共有メモリ並列計算機上の並列ガーベージコレクタの性能予測

Predicting Performance of Parallel Garbage Collectors on Shared Memory Multiprocessors

遠藤 敏夫 Toshio ENDO　田浦 健次朗 Kenjiro TAURA　米澤 明憲 Akinori YONEZAWA

東京大学大学院 理学系研究科
Graduate School of Science, the University of Tokyo

概 要

本稿は共有メモリ並列計算機における並列マークスイープガーベージコレクタ (GC) の性能予測モデルを提案する。モデルはGC時のオブジェクトグラフと計算機のメモリアクセスコストを入力として受取り、任意のプロセッサ数による並列マーク時間の予測値を出力する。GCは頻繁にメモリアクセスを行うため、良好な予測精度を得るには計算機間のメモリアーキテクチャの差異を考慮することが必須である。本モデルはキャッシュミスコストだけでなく、メモリノードにおけるアクセス要求の衝突コストを考慮する。対象型共有メモリ計算機 Ultra Enterprise 10000 と、分散共有メモリ計算機 Origin 2000 上の実験を通して実測と予測の比較することにより、本モデルの正当性を示す。

1 はじめに

追跡型ガーベージコレクション (GC) の主要な処理は、生きたオブジェクトからなるグラフを追跡することであり、この処理は並列性を持つ。我々はこれまでに共有メモリ並列計算機上で、GC処理を複数プロセッサで並列に行うマークスイープGC処理系を開発した [4]。実験を通して、この並列GCは対象型共有メモリ計算機 (SMP) 上で良好な速度向上を得られるにも関わらず、非対象型共有メモリ計算機 (DSM) 上では充分な性能を得られないことがあることが分かった。つまり、これらの計算機間では、記述間での移植性が保証されたとしても、性能面での移植性が保証されない。

この問題の大きな原因の一つは、計算機間のメモリアーキテクチャの差異である。共有メモリ並列計算機上のプログラムの性能は、メモリアクセス遅延やアクセス衝突などのメモリアーキテクチャに関する要因により大きく影響を受ける。それ以外にもタスクの並列性の規模やクリティカルパス等の要因も性能に影響する。

本論文の目的は並列GCの性能を低下させる要因の影響を定量的に理解することである。このために、並列GCの性能予測モデルを提案する。キャッシュミスコストをとらえるために、メモリアーキテクチャのモデル化を行なう。一方、タスクの並列性やクリティカルパスなどを考慮するために、Cilk 性能モデル [2] [6]を利用する。モデルの正当性を、予測性能と実測性能の比較を行うことによって確かめる。実験にはSun Enterprise 10000 (SMP) と SGI Origin 2000 (DSM) を用いる。

2章で予測対象である並列GCについて触れ、3章で提案する予測方法を説明する。4章で予測性能と実測性能の比較を行う。5章で関連研究について述べる。

2 並列マークスイープガーベージコレクタ

本研究の対象とする並列プログラムは、[4]で我々が報告した、共有メモリ型並列計算機のための並列マークスイープガーベージコレクタである。この並列GCは、CやC++のための保守的GCライブラリであるBoehm-Demers-Weiser GC [3]を基に、メモリ確保処理とGC処理を並列化拡張したものである。GC要求が起きると、全ユーザスレッドを停止し、複数のGCスレッドが協調的にマーク／スイープ処理を行う。本論文で性能予測の対象とするのは並列マーク処理である。

ヒープ中の全オブジェクトはスレッド間で共有される。GCが始まると各GCスレッドは、ルート (各ユーザスレッドのレジスタとスタック、大域変数) から再帰的に探索を行い、見つかったオブジェクトのマークビットを次々に不可分に更新していく。あるGCスレッドの仕事がなくなると、そのスレッドは別のGCスレッドから仕事を盗む。全スレッドの仕事がなくなったとき、マー

クフェーズは終了する。

このマーク処理を並列プログラムとして見たときの特徴は以下の通りである。

- 「並列マーク処理のタスクグラフ = GC 開始時のオブジェクトグラフ」であり、一般にはその構造は不規則である。
- 頻繁にメモリアクセスを行なうプログラムであり、アクセスコストが性能に大きく影響する。

