2B-03

グラフ書換え系から共有メモリ並列プログラムを生成する手法の検討

冨岡 太一 †
早稲田大学 大学院基幹理工学研究科 †

上田 和紀 ‡
早稲田大学 理工学術院情報理工学科 ‡

1 はじめに

近年，プロセッサのコア数の増大により並列プログラムの重要度が高まっている．しかし，ソフトウェア開発で用いられるプログラミング言語として主流である手続き型言語で並列アルゴリズムを実装するのは容易なことではない．特に，プログラムがメモリの動的確保，解放，再利用を含む場合，共有メモリの整合性をとるための同期処理はより複雑なものになる．

そこで，並列アルゴリズムをグラフ書換え言語で実装することを考える．書換え系では計算の非決定性によってプログラムの並列性を表現できる．中でも，グラフを書換えの対象とするグラフ書換え系は図的表現によってプログラムをより直感的に捉えることができる．

しかし，グラフ書換え言語と手続き型言語を結合する方法は自明ではないため，グラフ書換え言語での実装を活用することは実用上問題がある．この問題を解決するために，文献 [1] ではグラフ書換え言語 LMNtal のプログラムから手続き型言語 C の逐次プログラムを生成する手法を考案した．本論文ではさらにグラフ書換え言語のプログラムから並列プログラムを生成するための手法を検討する．

2 並列プログラムの生成手法

2.1 ADLMNtal

本論文では変換元となるグラフ書換え言語として LMNtal [2] の派生言語である ADLMNtal [1] を用いる．LMNtal は無向グラフと書換え規則からなるグラフ書換え系を記述可能な言語である．LMNtal においてはグラフ中のノードをアトム，エッジをリンクと呼び，アトムとリンクからなるグラフをプロセスと呼ぶ．ADLMNtal はプロセスに含まれるアトムを書換えを主導するアクティブアトムとそのほかのデータアトムに分類した上で，書換え規則の左辺に含まれるアクティブアトムが必ず 1 つでなければならないという制約を加えた言語である．

文献 [1] は ADLMNtal のプログラムから逐次 C プログラムを自動的に生成するものである．単純には前述のアクティブアトムは C 言語における関数，データアトムはメモリ上のオブジェクト，リンクはポインタに対応する．各関数の挙動はデータアトムの書換えを再現するように書換え規則から生成される．グラフ書換え系が持つ，書換えの前後でグラフの well-formedness を保存するという特徴を利用することで，生成される C プログラムではポインタが常に有効なメモリ領域を指すようにすることができる．結果として，複雑なポインタ操作を伴いながらもセグメンテーション違反を起こすことのない C プログラムを得ることができる．

2.2 生成される並列プログラムの構造

本研究で生成する並列プログラムは複数のスレッドがデータ競合を起こす共有メモリ並列プログラムとする．この種のプログラムがメモリの確保, 解放, 再利用を含むものであった場合はセグメンテーション違反を防ぐだけでも非常に困難な作業になる．なぜなら，GC を持たない処理系においては複数のスレッドから参照されている領域は他スレッドが解放することでいつ無効になるかわからないためである．この問題には Hazard Pointer など様々な対策が存在する [3] が，性能などとの兼ね合いを考えると非常に複雑な問題である．グラフ書換え系においては書換えの 1 ステップで生成または削除されるデータが明確である．そのため，生成した C プログラムで書換えに対応する操作を不可分操作として行えば他スレッドとのデータ競合によるセグメンテーション違反を防ぐことができる．

2.3 データ構造とアルゴリズム

生成後の C プログラムは Pthreads を用いたマルチスレッドプログラムになる．各スレッドはアクティブアトムを起点とし，書換え規則に基づいてメモリ上のオブジェクトとして表されたデータ構造を処理していく．全てのデータは共有メモリ上に置かれるため，複数スレッドが同時に書換えをする際の整合性を取るために排他制御を行う．

排他制御を実現するために補助ノード [4] とデータにタグを持たせる手法 [5] を用いる．補助ノードはポインタ間に挿入され，バッファの役割をすることでデータの削除時に他のスレッドが不正なポインタを参照することを防ぐ．ここでは補助ノードにタグを持たせ，タグの値をアトミック命令で操作することで排他制御を行う．補助ノードは識別のためのフィールドを持っており，通常のリンクと補助ノードが挿入されたリンクを判別することができる．

図 1 は補助ノードが挿入されたリンクとそのリンクを端点に持つデータアトムの模式図である．点線で囲まれた枠内の箱が補助ノードを構成するメモリオブジェクトである．枠外の箱はデータアトムを表しており，先頭のデータアトムの種類を識別するためのフィールドとポインタからなる．LMNtal ではアト

Towards generating shared memory parallel programs from graph rewriting systems

† Taichi Tomioka, Dept. of Computer Science and Enginerring, Waseda University
‡ Kazunori Ueda, Dept. of Information and Computer Science, Waseda University

