CSPを用いた並列処理設計とoccam実装例の紹介

首都大学東京
理工学研究科
福永研究室

目次

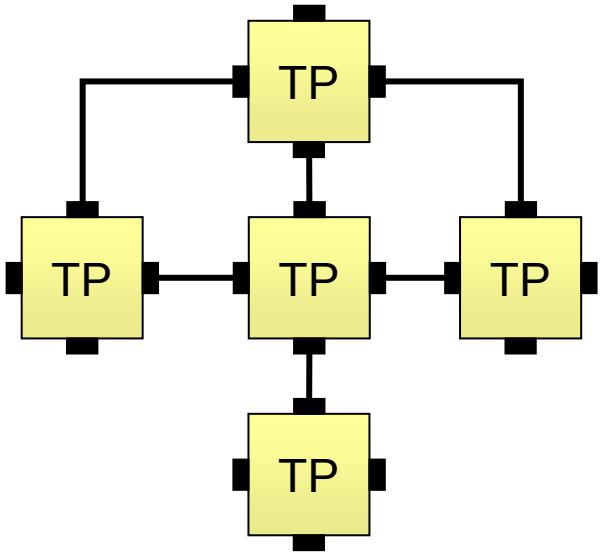
- はじめに
- 並列処理プロセッサTPCORE
- TPCOREによる分散ワーカモデル
- CSPによる記述
- FDR2による検証
- マンデルブロ集合計算例
- 結果と今後

はじめに

- 今回の目的
  - 簡単な例を用いてCSP記述、FDR2検証から実装までの流れを知る。
  - 教育用として今後、学ぶ人の役に立つよう作成した。

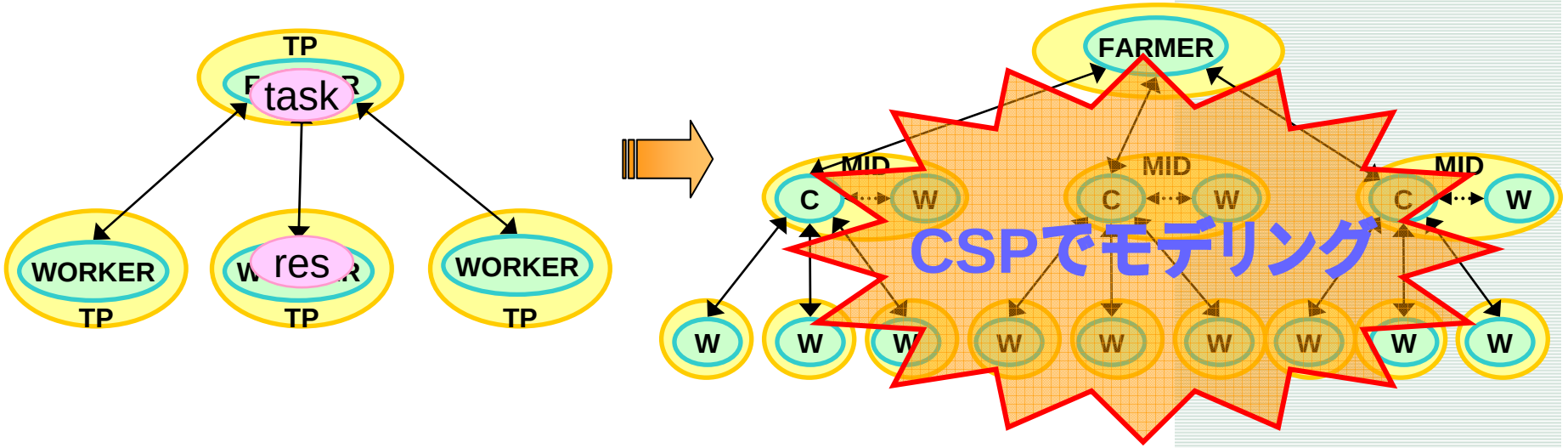
並列処理プロセッサTPCORE

- **当研究室で開発されたプロセッサ**
- **英国Inmos社のTransputerと アセンブリレベルで互換**
- **occamプログラムを実行可能**



