CbC 言語による OS 記述

135756F 氏名 宮城光希 指導教員：河野 真治

Abstract

We are developping Gears OS using Continuation based C (CbC). Gears OS provids highly reliable computation using meta computation. CbC gives Code Gear and Data Gear as programing units. A transfar from a Code Gear to another Code Gear is implemented using a CbC's goto statement, which is compiled as a jump instraction in CbC. Meta computations are key components of Gears OS, which provids memory managements, thread managements, managements of Data/Code Gear themselves. CbC's goto statments provids a ways of implementing meta computations. From a view point of meta computation, Data Gear or Code Gear are uniform data units, which are implemented as union Data in CbC. In the meta level, a transfar from a Code Gear is a goto statement to meta Code Gear with next Code Gear number and a Context which coresponds thread structure or an enviroments in a functional programing. A meta Code Gear handles meta computations such as meta computations. From a nomal level, meta structures are not visible directly and a Code Gear looks like a function using continuations. A stub Code Gear is used as a bridge between meta level and nomal level. In this paper we create scripts which generate meta Code Gear and stub Code Gear from nomal level Code Gear and Data Gear. Using these scripts, we can provide a interface mechanisms which are packages of Code Gears and Data Gears. A simple TaskManager is constracted using the interfaces which is a simple operating systems. We will constracts various compornents of Gears OS and meta computations which provids reliabilty. For an example, generating agda program from nomal level Code Gear provids proof suports of the Code Gear.

1 メタ計算の重要性

プログラムを記述する際、通常の処理の他に、メモリ管理、スレッドの待ち合わせやネットワークの管理、エラーハンドリング等、記述しなければならない処理が存在する。これらの計算を Meta Computation と呼ぶ。

Meta Computation を通常の計算から切り離して記述するためには処理を細かく分割する必要がある。しかし、関数やクラスなどの単位は容易に分割できない。

そこで当研究室では Meta Computation を柔軟に記述するためのプログラミング言語の単位として Code Gear、Data Gear という単位を提案している。

Code Gear は関数に比べて細かく分割されているので Meta Computation をより柔軟に記述できる。Code Gear、Data Gear にはそれぞれメタレベルの単位である Meta Code Gear、Meta Data Gear が存在し、これらを用いて Meta Computation を実現する。

Continuation based C (CbC) はこの Code Gear 単位を用いたプログラミング言語として開発している。

CbC は軽量継続による遷移を行うので、継続前の Code Gear に戻ることはなく、状態遷移ベースのプログラミングに適している。

また、当研究室で開発している Gears OS は Code Gear、Data Gear の単位を用いて開発されており、CbC で記述されている。

本研究では CbC を用いての Gears OS の実装と CbC における ユーザーの関知しない Meta Computation の自動生成を行なう。

2 Continuation based C (CbC)

CbC は 処理を Code Gear とした単位を用いて記述するプログラミング言語である。Code Gear は入力と出力を持ち、CbC では引数が入出力となっている。Code Gear から次の Code Gear へと goto による継続で遷移で処理を行い、引数として出力を与える。図 1 は Code Gear 間の処理の流れを表している。

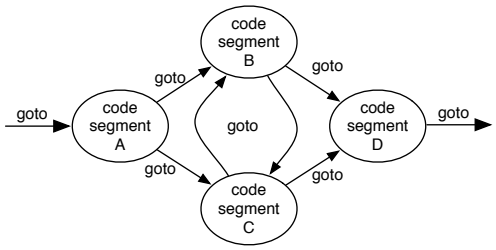

図 1: goto による code gear 間の継続

3 Code Gear

4 Gears OS

Gears OS では並列実行するための Task を、実行する Code Gear 、実行に必要な Input Data Gear 、Output Data Gear の組で表現する。Data Gear はデータの単位であり、int や文字列などの Primitive Type を持っている。Code Gear は 任意の数の Input Data Gear を参照して処理を行い、Output Data Gear を出力し処理を終える。また、接続

