Code Segment と Data Segment を持つ Gears OS の設計

小久保 翔平(並列信頼研究室)

並列環境下におけるプログラミング

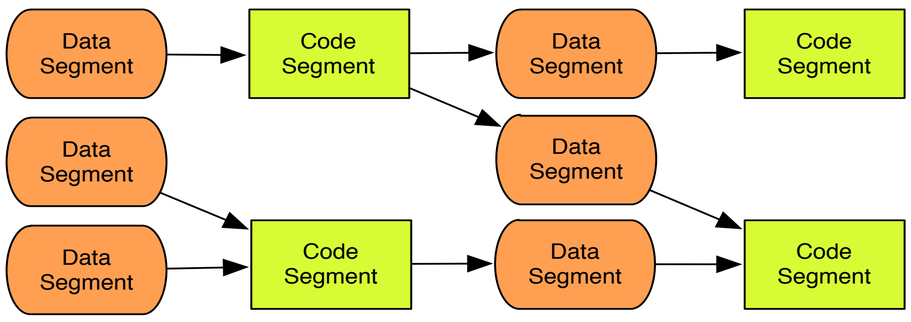
・マルチコア CPU の性能を発揮するには、処理をできるだけ並列化しなければならない
・アムダールの法則により、並列化による性能向上は並列化されていない部分の割合に大きく影響される
・並列化可能な部分の特定や依存関係の設定など並列プログラミングは難しい
・並列処理に適した形で記述する方法が必要になる
・本研究では先行研究である Cerium の開発で得られた知見を元に Code Segment と Data Segment を持つ Gears OS を設計した

Cerium の問題点

・Task 間の依存関係だけではデータの正しさを保証できない
・汎用ポインタを用いてデータの受け渡しを行うので、型情報が落ちて型システムで Task の組み合わせを検査することができずプログラムの正しさを保証することができない
・Allocator を Thread 間で共有しており、ある Thread がメモリ確保している間は他の Thread はメモリ確保することができず並列度の低下に繋がる
並列度の低下に繋がり、処理速度が低下する
・一般的に参照透過な処理は並列化を行いやすいが、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

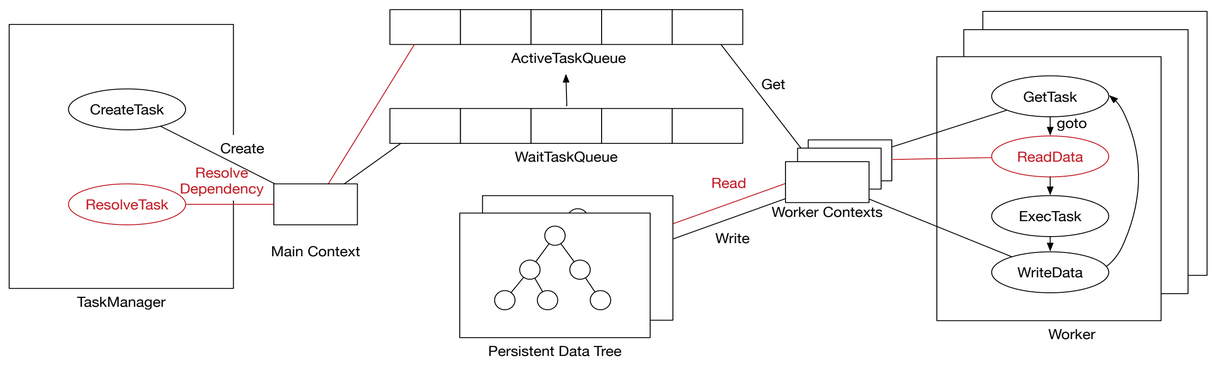
Code Segment と Data Segment

・並列実行の単位、データの分割、Code Segment 間の接続などになる
・Code Segment
任意の数の Data Segment を参照し、処理が完了すると任意の数の Data Segment に書き込む
接続された Data Segment 以外にアクセスすることができない
・Data Segment
分割されたデータそのもの
整数や文字列などの Primitive Data Type を複数持つ構造体として定義
型情報を持つ
・処理やデータの構造が Code/Data Segment に閉じているので実行時間やメモリ使用量などを予測可能なものにする