TPCOREによる分散ワーカモデル

- **外部リンクは4本。**
- **そのままWORKER数を増やせない。**

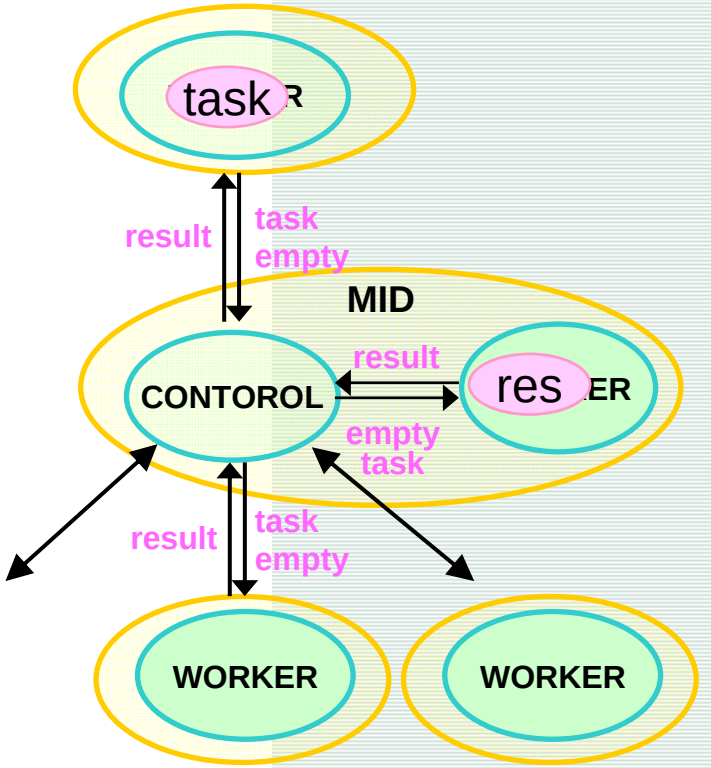


- **木結合により拡張。**
- **中間のTPCOREはコントローラを持つ。**

CSPによる記述(WORKER)

- workerはまず、result (ダミー) を送信し、最初のtaskをもらう。
- 受け取ったtaskに対して計算し、その結果を返す。
- emptyが来れば正常終了する。

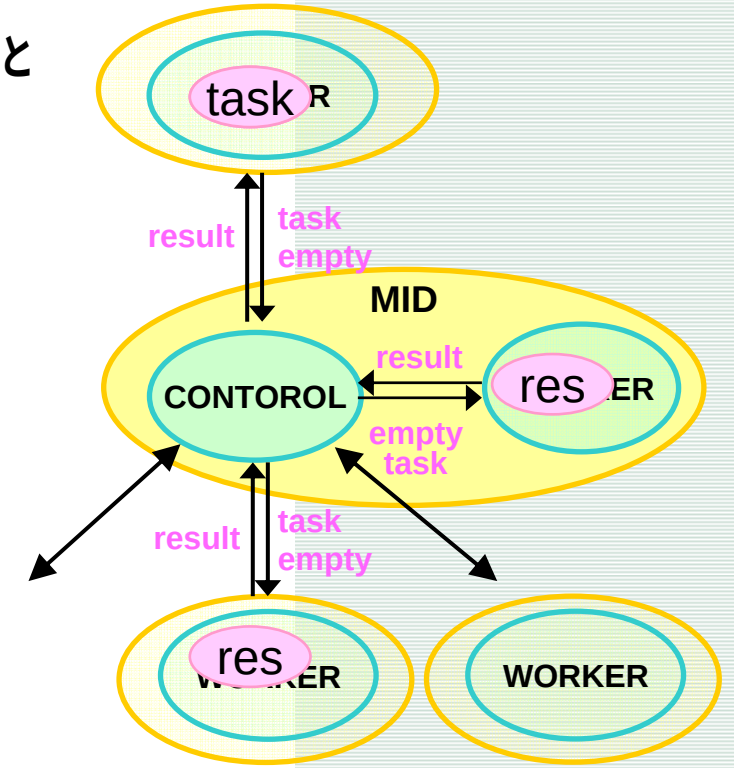
```
WORKER(in,out) =
  let
    MAIN =
      in?data ->
      if data==task then
            out!work(task) -> MAIN
      else --empty
            SKIP
    work(task) = res
  within out!res -> MAIN
```



