# Code Gear、 Data Gear に基づく OS の設計

125716B 氏名 伊波立樹 指導教員：河野 真治

## 1 Gears OS

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

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。

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。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。これは Alice の Code Segment、 Data Segment にそれぞれ対応する。

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 [4] がある。Gears OS では、Meta Code Gear, Meta Data Gear を Monad として定義し、Meta Computation を実現する。

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を用いた信頼性の確保を目的とする。

今回基本的な設計と Gears で Cerium 同等の機能を実装するため、並列で Data や Task を振り分けるための Synchronized Queue を実装した。

## 2 Continuation based C

Gears OS は本研究室で開発している CbC(Continuation based C)[5] を用いて実装を行う。CbC は処理を Code Segment で記述することを基本としているため、 Gears OS の Code Gear を記述するのに適している。

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ている。

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。

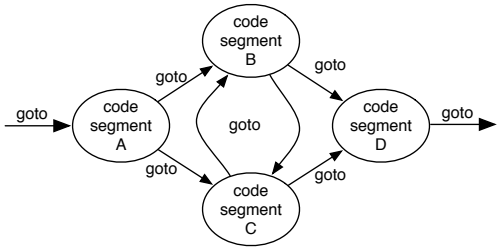


図 1: goto による Code Segment 間の継続

## 3 Code Gear と Data Gear

Code Gear はプログラムの実行コードそのものであり、OpenCL[6]/CUDA[7] の kernel に相当する。

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。これは Alice の Input Data Segment と Output Data Segment の関係に対応しており、依存関係を解決し、 Code Segment の並列実行を可能とする。

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。

## 4 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

## 5 List

List は Synchronized Queue を実装するために必要なデータ構造である。通常の List は要素と次へのポインタをもつ構造体で表現される。Gears OS の場合は Meta レベルでポインタを扱うため、 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。

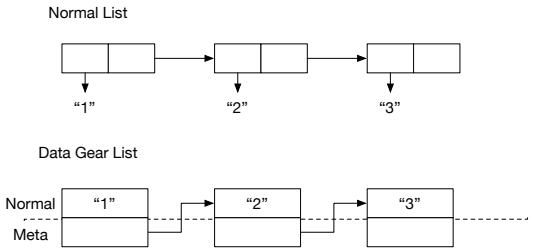


図 2: List の表現

## 6 Synchronized Queue

Synchronized Queue は Gear OS 内の Task を配るために必要なデータ構造である。Gears OS では List を表現する Code Gear, Data Gear に CAS(Compare and Swap) を行う Meta Code Gear, Meta 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 Get 処理の GearBox である。M:get が CAS でデータの取り出しの際の書き換えを行う Meta Code Gear となる。Put 処理を行う場合はこの Get 処理の流れはそのままに Put の処理を記述することになる。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 Mutex を使用した synchronizedQueue など仕様変更にも対応できる。

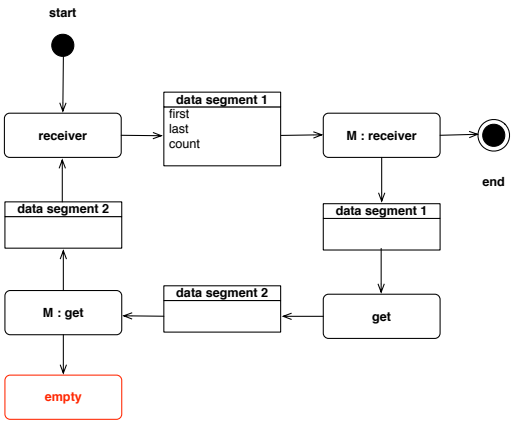


図 3: Synchronized Queue

## 7 今後の課題

今回、Gears OS の基本的な設計と必要な機能の一部を CbC を用いて実装した。今後の課題として Worker を実装する予定である。これにより, Cerium と同等の課題を動かす事が可能となる。動かす例題としては Bitonic Sort, Word Count を予定している。例題を実装後、 Gears OS の測定・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] Eugenio Moggi, Notion of Computation and Monads(1991)

[5] 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).

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

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