された Data Gear 以外には参照を行わない。処理やデータの構造が Code Gear、Data Gear に閉じているため、これにより実行時間、メモリ使用量などを予測可能なものにすることが可能になる。

Gears OS では Meta Computation を Meta Code Gear、Meta Data Gear で表現する。Meta Code Gear は通常の Code Gear の直後に遷移され、Meta Computation を実行する。

CbC は Code Gear を処理の単位として用いたプログラミング言語であるため、Gears OS の Code Gear を記述するのに適している。

図 2 に Gears OS の構成図を示す。

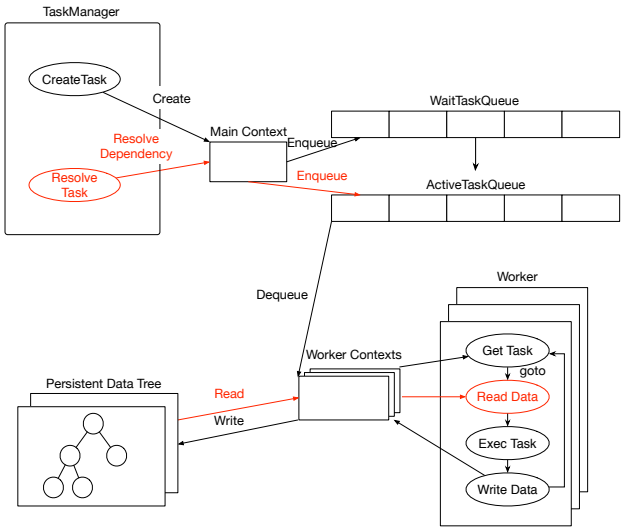

図 2: Gears OS の構成図

5 Context

Gears OS では Context と呼ばれる接続可能な Code/Data Gear のリスト、Temporal Data Gear のためのメモリ空間等を持っている Meta Data Gear である。Gears OS は必要な Code/Data Gear に参照したい場合、この Context を通す必要がある。メインとなる Context と Worker 用の Context がある。Temporal Data Gear のためのメモリ空間は Context 毎に異なり、互いに干渉することはできない。

Context は Task でもあり、TaskManager によって Context が生成され CPUWorker へ送られる。Worker に渡された Task である Context の Input/Output Data Gear の依存関係が解決されたものから並列実行される。

6 interface の記述

interface を記述することで Context から Code Gear が呼び出せるようになった。create は関数呼び出しで呼び出され、interface と impliment の初期化と Code Gear のポインタの設定を行う。return で interface を返し、その先で interface で指定した Code Gear へ継続できるようになった。

7 Gearef、GearImpl

Context には Allocation 等で生成した Data Gear へのポインタが格納されている。Code Gear が Context にアクセスする際、ポインタを使用してデータを取り出すためコードが煩雑になってしまう。そこで Code Gear がデータを参照するための Gearef というマクロを定義した。Gearef に Context と型を渡すことでデータの参照が行える。また impliment を参照する際も、ポインタでの記述が複雑になってしまうため 同様に GearImpl を定義した。GearImpl は Context と interface 名、interface の変数名を指定して参照する。

8 stub Code Gear

Code Gear が必要とする Data Gear を取り出す際に Context を通す必要がある。しかし、Context を直接扱うのはセキュリティ上好ましくない。そこで Context から必要なデータを取り出して Code Gear に接続する stub Code Gear を定義し、これを介して間接的に必要な Data Gear にアクセスする。stub Code Gear は Code Gear 毎に生成され、次の Code Gear へと継続する間に挟まれる。

9 Context, stub の自動生成

Gears OS では通常の Computation の他に Context や stub などの Meta Computation を記述する必要がある。Gears OS を現在の CbC の機能のみを用いて記述すると Context や stub Code Gear の記述を行わなくてはならず、これには多くの労力を要する。そのため、この記述を助けるために Context を生成する generate_context と stub Code Gear を生成する generate_stub を perl スクリプトで作成した。

10 まとめ

参考文献

[1] TightVNC Software. http://www.tightvnc.com.