CSPによる記述(MID)

- CONTROL
  - Workerからの入力（外部チャネルfromWorkと内部チャネルleft）を外部選択とする。
  - 入力のあったチャネルに対してfamerとのデータ送受信の仲介をする
- MID は CONTORLとWORKERからなる。

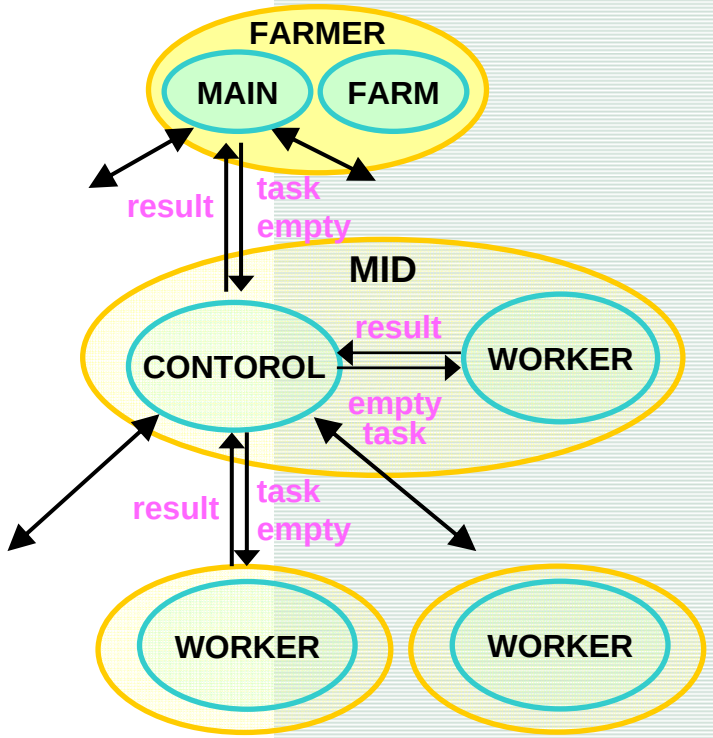
```
MID(toFarm,fromFarm,toWork,fromWork) =
 let
   CONTROL =
     [ ] i:Tag@ fromWork.i?res -> toFarm!res ->
           fromFarm?data -> toWork.i!data -> CONTROL
     [ ] left?res -> toFarm!res ->
           fromFarm?data -> right!data -> CONTROL
 within  CONTROL [|{|right,left|}|] WORKER(right,left)
```



CSPによる記述(FARMER)

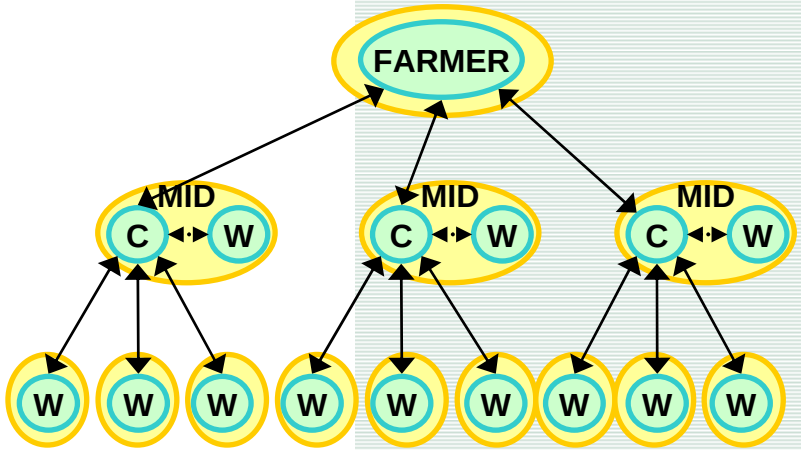
- FARMERはMAINとFARMからなる。
- FARMは次の仕事をMAINに渡す。
- MAINはtaskをMIDに割り振る。

