間の通信に使われるデータ領域(共有メモリ、メッセージバッファ等)を「共有オブジェクト」として扱う。並列プログラムの非決定性は共有オブジェクトを操作するPUの順序が一意でないことによるから、再現性を持たせるには各PUの操作の順序関係を記録する必要がある。そこで、共有オブジェクト一つ一つ(共有メモリであれば、各アドレス)に、バージョンナンバ(版数)持たせ、PUがある共有オブジェクトにアクセスした場合、対応するバージョンナンバを1増加させる。さらに、アクセスしたPUは、その時「何番目のバージョンにアクセスしたか」を記録する。全PUのアクセス記録を合わせれば、その共有オブジェクトがどのような順序で操作されたか特定できる。後に実行を再現する場合には、各PUが自分のアクセス記録に従って、「適切なバージョンナンバになるまで」そのオブジェクトのアクセスを待てばよい。このアプローチはInstant Replay[1]で採択されたものである。

以下に示す通り、本研究ではレコーダ部を効率良く実現するためのトレーサを設計し、慶應義塾大学とAllumer株式会社で開発したバス結合共有メモリ型並列計算機ATTEMPT[2]に実装した。ATTEMPTはPUにモトローラMC68030を用いた並列計算機のテストベッドで、強力な同期機構を持ち、Futurebusを介して最大20台まで結合することができる。

3 トレーサの設計

トレーサを設計するにあたって、汎用性及び実用性の観点から、以下のような条件を考慮した。

- ハードウェアに特別な改造を加えない。(バス監視用ハードウェアの追加、プロセッサのマイクロコードの変更等は一切行なわない。)
- トレース対象となるプログラムには、通常の実行コードを使用する。(プリプロセッサやコンパイラに変更を加えない。)

これらの条件の範囲で、

- 速度を極力低下させない。
- 履歴情報の総量を小さくする。

という点も可能な限り満足するよう考慮した。(ただし、履歴情報の削減は本質的にコンパイラレベルから上の問題であり、実行コードから冗長性を判断するのは困難である。)

3.1 共有オブジェクトのアクセス検出

速度の低下を押えるために、プログラムの中断頻度は可能な限り小さくすべきである。今回設計したトレーサでは、メモリフォールトの例外ハンドラを使用することにより、共有メモリのアクセスのみを検出することにした。

これはMC68030のMMU(Memory Management Unit)の機能を利用したもので、全メモリ空間を仮想アドレス上にマッピングすることにより実現する。本来これは補助記憶装置を使用したページング機能を実現するためのものだが、ここでは「仮想アドレス＝物理アドレス」となるようにアドレス変換テーブル(ページテーブル)を作成しておく。この状態では、計算機の動作は仮想記憶モードにする前と何ら変わりがない。

ここで、ページテーブルを操作して特定のページを故意にInvalidにしておけば、そのページ内のアドレスにアクセスした時、BUS ERROR例外割り込みが発生する。これを共有メモリ空間のページに適用すれば、共有メモリをアクセスした場合のみ、プログラムを中断して例外ハンドラに処理を移すことができる。(本当にアクセス禁止領域を読み書きしようとした場合もバスエラーとなるが、これは例外ハンドラ内で区別する)。例外ハンドラ内で実際にメモリの内容を読み書きできるよう、別の(本来アクセスすることのできない)領域を予めイメージ領域として設定しておく。(図1参照)

この方式では、共有オブジェクトの操作に関係ない命令の実行が阻害されず、速度の低下が少ない。また、アクセス検出の有無をページ単位でコントロールできるため、より自由度の高い条件設定が可能である。

3.2 例外ハンドラ

例外ハンドラは、バージョンナンバの制御とアクセス履歴リストの生成を行なう。

バージョンナンバは共有メモリの一部に格納し、(例外処理の対象となる)共有メモリの各アドレスに対応している。あるアドレスがアクセスされた場合、例外ハンドラはそのアドレスの対応したバージョンナンバを1増加させる。

アクセス履歴リストはプロセッサ毎に独立したデータであり、ローカルメモリに格納される。(ATTEMPTは共有メモリの他にPU毎に4Mバイトのローカルメモリを持つことができる)。履歴リストには、アクセスしたデータのバージョン番号が順に記録される。アクセスしたアドレス、データ、リー

Trace Techniques for Debugging Parallel Program

Kazunori INOUE, Takahiro YAKOH, Hideharu AMANO and Yuichiro ANZAI

Keio Univ.

