卒論後経過報告

71005 shuhei 大 角　周 平

1999 年 3 月 4 日

1　はじめに

前評判通り、卒論の発表本番よりも厳しかった米研の発表練習をなんとか乗り越えることができていたので、本番は比較的問題なくこなせたんではないだろうか。H 先生に動機についてつっこまれて窮したのでまずそこから話を始めたい。

並列環境が当然のこととなってきた今日となってはスクリプト言語にもスレッドを装備させておくことはあたりまえのこととなりつつある。

そういう情勢を踏まえた上で、 Python に StackThreads/MP へのスレッドインタフェースを実装する。

ただし、 Python はマルチプロセッサに対応していない、そして粒度の粗いスレッドインタフェースを残念ながら既に持っていたので、本卒業研究では、 StackThreads/MP を用いてマルチプロセッサで動作する、そして細粒度のスレッドインタフェースを代わりに実装することにする。

そのメリットの 1 つは普通ならスクリプト言語で (たぶん) 実行することができない (であろうと思われている) フィボナッチ関数を結構速く実行できることである。[1]フィボナッチを指数関数のオーダーで計算するナイーブなプログラムでも処理系が LTC に基づいてスレッド生成のオーバーヘッドを極力抑えて実行してくれるためにスケーラビリティが出やすくなっている。スケーラビリティの測定結果の詳細は卒業論文に載せたので省略するが、結論を言うと今のままの実装ではスケーラビリティが 14CPU で 6 倍くらいで頭うちになっている (ようだ)。

細粒度のスレッドインタフェースと言っても、今実装では、スクリプトプログラムの並列プリミティブから StackThreads/MP ライブラリの並列プリミティブを呼び出すまでに結構オーバーヘッドが介在している。正確に言うと、インタープリタがスクリプトの解釈に被るオーバヘッドに加えて、 StackThreads/MP の並列プリミティブを呼び出すまでの手続きにもオーバヘッドが存在している。[2]もちろん、そのオーバヘッドは Stack-

[1] どれくらい速いかの測定は未だ

[2] 正直に白状すると malloc もしている:-(

Threads/MP のレベルでタスクスティーリングが起きようと起きまいと[3]、いずれの場合にも結構大きいと思われる。(LTC を無意味にしているとまでは思われない)

そもそもスクリプトの細粒度って本質的に矛盾するものが存在するような気もする。

細粒度の定義はいまいち不明瞭になってしまうのだが、「スレッド生成のコストが小さい」という定義では、StackThreads/MP と比較すると細粒度ではない。しかし、「無数のスレッド生成を許容する」という定義では実際にフィボナッチを計算できているわけで十分に細粒度といってもよい (はずだ)。Pthread では計算できない指数フィボナッチ数を計算できている。

2 Current Treand and Countermeasures of My Research

2.1 ヒューチャーワークの後かたづけ

卒論後は小さいペンディングワークを 1 つ 1 つ片付ける作業も並列に行った。一方に 3 大フューチャーワークとして 1、台数効果問題の原因究明、2、無効にした Dealloc の復活、3、Python の参照カウンタによる GC と、SGC のマークスィープによる GC の性能比較が挙げられるが、いずれも一朝一夕にできる仕事ではないので頑張ってやりたいと思う。

[3] StackThreads/MP では Task Stealing が起こらない場合は約 10 RISC 命令 (起きる場合は?) に較べると ....