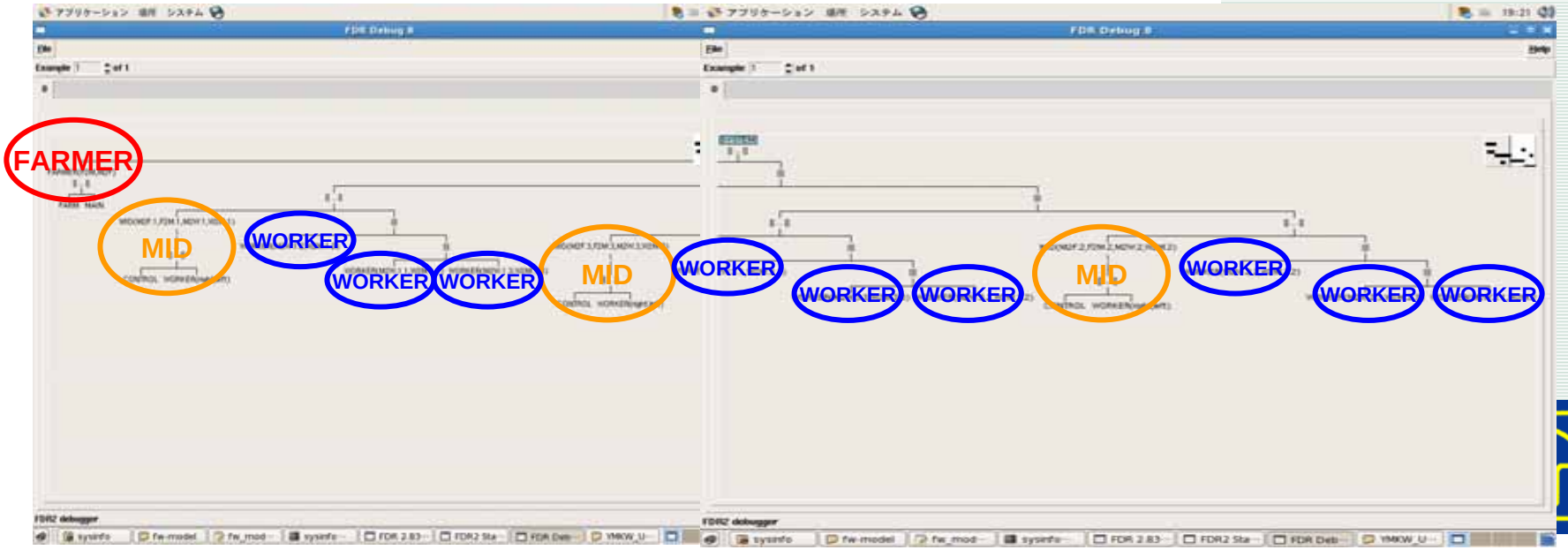
```
FARMER(toMid,fromMid) =
  let
    FARM = (;i:<1..TaskNum>@ inner!task->SKIP)
                    ; FARM --inner!empty->SKIP
    MAIN =
      inner?data ->
        if data==task then
          []i:Tag@ (fromMid.i?res ->
                          toMid.i!task -> MAIN)
        else -- data==empty
          ;j:<1..NumOfWorker>@ ([]i:Tag@
            fromMid.i?res -> toMid.i!empty -> SKIP)
  within FARM [|{|inner|}|] MAIN
```



FDR2による検証(デバック画面)