3 我々の予測手法

3.1 概　　要

提案する予測器は GC 時のヒープスナップショットを入力として受取り、P プロセッサで並列マークフェーズを行なう場合の実行時間を予測する。図 1は予測方法の概要を示す。予測を 3 段階に分けて行なう。まず、ヒープスナップショットを調査することにより、マーク処理の仕事量やメモリアクセスパターンを得る (3.2節)。次にキャッシュミスコストを含まない並列実行時間 T_P を求める (3.3節)。最後にキャッシュミスコストを含む実行時間 T''_P を求める (3.5節)。T''_P は、メモリノードにおけるアクセス衝突のコストも含む。

3.2 ヒープスナップショットの調査

予測の最初の段階で、ヒープスナップショットを調査することによりいくつかのパラメータを取得する。パラメータは仕事量 T_1、クリティカルパス長 T_∞、逐次実行時のキャッシュミス数 Q_1、アクセス分布 V_j である。本稿では 2 次キャッシュミスにのみ注目する。

仕事量 T_1 はキャッシュミスコストを含まない逐次マーク時間である。T_1 は生きたオブジェクトの数とサイズから求めることができる。同様にクリティカルパス長 T_∞ をオブジェクトグラフの深さから得る。

Q_1 は逐次にマークを行なった場合のキャッシュミス数であり、メモリアクセスパターンとキャッシュの構造から求めることができる。キャッシュミス数だけでなく、メインメモリアクセスの対象メモリノードの分布も得る。このパラメータはアクセス衝突コストの計算のために必要である。V_j を、メモリアクセスの対象ノードが j 番ノードである確率とする。

3.3 ミスコストを除いた性能予測

キャッシュミスコストを除いた予測マーク時間である T_P を、Cilk 性能モデル [2] [6]を利用して求める。このモデルにおいて、仕事量 T_1, クリティカルパス長 T_∞ であるプログラムの P プロセッサでの並列実行時間は $T_P \approx T_1/P + c_\infty T_\infty$ となる。c_∞ は仕事移動などのコストに影響される定数である。この議論は Cilk のスレッドスケジューラを前提としているが、並列マーク処理にも適用できる。本稿の性能評価では予備実験を通して得た値 $c_\infty = 4$ を用いる。

3.4 ミス数に関する仮定

ここでは並列実行時のキャッシュミス数について議論する。 3.2節で述べたように、ヒープから逐次実行時のキャッシュミス数 Q_1 を取得する。しかし性能予測に必要なのは並列実行時のキャッシュミス数 (以下 Q_P とする) であり、一般には非決定性を持つ。Q_P と Q_1 の関連について、 Frigo [6]は、 Cilk のスケジューラを前提とすると $Q_P = Q_1 + O(PT_\infty)$ であることを示した。本稿では Q_1 より PT_∞ が充分小さいことを仮定し、 Q_P と Q_1 の差を無視することにする。

3.5 ミスコストを考慮した性能予測

キャッシュミスコストも含めた予測マーク時間 T''_P について述べる。待ち行列理論を利用することにより、アクセス衝突コストも含めて予測を行なう。

まずアーキテクチャのモデルについて述べる。並列マシンは P 個のプロセッサと M 個のメインメモリノード、それらを結合するネットワークからなる。各プロセッサはそれぞれキャッシュメモリを持つ。

各プロセッサがキャッシュミスを起した場合、アクセス要求を対象アドレスのホームメモリノードに送り、返答を待つ。この時送信先がメモリノード j である確率は $V_j (0 \leq j < M)$ である。各メモリノードは待ち行列を持ち、各アクセス要求は行列内の先行アクセス要求の処理が終了まで待たされる。またアクセス要求の衝突はメモリノードでのみ起こり、ネットワークの他の部分では起こらないと仮定する。メモリアーキテクチャの性能をネットワークの一方向遅延 S_l と一度のアクセス要求によるメモリノードの占有時間 S_o で表現する。アクセス衝突がなければ、一回の通信時間は $2S_l + S_o$ であるが、一般にはアクセス衝突による待ち時間分長くなる。

