1K-3

並列Haskellを使ったN体問題の実装における評価戦略適用時のパフォーマンスの調査

大石　和也†　　岡本　秀輔†
成蹊大学大学院 理工学研究科 理工学専攻†

1.はじめに 近年、コンピュータの性能向上の方法としてCPUのマルチコア化が主流となっている。そのため、マルチコアCPUの効率的な使用によるアプリケーションの性能向上が期待されている。Haskellは近年注目されている関数型言語の一つで、他の多くのプログラミング言語と比較して、プログラムを容易に並列化できることが特徴である。また、並列Haskellプログラムは評価戦略によって性能が変わるという特徴がある。本研究では、N体問題を用いて並列Haskellプログラムにおける評価戦略ごとの性能の違いについて評価する。

2. 評価戦略 Haskellは遅延評価を行う言語である。遅延評価とは結果の値が必要になるまで値の計算を遅らせる評価手法である。不要な計算を回避できる場合があるため、プログラムの実行時間を大幅に短縮出来る可能性がある。その一方で、値を得るための式そのものは全てメモリ上に展開して保持するため、式が多くなるとメモリの使用量も多くなる場合がある。また、Haskellでは遅延評価の他に先行評価を指定することができ、これらの指定によって実行時間やメモリの使用量を変化させられる場合がある。また、並列Haskellには前述の指定に加えて、並列プログラムの引数の評価の順序を変えるものがあり、これらをまとめて評価戦略と呼んでいる。並列Haskellには4種類の評価戦略がある。これらの評価戦略の指定によって、プログラムの性能が変わる場合がある。

3.N体問題 N体問題はそれぞれの物体が重力によって相互に作用するN体の物体の動きを予測する問題である。本研究では文献[2]で扱われている3次元空間におけるN体問題とそのアルゴリズムについてとりあげる。

3.1 All-Pairsアルゴリズム All-Pairsアルゴリズムは、ある物体と他の物体という2つ1組の物体の間での加速度の計算を、全ての組み合わせに対して行うものである。物体間での計算を完了するまでに $O(N^2)$ の計算時間を必要とする。All-Pairsプログラムでは、全ての物体に対して、速度の更新をする関数と位置の更新をする関数を合成したものをmap関数を使って適用する。

初めにmap関数をparMap関数に置き換えることで並列化を行った。次に、物体をいくつかの塊に分けて、塊ごとに合成関数を並列に適用するような並列化を行った。

3.2 Barnes-Hutアルゴリズム Barnes-Hutアルゴリズムは、木の構築と加速度計算から成り立つ。

・木の構築

与えられた物体から領域を決め、その領域をいくつかの領域に分割する。これを、領域内に存在する物体が1個以下になるまで繰り返す。結果として各領域がノードとなる木が構築される。

・加速度計算

ノードを1つの物体とみなして、ある物体との間の加速度計算を行い、近似的な解を得る。

Barnes-Hutアルゴリズムでは、1つの領域内に含まれる全ての物体が1つの大きな物体として扱われるため、All-Pairsアルゴリズムと比較して、大量の物体を持つデータの計算を行う際に適している。

Barnes-Hutプログラムでも、All-Pairsプログラムと同様の並列化を行った。

4.実験 実験として、All-Pairsプログラムと Barnes-Hutプログラム双方で、逐次プログラムと、塊ごとに合成関数を並列適用する並列プログラムに対して4種の評価戦略を適用したときの実行時間とメモリ使用量を計測した。

実験環境にはIntel Xeon E5420 2.50GH z x2、メインメモリ 16GBを搭載するマシンを使用した。 OSはLinux3.6.9-2.fc17.x86_64、コンパイラはGHC7.4.1である。

4.1.実験結果 All-Pairsプログラムでは物体を16000～24000個の間で指定したとき、Barnes-Hutプログラムでは120000～240000個の間で指定したときの実験結果を示す。

図1,2は、All-Pairsプログラムにおける実行時間とメモリ使用量のグラフである。グラフは、逐次プログラムと、各評価戦略を用いた並列プログラムにおける結果で構成される。図1より、評価戦略rseqとrdeepseqを指定した場合は実行時間にほとんど差はなかった。一方でrparを指定した場合には前述の2つの戦略の約2倍、r0を指定した場合には約4.8倍の実行時間となった。図2より、どの評価戦略の場合も逐次プログラムよりメモリ使用量は多く、各評価戦略において差はほとんどなかった。これは並列化する際に伴うオーバーヘッドによるものだと考え

A Performance Study of n-body problem using Parallel Haskell with Evaluation Strategies
†Seikei University