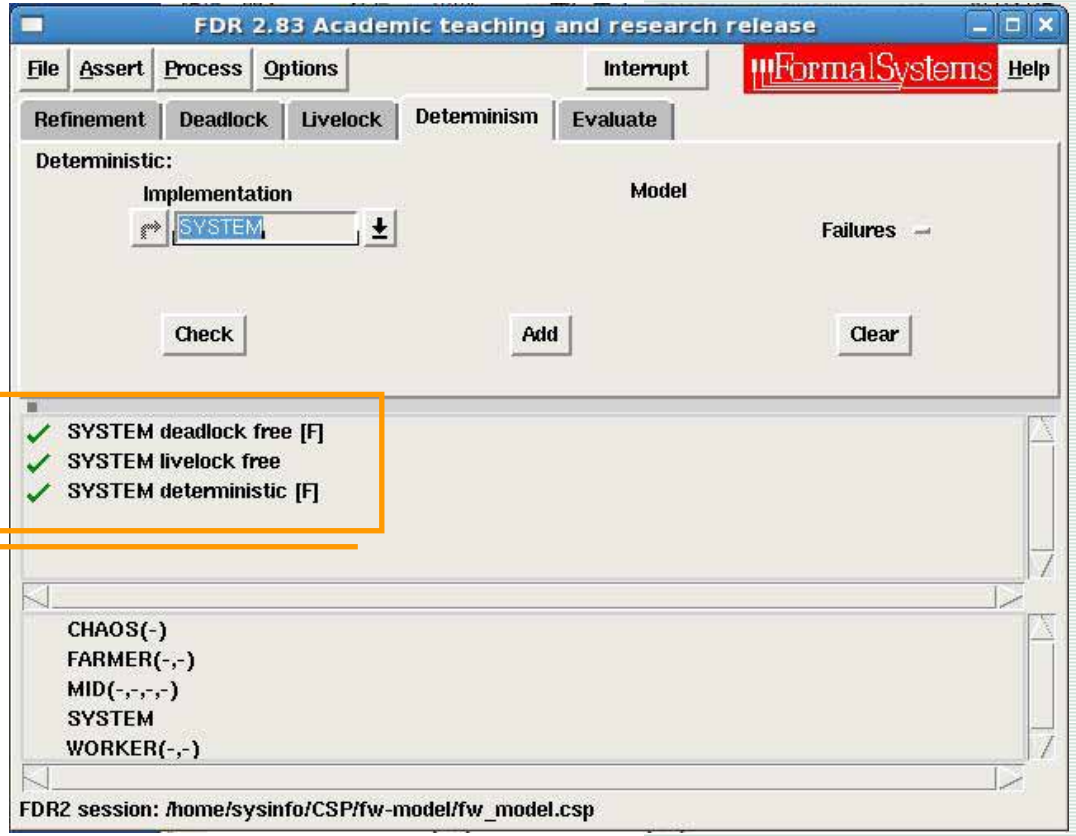
```
SYSTEM =
    FARMER(F2M,M2F)
    [|{|F2M,M2F|}|]
    |||i:Tag@
       ( MID(M2F.i,F2M.i,M2W.i,W2M.i)
         [|{|W2M,M2W|}|]
         |||j:Tag@ WORKER(M2W.i.j,W2M.i.j) )
```