以下、待ち行列理論を用いて衝突コストを含めたマーク時間 T''_P を求める。各プロセッサは T''_P 時間のうち平均 Q_1/P 回のキャッシュミスを行なうので、頻度 (単位時間あたりの回数) は平均 Q_1/PT''_P である。メモリノード j を対象とした全プロセッサからのアクセス要求頻度の合計は $V_j Q_1/T''_P$ なので、メモリノード j における各アクセス要求の平均待ち時間は $S_o/(1 - S_o V_j Q_1/T''_P)$ となる。ここから、 T''_P は以下の式で表される。

$$T''_P = T_P + 2S_l \frac{Q_1}{P} + \sum_j \frac{S_o}{1 - S_o V_j Q_1/T''_P} \cdot \frac{V_j Q_1}{P}$$

一般に T''_P を単純な多項式で表すことはできないため、後述する実験では Newton 法により計算する。

3.6 ハードウェアパラメータ

3.5で説明したハードウェアパラメータを、各並列計算機上でのベンチマークにより得た。

3.6.1 Origin 2000 のモデル化

SGI Origin 2000 (以下 O2000) は 195MHz の R10000 プロセッサを持つ分散共有メモリ並列計算機である。計算機は、 2 つのプロセッサと 1 つのメインメモリノードを持つノードから成り立つ。ノード外のメモリに対するリモートアクセスの遅延はローカルアクセス時より大きい。 Origin 2000 でのメモリ配置方式を以下に述べる。16KB のページごとに配置が決定され、各ページははじめてアクセスしたプロセッサと同じノードに配置される。メモリ領域の配置がソフトウェアによって制御することができるため、メインメモリノード間でのメモリ配

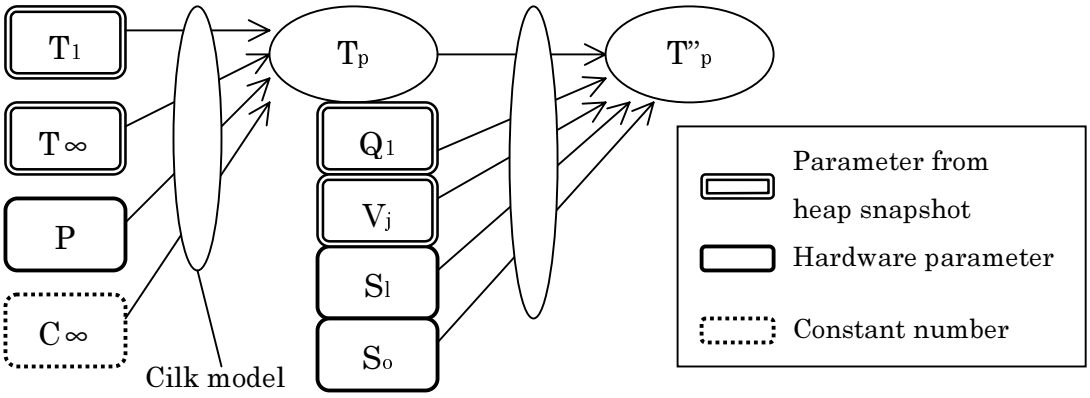

図 1　性能予測方式の概要。ヒープスナップショットを表すパラメータとアーキテクチャを表すパラメータを入力とする。

置の不均衡が起こりうる。

ベンチマークを通して、$S_o = 230 \sim 490$ (ns) を得た (2 つの値はリード時とライト時を表す)。また、S_l はローカル時に 20 ～ 180(ns), リモート時に 260 ～ 500(ns) である。

