2002 年 夏のプログラミングシンポジウム 「リソース・コンシャス・プログラミング」 発表資料
2002 – 09 – 19

ごみ集めのための人工ベンチマーク

寺田 実 (電気通信大学情報通信工学科)

本稿では, ごみ集め (以下 GC) の性能評価の手段として, 人工的なベンチマークプログラム (Synthetic Benchmark) について, その必要性, 要求される機能, 実現に関する問題点などについて検討する.

1 GC の性能評価

GC に要求される機能は, ルートセットからの到達不可能セル (いわゆるごみ) を識別して回収するという, 単純なものである. したがって, GC の評価においては, その提供する機能によって評価することはできず, 本質的には性能のみによって評価されるものであるといえる.

性能以外の評価項目としては, 以下のようなものが考えられるが, (1 以外は) 主要な評価点ではない. 1 についても, 問題になるのは既存の言語処理系への新しい GC アルゴリズムの実装といった場面に限られる (たとえば conservative collector の優位性).

1. mutator (プログラム処理を担当する部分) に要求される協力の程度

   たとえば, mutator によるセルのフィールドの書き換え時に, 特別な処理 (ライトバリア) が必要かどうかなど.

2. アルゴリズムの単純さ

   実装において誤りが発生しにくい.

3. 特殊なサービス機能

   Weak Pointer など.

性能評価におけるベンチマークプログラムについては, その選択が重要である. 現実的なアプリケーションの挙動をできる限り反映したベンチマークを用いることで, 実際の利用時の性能が推測できる.

しかしここで, 対象を GC の研究に絞った場合, 研究の形態が以下の二つに大別できることがわかる. それぞれの形態に応じて, 適切なベンチマークが大きく異なるのである.

- 形態 1: 特定の言語処理系の実装が目標の場合

  なにか特定の言語処理系を実装するため, そのひとつのコンポーネントとして, 性能の良い GC の実現を目指すケースである.

  この場合, mutator の実装に影響を及ぼすという点で, GC アルゴリズムの選択は設計の初期段階で行なうことが多く, 性能向上のために行なわれるその後の努力は, おおむねパラメータの最適化程度にとどまると考える.

  また, アプリケーションとは別に処理系が固有に消費するメモリについても考慮に入れることができる. たとえば実行時のスタックフレームやゴールレコードをヒープに割り付けるなどの場合である.

- 形態 2: GC アルゴリズムそのものが目標の場合

  新しい GC アルゴリズムの開発や, 既存のアルゴリズムの改良など, GC アルゴリズムそのものが目標となる研究形態である.

  この場合, GC の正当性や性能を評価するためのテストベッドとして, mutator である言語処理系を実装する必要がある. それに要求される属性としては,

  - GC 実現が容易であること. たとえばライトバリアの実装など.
  - 処理系の実装にかかる労力が低いこと.
  - 言語としての基本機能 (プログラムを実行できること)
  - 種々の計測機能

  などがある. 結果的に使われる処理系は, 標準的な (つまり, 規格が定まった) 処理系のフルセットであることはまれで, 典型的には基本関数だけを備えた Lisp のように, 必要最小限の処理系であることが多い.

  また, 処理系が「本気で」作られていないため, コンパイラを備えていない, 最適化が十分ではないなどの問題点もある. このことが, メモリの余計な消費に結び付く可能性もある.

形態 1 では, ベンチマークとしてはその処理系が扱うことになる代表的なプログラムを採用すれば良く, その実行に特別な問題もない.

たとえば, Java 処理系における GC ベンチマークとしては, SpecJVM といった, 処理系全体の性能をはかるものが使われているようである.

しかしそうした「現実的」なベンチマークは, GC の観点から見た場合の振舞 (セルの生成率, 寿命の分布など) が明らかにされておらず, さらにその振舞が制御できないため, GC アルゴリズムの評価も全体的な実行性能というマクロな視点からだけに限られる.

さらに問題であるのは形態 2 のほうで, 標準的なベンチマークを実行するのに十分なほどには処理系が完備されていないため, その処理系で実行可能なプログラムを用いるほかなく, その選択には恣意性がつきまとう.

また, そうしたベンチマークについても, 形態 1 と同様, ベンチマークの性質が明らかにはならない.