FDR2による検証（検証画面）

- **正当性の確認**
  - Deadlock free
  - Livelock free
  - Deterministic



マンデルブロー集合計算例

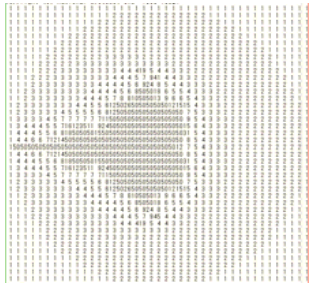
TPCOREによる分散ワーカモデルで
マンデルブロ集合を求める。

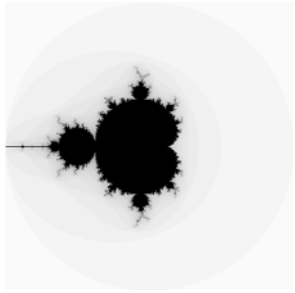
マンデルブロ集合とは、次の漸化式

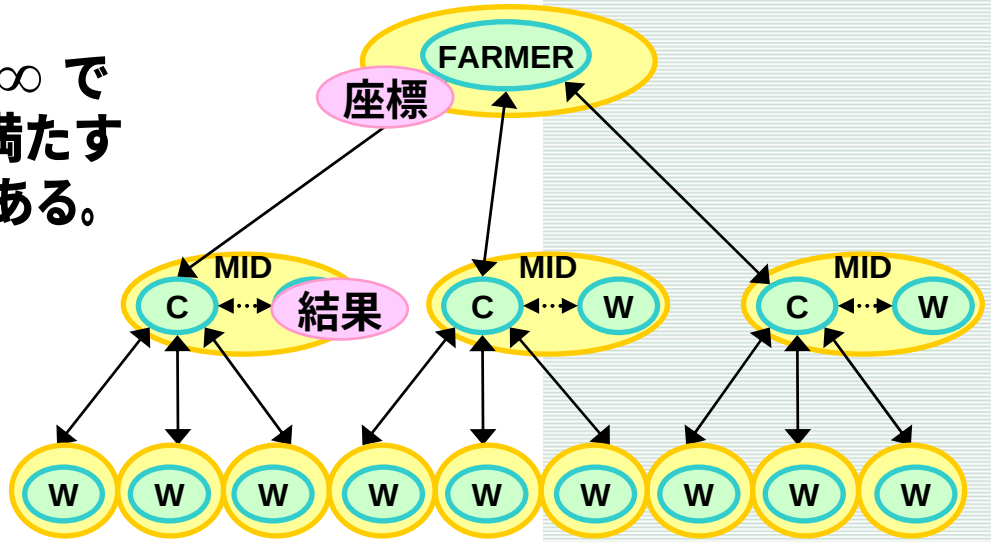
$$Z_{n+1} = Z_n^{\,2} + c$$

$$Z_0 = 0$$

で定義される複素数列$\{Z_n\}$が$n \to \infty$ で
無限大に発散しない という条件を満たす
複素数 c 全体が作る集合のことである。







- 各WORKERには座標を与える。
- WORKERは座標の計算結果を返す。

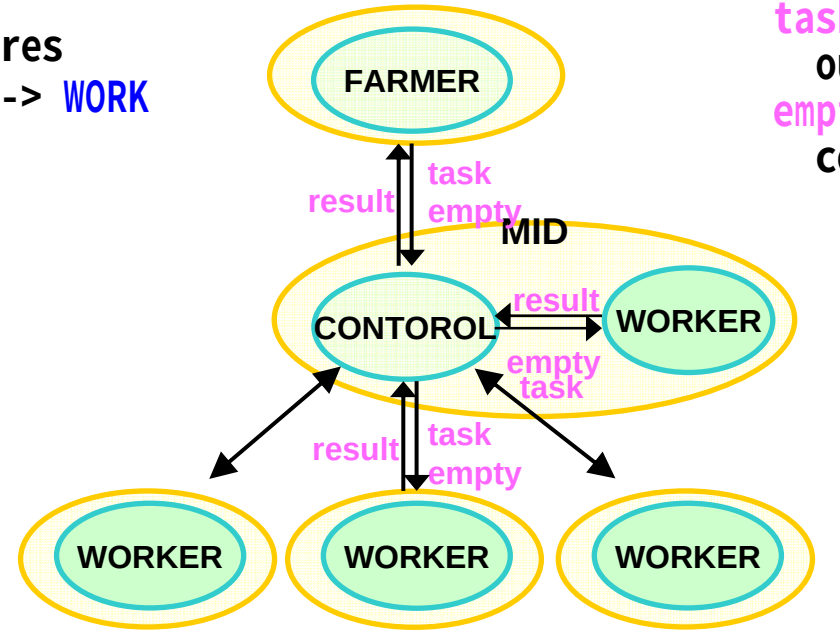
マンデルブロー集合計算例(WORKER)

