Code Gear、 Data Gear に基づく Gears OS の設計

125716B 氏名 伊波立樹 指導教員：河野 真治

1 先行研究

本研究室では並列プログラミングフレームワーク Cerium[1] と分散ネットフレームワーク Alice[2] の開発を行なってきた。

Cerium では Task と呼ばれる分割されたプログラムを依存関係に沿って実行することで並列実行を実現する。依存関係はプログラマ自身が意識して記述する必要があり、Task の種類が増えると記述が複雑になり、 負担が大きくなる。Task の依存関係がデータの依存関係を正しく保証しない場合があるという問題がある。また、Task の取り扱うデータに型情報がないため、 汎用ポインタをキャストして利用するしかなく、型の検査が行われていない。Cerium は C++ で実装されているが、オブジェクトと並列処理が直接対応していないのでオブジェクト指向で記述する利点が少ない。

Alice では処理の単位である Code Segment、 データの単位である Data Segment を用いてプログラムを記述 [3] する。Code Segment は使用する Input Data Segment, Output Data Segment を指定することで処理とデータの依存関係を解決する。Alice は Java で実装されており、実効速度が遅いという問題がある。また、 Data Segment をアクセスする API のシンタックスが特殊なため、Alice を用いてプログラムを作成するには慣れが必要になる。

2 Gears OS

Cerium と Alice を開発して得られた知見から、並列実行をサポートするだけでなく、信頼性も確保した Gears OS の設計・開発を行う。

Gears OS では Gear という単位を用いてプログラムを Code Gear, Data Gear に細かく分割する。Code Gear は Input Data Gear から Output Data Gear を生成する。Input と Output の関係から Code Gear 同士の依存関係を解決し、並列実行を行う。

Cerium は初め、Cell[4] 向けのフレームワークとして設計されたという経緯からプロセッサ毎の実行形式が異なる。Gears OS では Many Core CPU、GPU をはじめとする様々なプロセッサを同等な実行機構でサポートする。

従来の OS が行う排他制御、メモリ管理、並列実行などは Meta Computation に相当する。Meta Computation は本論の Computation を支える Computation のことである。関数型言語では Meta Computation に Monad を用いる手法 [5] がある。Gears OS では、Meta Code/Data Gear を Monad として定義し、Meta Computation を実現する。

Gears OS は並列実行をサポートするだけでなく、 信頼性も確保する。そのために Gears OS を用いて作成されたプログラムに対する Model Checking を行う機能 [6] を提供する。並列プログラムに Model Checking を行うことでそのプログラムがとり得る状態を列挙する。これにより、並列実行時のデッドロックの検出などを行うことでプログラムの信頼性を確保する。Model Checking の実現には Meta Code/Data Gear を用いる。

Gears OS は Many Core CPU, GPU といった並列実行環境に合わせた設計・実装を行う。また、接続する Gear を変更することでプログラムの振る舞いを変更することを可能にする柔軟性、Monad に基づくメタ計算による並行制御、Model Checking を用いた信頼性の確保を目的とする。

3 Continuation based C

Gears OS は本研究室で開発している CbC(Continutaion based C)[7] を用いて実装を行う。

CbC のプログラムでは C の関数の代わりに Code Segment を用いて処理を記述している。Code Segment は C の関数と異なり戻り値を持たない。Code Segment の宣言は C の関数の構文と同様に行い、型に __code を使うことで宣言できる。

Code Segment から Code Segment への移動は goto の後に Code Segment 名と引数を並べて記述するという CbC 独自の構文で行う。この goto による処理の遷移を継続と呼ぶ。図 1 は Code Segment 間の継続関係を表したものである。

C では関数呼び出しを繰り返し行う場合、呼びだされた関数の引数の数だけスタックに値が積まれていくが、戻り値を持たない Code Segment ではスタックに値を積んでいく必要が無くスタックを変更する必要が無い。このようなスタックに値を積まない継続、つまり呼び出し元の環境を持たない継続を軽量継続と呼び、軽量継続による並列化、ループ制御、関数コールとスタックの意識した最適化がソースコードで行えるようになる。

4 Code Gear と Data Gear

Gears OS ではプログラムの実行単位として様々な Gear を使う。Gear が平行実行の単位、データ分割、Gear 間の

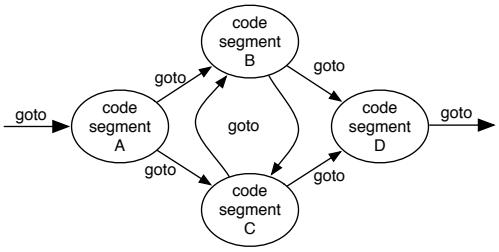

図 1: goto による Code Segment 間の継続

接続などになる。

Code Gear はプログラムの実行コードそのものであり、OpenCL[8]/CUDA[9] の kernel に相当し、 CbC では Code Segment で記述を行う。

Code Gear は処理の基本として、 Input Data Gear を参照し、一つまたは複数の Output Data Gear に書き込む。また、接続された Data Gear 以外には参照を行わない。

Code Gear は function call ではないので、呼び出し元に戻る概念はない。その代わりに、次に実行する Code Gear を指定する機能 (軽量継続) を持つ。

Data Gear には、int や文字列などの Primitive Data Type が入る。

Gear の特徴の一つはその処理が Code Gear, Data Gear に閉じていることにある。これにより、Code Gear の実行時間、メモリ使用量を予測可能なものにする。