3.6.2 Enterprise 10000 のモデル化

Sun Enterprise 10000 (以下 E10000) は 250 MHz の Ultra SPARC プロセッサ を 64 個持つ対象型共有メモリ並列計算機である。メモリアクセスコストはいつも均一であり、唯一のメモリノードを持っていると仮定してよい。このとき全てのアクセス要求の対象は唯一のメモリノードであり、$M = 1, V_0 = 1.0$ となる。また、$S_l = 270 \sim 290$ (ns), $S_o = 16 \sim 26$ (ns) である。

4 実験結果

本章ではこれまでに述べた方法による並列マーク処理の予測性能を、実測性能と比較する。並列プログラムの過程で起こった複数回 GC の平均マーク速度について調査する。

実験に用いた並列プログラムは以下の通りである。これらのプログラムは C/C++ で書かれており細粒度スレッドライブラリ StackThreads/MP [8] を用いて並列化されている。

N-Body Barnes-Hut アルゴリズム [1]により N 個の質点の運動をシミュレートする。実験に用いたプログラムはメモリ確保を逐次に行うため、O2000 においてオブジェクトのほとんどが特定メインメモリノード上に集中してしまいやすい。

CKY 自然言語の文法規則ファイルと文章を入力とし、可能な構文解析木を全て求める [7]。O2000 においてオブジェクトの配置は比較的分散している。

4.1 予測結果の評価

図 2は、3 つのアプリケーションの並列マークフェーズ単体の性能を示す。左側が O2000 上の結果で右側が E10000 上の結果である。各グラフの横軸はプロセッサ数であり、縦軸は逐次実測値に対する台数効果を表す。'Real' は実測を、'Pred' は予測値を表す。また 'Pred(no-c)' はメインメモリノードにおけるアクセス衝突がないと仮定したときの予測値である。クリティカルパス長にかかる係数 $c_\infty = 4$ としている。

N-Body においては、E10000 では台数増加に伴い順調な速度向上が見られるが、O2000 では 32 プロセッサ程度で性能は頭打ちになってしまう。提案したモデルはその特性を捉えることに成功している。また、O2000 の 'Pred' と 'Pred(no-c)' の間には大きな差があり、衝突コストが頭打ちの大きな原因になっていることが分かる。モデルにアクセス衝突を入れない場合、実計算機で起こる頭打ちを捉えることができないため、本モデルにアクセス衝突を入れたのは正当であると言える。

CKY の場合は、'Pred' と 'Pred(no-c)' の差が N-Body に比べ小さい。このことから CKY ではアクセス衝突コストはあまり影響しないといえる。

5 関連研究

並列計算機上のプログラムの性能理解に関する研究はこれまでに数多くなされているが、その多くは通信コストの見積もりが比較的行いやすい規則的プログラムを対象としている。

Cilk [2] [6] はマルチスレッド並列プログラミング言語であり、規則的／不規則的な並列プログラムを記述することができる。Cilk 性能モデルは並列プログラムの Cilk スケジューラ上での実行時間を与える。我々の予測方式は並列マーク時間を見積もるためにこのモデルを用いる。

アーキテクチャのモデル化に関する研究は数多くなされている。その中の LoPC モデル [5]はアーキテクチャをネットワーク遅延、メッセージ処理オーバヘッド、プロセッサ数というパラメータで表現し、さらにメッセージ衝突コストを考慮している。本論文のアーキテクチャモデルは LoPC モデルに強く影響を受けている。

