Erlangの
並列計算の入り口

2007/OCT/23

2010/FEB/16

たけおか

Erlanはコミッティド・チョイス言語だ

2010/FEB/16追記

- コミッティド・チョイス言語
  - 節の頭にガードがある
  - ガードを超えた節だけが、選ばれて走行する
  - Erlanでは、ガード部分が、パターン・マッチングになっている

- 多くのプロセスが、チャンネル通信する
  - ErlangはOccam言語とそっくり

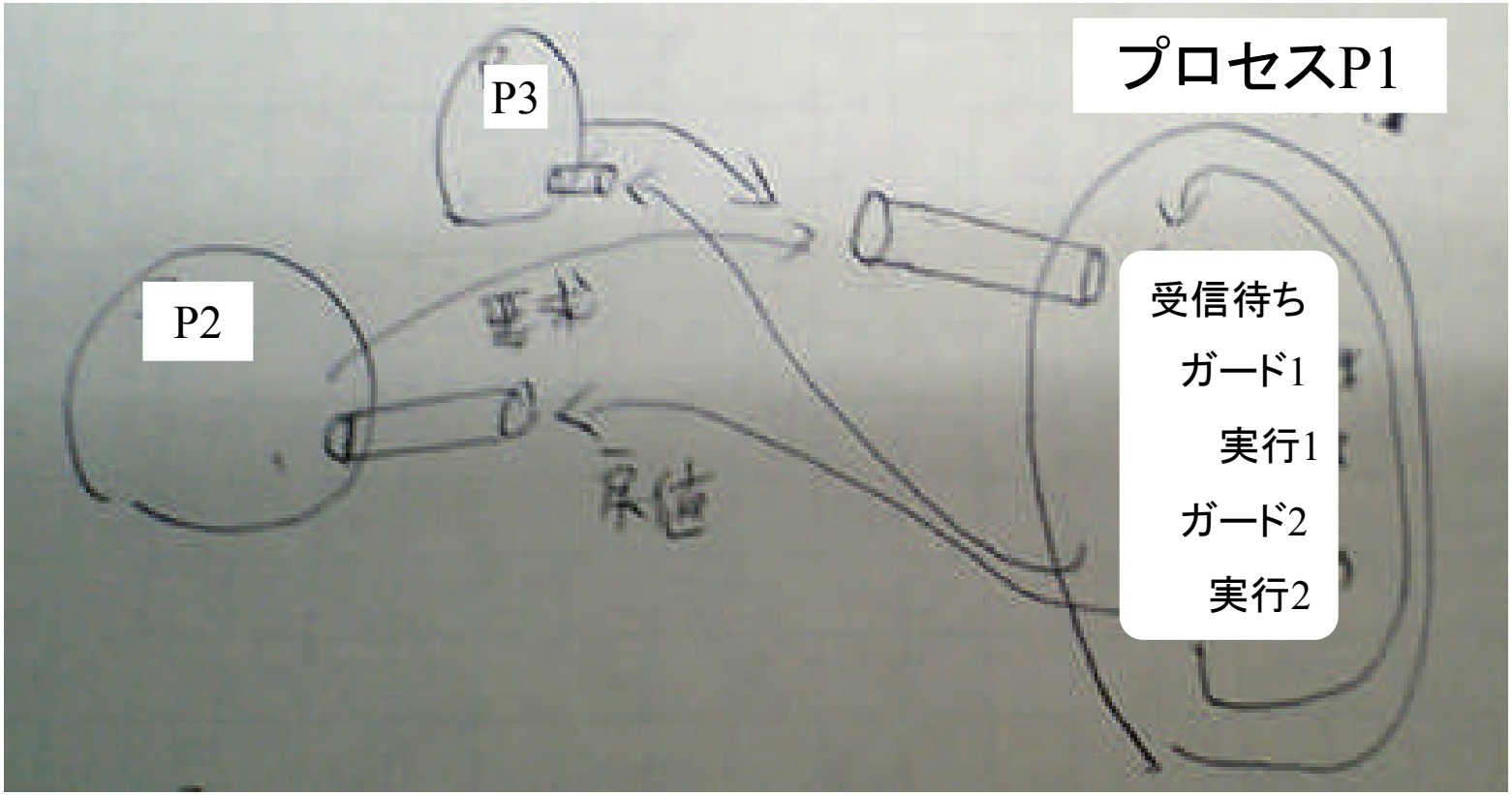
  - 詳しくは、中身を読もう

RPC

- RPC: Remote Procedure Call 遠隔手続き呼び出し
  - 遠隔にある手続きを、同期的に呼び出す(最近は非同期的なRPCもある)
  - 遠隔の手続きは、仕事が終わると返り値をもどす
- 同期的
  - 呼ばれた側の仕事が終わるまで、呼び出し側は止まる
  - バグが出にくい
  - 素朴な実装の場合、呼ばれる側の関数は、同時に複数 入ってこないため、簡単で良い。(再入可能性の検討など不要)
- 非同期的
  - 呼ばれた側は、呼び出し側と無関係に仕事を進める
  - 終了の通知方法は?
  - 仕事の結果を置く場所は? 呼び出し側は正しくハンドリングするか