ド／ライト等の情報は、再生時に再び得ることができるので、リストに記録する必要はない。

バージョンナンバの更新は不可分に行なう必要があるため、一つのPUがロックを掛けている間は他のPUは待ちキューに入る。これは整数セマフォを用いて実現している。

また、ATTEMPTは同期機構にシンクロナイザを使用しており、基本的にPU間の同期操作はこれを介して行なわれる。シンクロナイザの操作は特定のメモリ空間を読み書きすることによって行なわれるため、共有メモリと同様にページテーブルを操作して、動作履歴をとることができる。

4 評価

アクセス履歴を保存するためのオーバーヘッドを評価するため、テスト用のプログラムをトレースし、実行時間を計測した。評価用のプログラムは共有メモリの書き込みループで、PU数を変化させて測定した。その結果を表1に示す。(ループ回数は0xfffff=1048575回。オーバーヘッドはトレース有無の場合の時間差をループ回数で除算した値である。)なお、PUの動作クロックは20MHz、共有メモリ書き込時のウェイトはPU間のアービトレーションがない場合で約10クロックに相当する。

オーバーヘッドは1回の例外処理に必要な時間を示し、実際の遅延時間はこの値とトレース対象の共有メモリアクセス頻度との積になる。ただし、PUが複数の場合は対象プログラムによってオーバーヘッドに変化が生じる。これはバージョンナンバのコントロール時にセマフォによるロックををかけるためである。今回使用したテストプログラムは共有メモリのアクセス頻度が極めて高い。遅延時間は共有オブジェクトのアクセス頻度に比例するため、他のアプリケーションの多くは速度低下が図1の結果より小さなものとなることが予想される。

5 終りに

現在のバージョンでは、アクセス履歴がメモリ空間から溢れるほどの長い(共有メモリアクセスの多い)プログラムは最後までトレースできない。今後デバッグシステムを構築していく上では、オブジェクトレベル、コンパイラレベルで履歴の量を少なくする手段を考える必要がある。オブジェクトレベルではループ中の冗長な履歴の削減、あるいは履歴データの圧縮や外部記憶の利用が考えられる。しかし、本質的にはコンパイラ以上のレベルでの最適化が望まれる。即ち履歴を取らなくても再生に支障のない共有データを選別、削除する必要がある。その際、本研究で提案したトレース手法はページ単位でアクセス検出の有無を制御できるため、コンパイラの要求に応じて効率良くトレースを行なうことが可能である。

参考文献

[1] Thomas J. Leblanc, John M. Mellor-Crummey. Debugging Parallel Programs with Instant Replay. *IEEE Transactions on COMPUTERS, Vol.C-36, No.4*, 1987

[2] 鳥居 淳 他. 並列計算機ATTEMPTの実装と評価. 情処第40回全国大会公演論文集(III), 1990

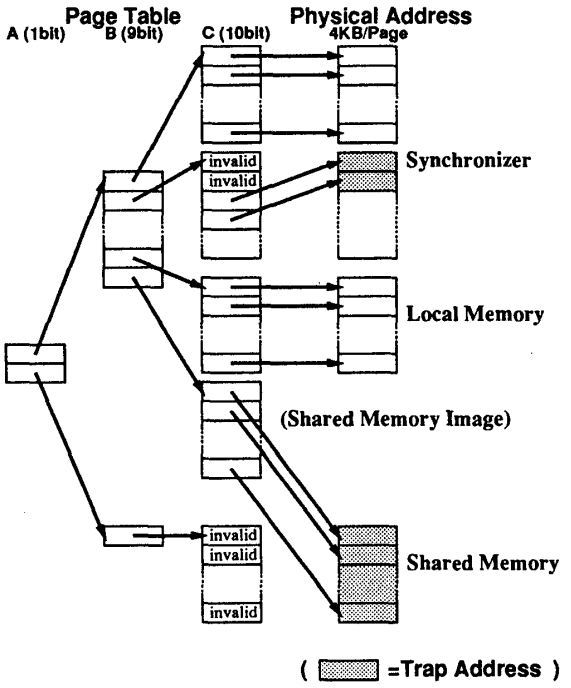

図1: MMUページテーブルの構成

PU数	1	2	3
実行時間:トレース有り(s)	139.0446	212.4075	315.6310
実行時間:トレース無し(s)	6.3401	7.8287	7.6728
実行速度の比(無し/有り)	21.9308	27.1386	41.1366
共有メモリアクセスの頻度($times/s$)	165,386	133,940	136,662
例外処理1回あたりのオーバーヘッド(μs)	126.577	195.102	293.692

表1: トレースによるオーバーヘッド