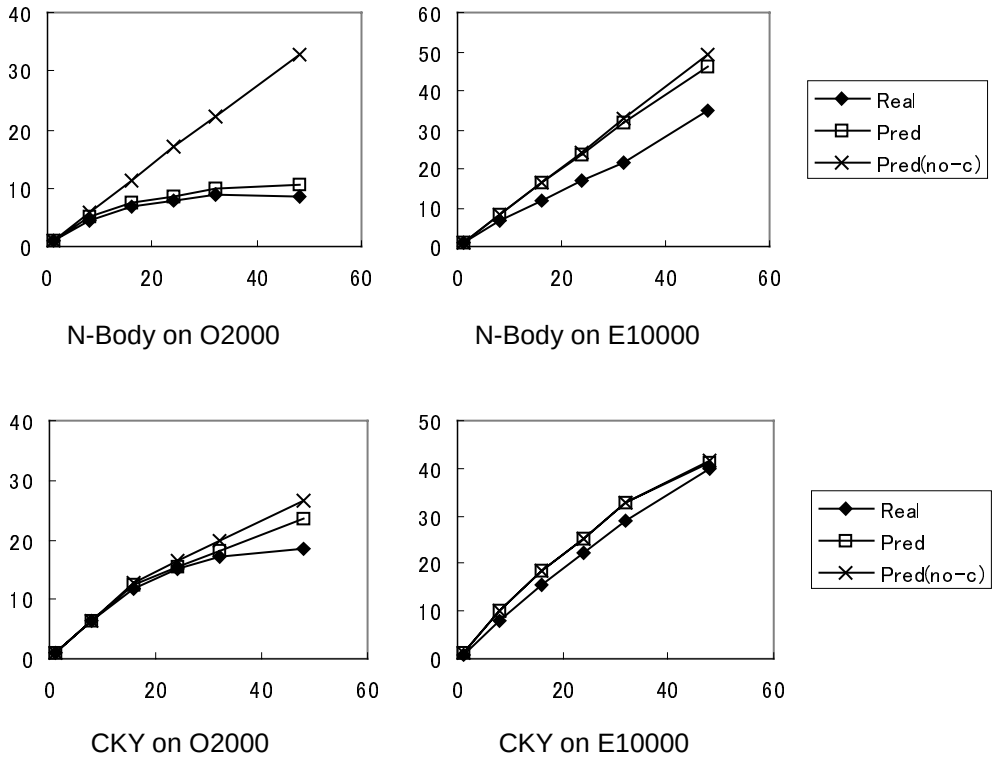

図 2　並列マーク処理の台数効果。各グラフの横軸はプロセッサ数、縦軸は台数効果を表す。

6 おわりに

共有メモリ並列計算機上の並列 GC の性能予測モデルを提案した。対象とした並列 GC は複数プロセッサが協調的に GC 処理を行なう並列マークスイープ GC である。提案した方式はヒープスナップショットを入力とし、並列マーク時間を定量的に予測する。クリティカルパスなどをとらえるために Cilk 性能予測モデルを利用する。一方ハードウェアのメモリ階層による影響をとらえるためにアーキテクチャのモデル化を行なった。このモデルはキャッシュミスの遅延だけでなく、メモリノードにおけるメモリアクセス要求の衝突コストを考慮する。

2 種類の並列計算機上での実験を通し、予測値と実測値の比較を行った。その結果、 Origin 2000 ではメモリ配置の不均衡によるアクセス要求衝突コストが性能に重大な影響を及ぼすことを示した。衝突コストを考慮に入れないモデルでは、性能の挙動を捉えることは決してできないため、衝突コストをモデルに入れたのは正当であると言える。

将来の課題は以下の通りである。

- 予測精度の向上
- ソフトウェア DSM など他のアーキテクチャへの予測モデルの適用
- 予測結果を用いた GC アルゴリズムの自動適応化
- 一般の並列プログラムへの予測モデルの適用

参考文献

[1] Josh Barnes and Piet Hut. A hirarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.

[2] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *The Journal of Parallel and Distributed Computing*, 37(1):55–69, August 1996.

[3] Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Software Practice and Experience*, 18(9):807–820, 1988.

[4] Toshio Endo, Kenjiro Taura, and Akinori Yonezawa. A scalable mark-sweep garbage collector on large-scale shared-memory machines. In *Proceedings of ACM/IEEE Conference on High Performance Networking and Computing (SC97)*, November 1997.

[5] Matthew I. Frank, Anant Agarwal, and Mary K. Vernon. LoPC: Modeling contention in parallel algorithms. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 276–287, June 1997.

[6] Matteo Frigo. *Portable High-Performance Programs*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1999.

[7] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, 1965.

[8] Kenjiro Taura, Kunio Tabata, and Akinori Yonezawa. StackThreads/MP: Integrating futures into calling standards. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 60–71, May 1999.