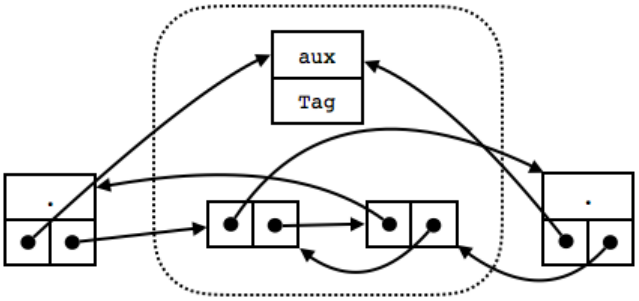

図 1　補助ノードを挿入したリンク

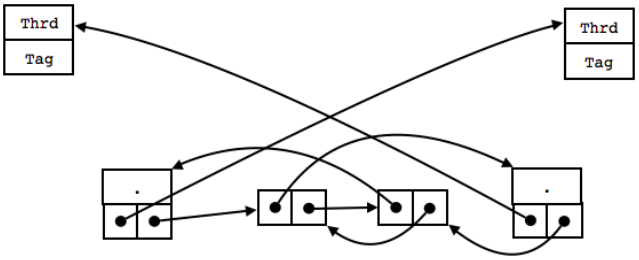

図 2　Owner-Computes Rule を採用したリンク

ムに接続するリンクの端点に順序が存在するため，それを忠実に再現するためにはリンク 1 つにつき 2 つのポインタが必要となる [1].

さらに, キャッシュヒット率を高めるために owner-computes rule を採用する．全てのアクティブアトムとデータアトムはいずれかのスレッドに所有されるとする．そして，全てのアトムは所有者であるスレッドからしか読み書きできないとする．その場合，所有者の異なるアトム同士を接続するリンクはデータ競合を起こす可能性があるため，排他制御を可能にするために補助ノードを挿入する．

図 2 は所有者の異なるアトム同士を接続するリンクを表現するデータ構造の模式図である．各スレッドは 1 つのタグを持ち，書換えを始める際にタグの値によって書換えが可能かどうか判断する．書換えの際図 2 のデータ構造に遭遇した際，他のスレッドのタグを参照することで排他制御を行う．

2.4　議論

通常のリンクと補助ノードを挿入したリンクは混在させることができる．したがって，どのリンクに補助ノードを挿入するか選択する余地が生まれる．全てのリンクに補助ノードを挿入することで細かく排他制御することができるが，その分アトミック命令の数が増えて性能が著しく悪化する．そのため，補助ノードはなるべく少なくすることが望ましい．しかし，複数スレッドが同時に書き換えうるリンクには補助ノードを挿入しなければデータの整合性を保てない．

生成後のプログラムで効率性と安全性を両立できるように補助ノードを挿入することが重要となる．リストや木といった規則性のあるデータ構造をその構造を変化させずに扱う際には挿入箇所はほとんど自明に決定することができる．しかし，グラフ構造を持ったより複雑なデータ構造を扱う場合やデータ構造を大きく変化させる場合は最適なアトムの所有権の割り当ては難しい．これらの問題については，生成元となる ADLMNtal プログラムを静的に解析することである程度決定できると考えられる．

試験実装では排他制御に単純なスピンロックを用いた．すなわち，書換え規則の適用時に書換えの対象となる補助ノードのアクセス権を得たスレッドがタグの値を書換え，その後書換え規則の適用が終わった時点で補助ノードのタグの値を書き戻す．こうすることである補助ノードへの同時アクセスが可能なスレッドが 1 つに制限されるため，そのリンクを含む競合する書換え規則は同時には実行されない．この実装ではプログラムが livelock に陥る可能性があるが，実際に資源飢餓が起きる確率は極めて低く，実用上はスピンロックで問題ないと考えられる．なぜなら，多くの場合，グラフ書換え系で書換え 1 ステップに読み書きするデータはグラフ全体に比べて十分に小さく，1 回の書換えは速やかに終了するためである．より複雑なアルゴリズムを排他制御に用いればロックフリー性を保証することもできると考えられる．

3　まとめ

本論文ではグラフ書換え系から手続き型言語の共有メモリ並列プログラムを生成する手法を考案した．生成したプログラムは並列性に依拠するエラーを起こさずに実行できた．これによりユーザが容易に並列プログラムを得ることができるようになった．しかしながら，排他制御を実現するために必要十分で高効率な補助ノードの配置は自明ではないため，並列プログラムの自動生成手法は今後の課題である．

参考文献

[1] 冨岡 太一 and 上田 和紀. “グラフ書換え言語 LMNtal からの C プログラムの自動生成”. In: **日本ソフトウェア科学会 第 *34* 回大会**. 2017.

[2] Kazunori Ueda. “LMNtal as a hierarchical logic programming language”. In: *Theoretical Computer Science* 410.46 (2009), pp. 4784–4800.

[3] Maged M. Michael. “Safe Memory Reclamation for Dynamic Lock-free Objects Using Atomic Reads and Writes”. In: *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*. PODC ’02. ACM, 2002, pp. 21–30.

[4] John D Valois. “Lock-free linked lists using compare-and-swap”. In: *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*. ACM. 1995, pp. 214–222.

[5] Maged M Michael. “High Performance Dynamic Lock-free Hash Tables and List-based Sets”. In: *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA ’02. ACM, 2002, pp. 73–82.