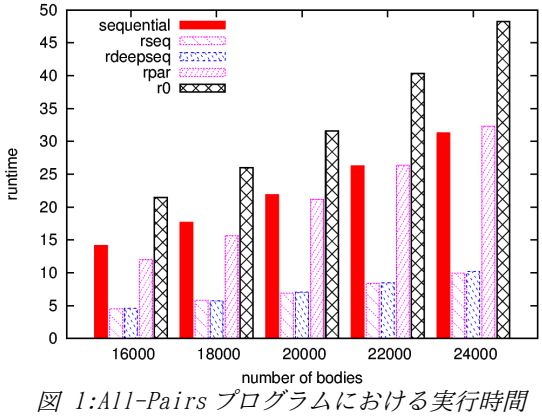

図 1:All-Pairs プログラムにおける実行時間

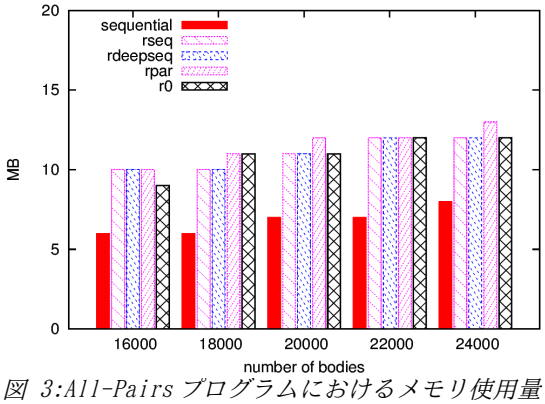

図 3:All-Pairs プログラムにおけるメモリ使用量

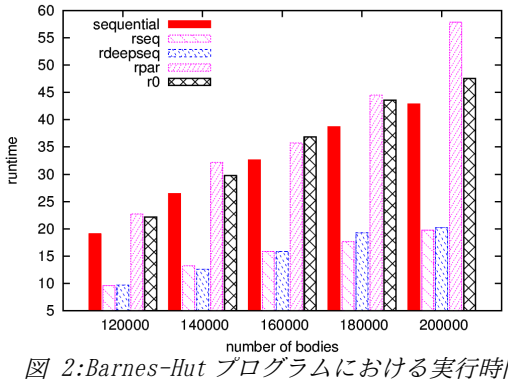

図 2:Barnes-Hut プログラムにおける実行時間

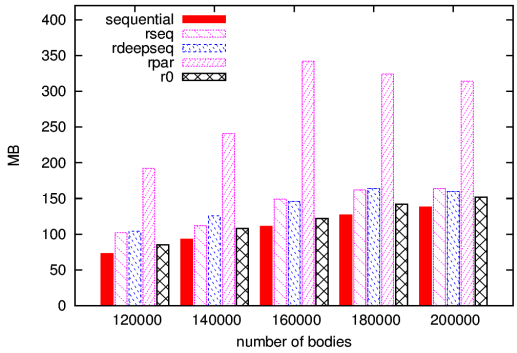

図 4:Barnes-Hut プログラムにおけるメモリ使用量

られる。

図 3,4 は、Barnes-Hut プログラムにおける実行時間とメモリ使用量のグラフである。図 3 より、Barnes-Hut プログラムにおいても rseq と rdeepseq を指定した場合、実行時間にほとんど差が出なかった。また、rpar を指定した場合には、最大で前述の 2 つの戦略の約 3 倍、r0 を指定した場合には、約 2.3 倍の実行時間となった。物体の数に比例して rpar を指定したときの実行時間が長くなっており、物体が 200000 個の場合においては r0 を指定したときの実行時間を約 10 秒上回っている。図 4 より、メモリ使用量はどの評価戦略においても逐次プログラムより多くなった。各評価戦略においては、rpar を指定したときに最も使用量が多くなった。実行時間、メモリ使用量共に rpar を指定したときに最も大きな数値が現れた。All-pairs アルゴリズムでは物体の数のみ、Barnes-Hut アルゴリズムでは、物体の数と木のノード数が計算量に影響すると考えられる。後者のアルゴリズムでは木のノードの数が爆発的に増える可能性がある。それに伴ってスレッドあたりの計算量も大きく増える可能性が生じる。rpar を指定すると更にスレッドを生成しようとするため、オーバーヘッドも大きくなり、実行時間、メモリ使用量共に他の戦略よりも高い数値になったと考えられる。また、双方のプログラムで、物体の数が多くなると rdeepseq の実行時間が微少ながら rseq 指定時を上回った。rdeepseq による簡約に時間が掛かった可能性が考えられる。

5. まとめ 本研究では N 体問題を用いて並列 Haskell プログラムにおける評価戦略ごとの性能の違いについて評価した。All-Pairs アルゴリズムと Barnes-Hut アルゴリズムを用いたプログラムを実装し、計測を行った。実験結果から、どちらのアルゴリズムでも、rpar と r0 を指定した場合には逐次プログラムと同等かそれ以上の実行時間となった。rseq と rdeepseq を指定した場合に他の評価戦略を指定した場合よりも実行時間が短くなった。また、物体の数が多くなった場合、rseq 指定時に実行時間が最も短くなる可能性がある。メモリ使用量は、Barnes-Hut アルゴリズムにおいて、rpar を指定したときに最も多くなった。

参考文献

[1] Simon Marlow, Patrick Maier, Hans-Wolfgang Loidl, Mustafa K.Aswad, Phil Trinder: Seq no more: Better Strategies for Parallel Haskell
[2] Prabhat Totoo, Hans-Wolfgang Loidl: Parallel Haskell implementations of the n-body problem
[3] Tom Ventimiglia, Kevin Wayne: The Barnes-Hut Algorithm, http://arborjs.org/docs/barnes-hut