ode Segment と Data Segment を持つ Gears OS の設計

Shohei KOKUBO

22/Feb/2016

並列環境下におけるプログラミング

マルチコア CPU の性能を発揮するには、処理をできるだけ並列化しなければならない。これはアムダールの法則により、並列化できない部分が並列化による性能向上を制限することから言える。つまり、プログラムを並列処理に適した形で記述するためのフレームワークが必要になる。

並列環境下におけるプログラミング

マルチコア CPU 以外にも GPU や CPU と GPU を複合したヘテロジニアスなプロセッサが登場している。並列処理をする上でこれらのリソースを無視することはできない。しかし、これらのプロセッサで性能を引き出すためにはそれぞれのアーキテクチャに合わせた並列プログラミングが必要になる。並列プログラミングフレームワークではこれらのプロセッサを抽象化し、CPU と同等に扱えるようにすることも求められる。

Cerium の問題点

- Task 間の依存関係
  - データの正しさを保証できない
- データの型情報
  - 並列処理の組み合わせが型的に安全なのか保証できない
- メモリ確保
  - ある Thread がメモリを確保しようとしている間、他の Thread はメモリを確保することができない。結果、処理速度の低下に繋がる
- オブジェクト指向と並列処理
  - 一般的に参照透過な処理は並列化を行いやすい。一方、オブジェクト指向ではオブジェクトの状態によって振る舞いが変わるため並列処理との相性が悪い

Gears OS

- Code Segment と Data Segment という単位で構成される
  - Code Segment は並列処理の単位
  - Data Segment はデータそのもので型情報を持つ
- Continuation based C(CbC) による実装
  - 並列化、ループ制御、関数コールとスタックの操作を意識した最適化がソースコードレベルで行える
- Allocator
  - Thread ごとに独立したメモリ空間を持つ
- TaskQueue
  - Task の管理
  - CAS 命令を利用して操作することでデータの一貫性を保証する
- Persistent Data Tree
  - Data Segment の管理
  - 非破壊木構造で構築するので平行して読み書き可能
  - Red-Black Tree アルゴリズムを用いて平衡性を保ち、挿入・削除・検索における処理時間を保証する

Gears OS

- Worker
  - TaskQueue から Task を取得し、実行
  - 並列に実行される Code Segment は通常の Code Segment と同等
    - 依存関係のない Code Segment はすべて並列に動作させることが可能
- 1 CPU と 12 CPU で約 11.8 倍の性能向上を確認
- Gears OS の実装自体が Gears OS を用いて並列処理を記述する際の指針となっている