以上をまとめると, 形態 2 の元で行なう GC 研究におけるベンチマークについて, 以下のような問題があるといえる:

- mutator の実装が必要, しかし完全な処理系は無理
- 現実的なベンチマークプログラムが使えない
- ベンチマークの振舞が不明
- ベンチマークの振舞の制御が不可能

2 人工ベンチマーク

人工ベンチマークとは, メモリシステムに付加をかけるためだけのプログラムである. その特徴は以下のとおり:

- 制御可能なパラメータのセットを持っている.

- メモリシステム (アロケータ, セルの書き換えハンドラなど) を直接呼び出すため, mutator の実装は必要ない.
- パラメータを変化させることで, 種々のタイプの付加を発生することが可能で, とくに適切に設定すれば既存の「現実的」ベンチマークのシミュレートも可能であろう.

これを用いれば, 前述の形態 2 における問題点が解消できることになる.
以下で, 人工ベンチマークについて, いろいろな面から検討を加える.

2.1 シミュレートのレベル

人工ベンチマークはベンチマークプログラムをシミュレートするものであるが, どのような局面までをシミュレートするかによって, 以下のようなレベルが考えられる:

1. 寿命も含めて生存セルの個数をシミュレートする.

   このレベルではセルの接続関係は無視し, いわば「ポインタを含まないオブジェクトの集合」として扱うことを意味する.

2. 1 に加えて, セルの接続関係の一部もシミュレートする

   たとえば, セルのフィールド書き換えや, その際の参照先の年齢なども考慮に入れる. これは世代別 GC などの評価の際には必要である.

3. 2 に加えて, セルのメモリ上での配置もシミュレートする

   GC アルゴリズムにおける, ヒープの局所性の向上といった評価のためにはこのレベルが必要になる. そのためには, セルの接続関係を「現実的」ベンチマークと一致させる必要がある.

また, それとは別に,「現実的」ベンチマークではプログラムの実行に段階があることもあり, それぞれのフェーズでの振舞が異なるケースがある. このような「振舞の時間変化」はまた別な問題であるが, 本稿ではこれは検討対象とはしない. つまり, 本稿では振舞が定常的であるベンチマークに限定することとする.

2.2 レベル 1 シミュレータ

セルの寿命分布が主要な制御パラメータである. ベンチマークにおいて絶対的な経過時間には意味がないので, 時刻をアロケート量を単位として計ることにする. (特に, 本稿では振舞が定常であることを仮定するので, 絶対時刻とシミュレートされた時刻とは比例的な関係となる.)
寿命の分布を, 生成から t だけ経過した時点での生存率 $f(t)$ であらわすことにする. この場合,

$$\text{平均寿命} = \int_0^\infty f(t)dt$$

となる. また, セル総量と平均寿命との間には

$$\text{セル総量} = \text{セル生成率} * \text{平均寿命}$$

の関係があり, 独立ではない. さらに, セル生成率は一般性を失わずに 1.0 とできるから, セル総量は平均寿命と一致する.
以下では, いくつかの寿命分布について考察する.

モデル	平均寿命	$f(t)$
(寿命一定) すべてのセルが生成ののち一定の時間 c でごみになる	c	$\begin{cases} 1 & (0 \leq t < c) \\ 0 & (c \leq t) \end{cases}$
(均等に死滅) 生存率が t に対して一定の割合で減少して, 時刻 c で 0 となる	$c/2$	$\begin{cases} 1 - t/c & (0 \leq t < c) \\ 0 & (c \leq t) \end{cases}$
(指数分布) 生存セルのうち一定の割合 $(1/c)$ が死滅していく	c	$e^{-t/c}$
(二段階分布) セルの一部 (α) は c_0 まで生きるが, 残りはそれよりも短い c_1 で死滅する	$\alpha * c_0 + (1 - \alpha) * c_1$	$\begin{cases} 1 & (0 \leq t < c_1) \\ \alpha & (c_1 \leq t < c_0) \\ 0 & (c_0 \leq t) \end{cases}$

2.3 レベル 2 シミュレータ

セルのフィールドの書き換えをシミュレートする. その際, 書き換えの頻度と, 書き換えによって新しい参照先となるセルの年齢分布がパラメータとなる.

さらにモデルを複雑にするのであれば, 書き換えの起こる頻度をすべてのセルで一定にするのではなく, 年齢による分布を導入してもよいであろう.

また, 新しい参照先の年齢分布が, 書き換えを受けるセルや古い参照先の年齢と関係を持たせるという方法もある.