Gears OS の構成

・Context
接続可能な Code/Data Gear のリスト、TaskQueue へのポインタ、Persistent Data Tree へのポインタ、独立したメモリ空間を持つ
Context は Thread ごとに用意する
・TaskQueue
ActiveTaskQueue と WaitTaskQueue を持つ
依存関係がある Task は WaitTaskQueue に挿入され、依存関係が解決されると実行可能な状態となり ActiveTaskQueue に移される
Compare and Swap 命令を用いることで平行に操作してもデータの一貫性を保つ
複数の Context で共有される
・Persistent Data Tree
Data Gear の管理を行う
非破壊木構造で構築されるので平行して読み書き可能
挿入・削除・検索における計算量を保証するため Red-Black Tree アルゴリズムを用いて平衡性を保つ
複数の Context で共有される
Persistent Data Tree への書き込みのみで相互作用を発生させ目的の処理を達成する
・TaskManager
Worker の起動・停止、Persistent Data Tree を監視して Task の依存関係を解決する
Task の依存関係は Input/Output Data Gear から自動的に決定する
・Worker
TaskQueue から Task を取得し実行する
Task に必要な Data Gear は Persistent Data Tree から取得する



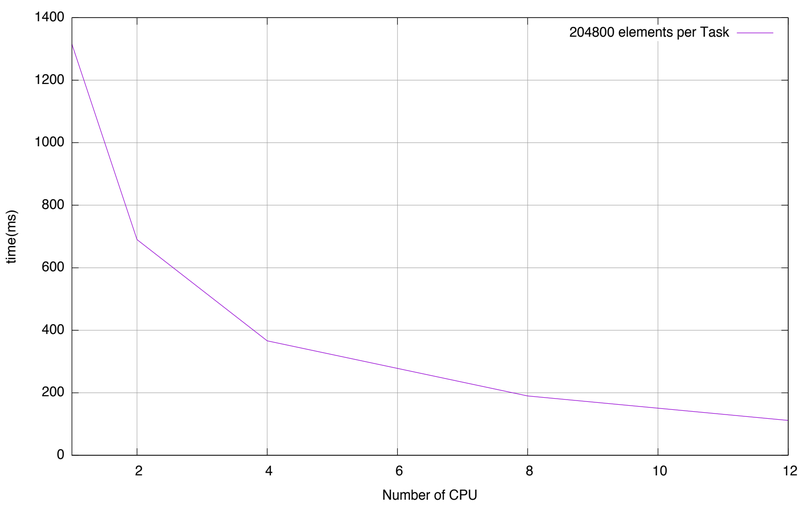
Gears OS の評価

与えられた整数配列を2倍にする Twice という例題を用いて Gears OS を評価する

・配列のサイズを元に処理の範囲と量を決める index, alignment, 配列へのポインタを持つ Data Gear に分割
・生成した Data Gear を Persistent Data Tree に挿入
・実行する Code Gear(Twice) と実行に必要な Data Gear への key を持つ Task を生成
・生成した Task を TaskQueue に挿入
・Worker の起動
・Worker が TaskQueue から Task を取得
・取得した Task に設定されている key を用いて Persistent Data Tree から必要な Data Gear を取得
・Code Gear(Twice) の実行、完了したら Task の取得に戻る

・要素数：2^17 x 1000 elements
・分割数：640 tasks
・1 Task 当たりの処理量：2^11 x 100 elements

・1 CPU と 12 CPU で約11.8倍の速度向上
・台数効果による十分な性能向上を確認した



今後の課題

・複雑な並列処理を実行できるようにするために依存関係を解決する TaskManager を実装する必要がある
・GPU などの各プロセッサのアーキテクチャにマッピングした実行機構の実装