CSP記述

```
WORKER(in,out) =
  let
    WORK = in?data ->
      if data==task then
           out!work(task) -> WORK
      else --empty
           SKIP
    work(task) = res
  within out!res -> WORK
```

occamプログラム

```
PROC WORKER(in, out)
  SEQ
    cond := TRUE
    out! 0
    WHILE cond
      SEQ
        in? CASE
          task; x;y
            out!work(x,y)
          empty;
            cond := FALSE
```



* 紙面の都合上
プロトコル定義
チャネル宣言、
変数宣言等省略

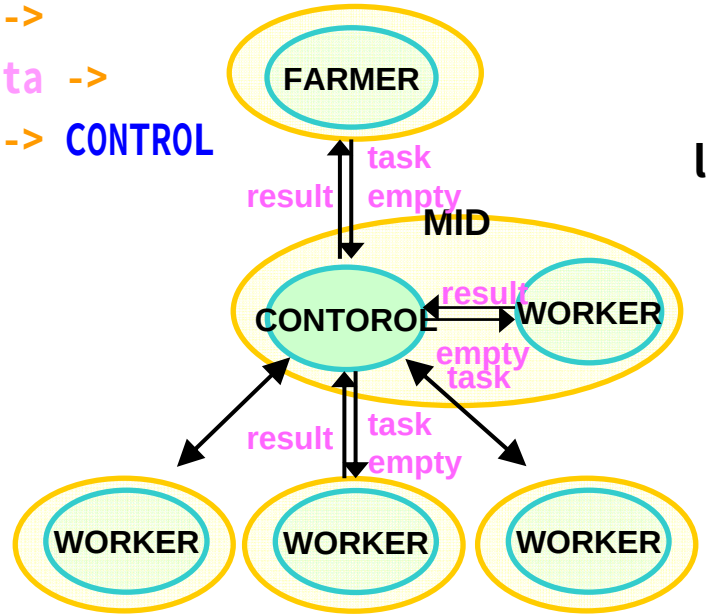
マンデルブロー集合計算例(CONTROL)

CSP記述

```
CONTROL =
  [] i:Tag@ fromWork.i?res ->
            toFarm!res ->
            fromFarm?data ->
            toWork.i!data -> CONTROL
  [] left?res ->
        toFarm!res ->
        fromFarm?data ->
        right!data -> CONTROL
```

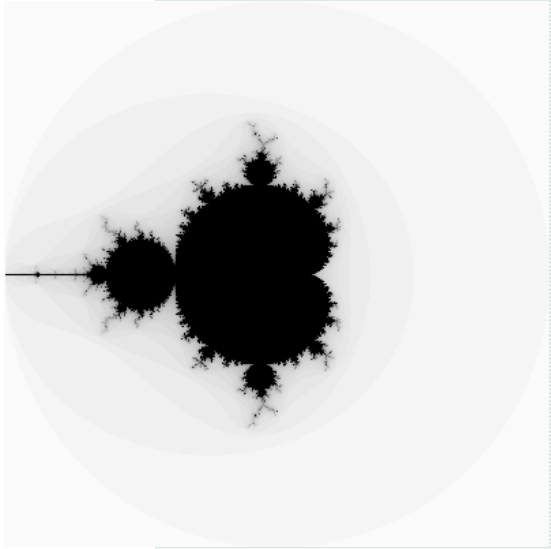
occamプログラム

```
PROC CONTORL
  WHILE TRUE
    SEQ
      ALT
        ALT i=0 FOR 3
          fromWork[i]? res
            SEQ
              toFarm! res
              fromFarm? x;y
              toWork[i]! x;y
        left? res
          SEQ
            toFarm! res
            fromFarm? x;y
            right! x;y
```



結果と今後

- CSP記述からの実装により
  正しく計算結果を返している。
- workerにindexを付け、全ての
  workerが動作していることも確認した。
- CSPの有用性が確認できた。
  - 並行性特有の問題が軽減
  - テスト工程の短縮



実行結果

- 現在、研究室で作成しているルータの使用に
  よりJCSPのようにあらゆる接続形態が可能
- より複雑なシステムに対して
  CSP、TPCOREを適用