これらについては, 後述するモデル同定によって決定していくべき事項であろう.

2.4 レベル 3 シミュレータ

セルの接続関係を実際のアプリケーションと一致させるには, 実際のアプリケーションのアロケーションのパターンを模倣する以外に有効は方法はなさそうである. つまり, このレベルのシミュレーションをするには, ログの再生が必要になる.

ログを利用する GC の評価の研究はかなり広範に行なわれている. そこでは, ログの再生により, mutator を実装することなく性能評価が可能であり, 有効な手法である.

問題点としては, 以下のようなものが考えられる:

- ログを作成するについては, そのための機能を持ったフルセットの言語処理系が必要である
- ログの分量が膨大になる
- パラメータの操作は不可能である

3　既存ベンチマークのパラメータ同定

人工ベンチマークが既存の「現実的」ベンチマークをシミュレートするためには, そのパラメータを同定しておく必要がある. 寿命分布, フィールドの書き換えの頻度や分布などを測定する必要があるが, そのためには測定用の (フルセットの) 言語処理系を作るか, 前述のログを利用するなどが必要である.

4　人工ベンチマークの実装方式

人工ベンチマークは, 基本的にはメモリシステムが mutator に対して用意する関数群を利用して動作する. たとえば:

- ヒープからのアロケート (例: `cons`)
- セルのフィールドの書き換え (例: `rplaca`, `rplacd`)

動作は, セルを一つアロケートしたのち, 適当な寿命を持つセルを乱数を利用して寿命分布から決定し, それを死滅させることを繰り返す.

この人工ベンチマークの記述言語の選択に二通りがありうる:

1. メモリシステムと同じレベルの言語で記述

   たとえばメモリシステムが C で記述されているのなら, 同じく C で書くということである. mutator を全く作らない形態では, これが唯一の選択肢となる.

2. mutator が実現する (フルセットでない) 言語で記述する

   前述のメモリシステムアクセスの関数は最低限のものであるから, mutator の提供する限定された機能のなかでも利用できる.

このうち, 後者では人工ベンチマークの実装が容易となる利点もあるが, 以下のような問題点もある:

- 人工ベンチマークの実行自身がメモリを消費する可能性がある. たとえば, 実行時フレームをヒープに割り当てる設計になっていると, 意図したセル消費パターン以外にもメモリが消費されてしまう.
- 人工的に生成するセル群のルート

  人工ベンチマークも普通のアプリケーションであるから, メモリシステムへ負荷としてアロケートしたセル群を保持するためにルートからの到達性を確保する必要がある. そのための手段として, たとえば mutator における変数から参照することにすると, その変数は非常に高い頻度で書き換えを受けることになり, ライトバリアの動作によってメモリシステムに大きな干渉を与えてしまう.

  これを回避する一つの方法は, 人工ベンチマークのルートをメモリシステムに「付加的なルート」として提示する方法である. ヒープに含まれていない変数をこの「付加的なルート」とし, その変数から人工ベンチマーク内のデータを参照すればよい.

  同様な方法として, mutator の実行時スタックからヒープへの参照が可能であるならそれが利用できるし, あるいは mutator がルートとして扱う静的な (ヒープ外の) メモリも利用できる可能性がある.

5　人工ベンチマークの実装に向けて

現段階では, 以上のような検討を行なっただけで, 実装には着手していない. 当面計画しているのは以下のような手順である.

1. 既存の言語処理系上で, 前節の 2. で示したように対象言語で人工ベンチマークを記述する. その際, ルートの問題を回避するために, レベル 1 のシミュレータとする.

   そのうえで, 寿命分布の種類や平均寿命をコントロールして, システムの性能を組み込みの測定手段でモニタする.

   この実験によって, 人工ベンチマークのパラメータ制御機能の有用性を確かめたい.

2. さらに, レベル 2 のシミュレータに進む. この場合, ルート問題を解決する必要があるが, 場合によっては既存の処理系を一部修正する必要があるかも知れない.

   これによって, たとえば世代別 GC の性能とパラメータの関係について, 「短寿命のものが大部分を占める場合に有効」といった定説を確認することで有用性を示したい.

3. 複数の GC アルゴリズムを実装したメモリシステムだけを適当な言語で作成し, そのうえで人工ベンチマークを用いた性能計測を行ないたい.