Code Gear, Data Gear はポインタを直接には扱わない。これにより、Code と Data の分離性を上げて、ポインタ関連のセキュリティフローを防止する。

Code Gear, Data Gear はそれぞれ関係を持っている。例えば、ある Code Gear の次に実行される Code Gear、全体で木構造を持つ Data Gear などである。Gear の関連付けは Meta Gear を通して行う。Meta Gear は、いままでの OS におけるライブラリや内部のデータ構造に相当する。なので、Meta Gear は Code Gear, Data Gear へのポインタを持っている。

5 Context

ある Code Gear から継続するときには、次に実行する Code Gear を名前で指定する。Meta Code Gear が名前を解釈して、処理を対応する Code Gear に引き渡す。これらは、従来の OS の Dynamic Loading Library や Command 呼び出しに対応する。名前と Code Gear へのポインタの対応は Meta Data Gear に格納される。この Meta Data Gear を Context と呼ぶことにする。これは従来の OS の Process や Thread に対応する。

Context には以下のようなものが格納される。

- Code Gear の名前とポインタの対応表
- Data Gear の Allocation 用の情報
- Code Gear が参照する Data Gear へのポインタ
- Data Gear に格納される Data Type の情報

6 List

通常 List は要素と次へのポインタを持つ構造体で表現される。Gears OS の場合、Meta レベル以外でポインタは扱わないので図 2 のように任意の要素を持つ Data Gear と次へのポインタを持つ Meta Data Gear の組によって List は表現される。

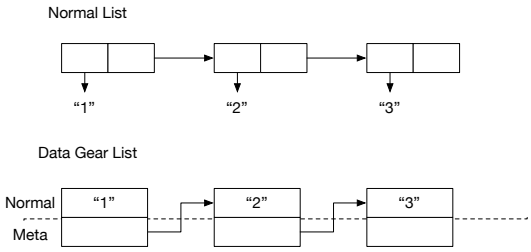

図 2: List の表現

7 Synchronized Queue

Gears OS では List を表現する Code/Data Gear に CAS(Compare and Swap) を行う Meta Code/Data Gear を接続することで Synchronized Queue を実現する。Gears OS の機能は状態遷移図とクラスダイアグラムを組み合わせた図で表現する。この図を GearBox と呼ぶことにする。図 3 は Synchronized Queue の GearBox である。M:get/put が CAS を行う Meta Code Gear となる。今回は CAS で実装しているが、接続する Meta Code Gear を変更することで通常の Queue や Lock を使用した synchronizedQueue など仕様変更にも対応できる。

8 今後の課題

Gears OS の今後の目的は Cerium と同等の例題を動かすことである。現在、必要な機能として Context、 Allocator、 List、 Non-Destructive Red-Black Tree、 Synchronized Queue を CbC を用いて実装しており、足りない機能としては Worker がある。Worker の実装後、動かす例題としては Bitonic Sort、 Word Count を予定している。例題の実装後、測定・評価を行う。

参考文献

[1] 宮國　渡，河野真治，神里　晃，杉山千秋：Cell 用の Fine-grain Task Manager の実装，情報処理学会シス

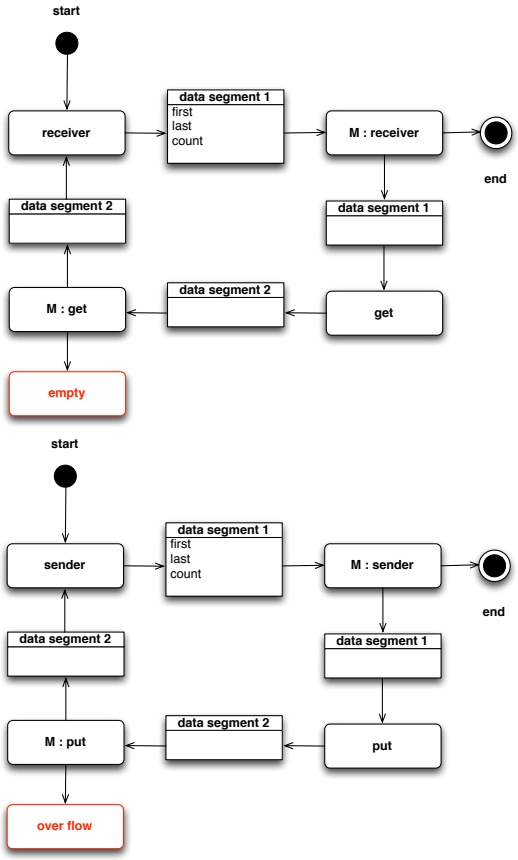

図 3: Synchronized Queue

テムソフトウェアとオペレーティング・システム研究会 (OS) (2008).

[2] 赤嶺一樹，河野真治：DataSegment API を用いた分散フレームワークの設計，日本ソフトウェア科学会第 28 回大会論文集 (2011).

[3] 河野真治，杉本　優：Code Segment と Data Segment によるプログラミング手法，第 54 回プログラミング・シンポジウム (2013).

[4] Sony Corporation: Cell broadband engine architecture (2005).

[5] Eugenio Moggi, Notion of Computation and Monads(1991)

[6] 下地篤樹，河野真治：線形時相論理による Continuation based C プログラムの検証，情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS) (2007).

[7] 徳森海斗，河野真治：Continuation based C の LLVM/clang 3.5 上の実装について，情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS) (2014).

[8] Aaftab Munshi, Khronos OpenCL Working Group: *The OpenCL Specification Version 1.0* (2007).

[9] : CUDA, https://developer.nvidia.com/category/zone/cuda-zone/.