チャンネル通信

- CSP Communicating Sequential processes
  - Hoare(ホーア)が考えた
  - チャンネルで通信する、普通のプログラム
    - 排他制御の問題とか出ない
    - 基本の構想はRPCに近い
      - CSPは戻り値も、チャンネル通信で返さねばならないが
      - 非同期的な呼び出しも可能
    - デッドロックが起こりにくい。デッドロック発生の検出が容易
    - 最近のTCPコネクションごとにthreadを貼り付けるのも近い考え



シーケンシャルなプログラム(プロセス)の集まりで仕事をする。
主にRPCを用いて

CSP をもとにした現実システム

- Inmos社(英)
  - TransputerとOccamを実現して販売していた
  - イギリスの国策会社
  - ST Microに吸収された
- Occam
  - プログラミング言語
- Transputer
  - Occamと同時に考えられたハードウェア
  - CPUをトランジスタのごとく並べて使用。

Transputer

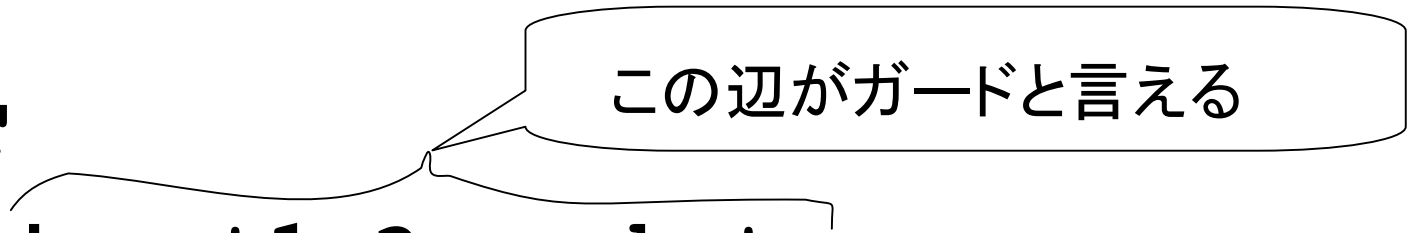
- CSP をもとにした現実システム
- Occamと同時に考えられたハードウェア
- CPUをトランジスタのごとく並べて使用
  - Transputerは、Transistor + Computerという造語
- 4～8本のシリアル通信ハードウェアを持つ。
  - 多数のCPUを2次元のメッシュor3次元以上のハイパーキューブ状に配置し、シリアル・リンクで近傍を接続。
  - 遠くへの通信は、近傍のCPUに順次リレーしてもらう
- Transputerは、CPUコア上にタスク(スレッド)をハードウェア・レベルで実装
  - jump命令を実行するたびに、スケジューリングとタスク・スイッチ
  - デフォールトのスケジューラはハードウェア。ラウンドロビン方式
  - ユーザがソフトウェアで書いたスケジューラに切り替えることも可能。
    - あるベクトルにユーザのスケジューラ・ルーチンの先頭のポインタを置くと、そのスケジューラ・ルーチンが呼び出される
- 全盛期は、FPUが高速でコストパフォーマンス良し。C言語でプログラミング
- ST-20はTransputerのこと

Occam

- CSP をもとにした現実システム
- プログラミング言語
  - 「Occamのかみそり」
    - 「不要なものはそぎ落とせ」から来た名前
- チャンネル通信
  - Communicating Sequential Processes
- Committed Choice言語
- ガードがある

- インデントに意味がある。区切り記号が少ない
  - 専用の構造化エディタでプログラムを書いた

Occamのプログラム断片例




```
ALT
    input1 ? packet
       output ! packet
    input2 ? packet
       output ! Packet
```

ALTがCominttedChoiceの指定プリミティブ

- 複数の候補のうち、ガード(上記では受信)を満たした、先着の一つだけが選ばれ、その実行部が実行される
- Occamのガードは通信しか書けない

Committed Choice

- ガードを設けて、ガードを超えたものが実行を開始する
- CSP/Occam
  - 通信もガード条件の一種
- GHC: Guarded Horn Clause
  - 上田さんが考えた。(当時NEC)
  - AND並列Prologの一種
  - Prologの節のコミット・バー‘!’までをガードとする。
    - GHCではコミットバーは、「ガード記号」と呼ばれる
  - チャンネル通信をしていると考えるべし
  - 通常はバックトラックしない

Erlang

- ST Microがハードウェア設計/検証のために作ったと言われている
- 変数を使用したチャンネル通信
- ガードがあり、ガードを超えた節が排他的に実行される
- パターンマッチングにユニフィケーションを使用
  - 節のヘッドでのパターンマッチングはprologのようだが、実行の意味は違う
- 型がない
  - 関数呼び出し時に、パターンマッチングする言語としてはMLが有名だが、MLは強い型の言語
- テレコム・アプリケーションを書いた実績あり

Erlangのプログラム例

```
rcv_messages() ->
        receive
            {foo, X} ->
                io:fwrite("foo~n"),
                ture;
            {bar, X} ->
                io:fwrite("bar~n"),
                ture;
        rcv_messages().
```

ここがガード

ガードは'->'の
直前まで
receiveの
受けの部分は、
単純変数,
定数、タプルなどの
パターンが書ける.
単純変数を書くと、
Occamなどの
チャンネルからの
受信変数になる

RPC指向の言語の排他制御

- 1つのシーケンシャル・プロセスに資源を持たせ、そのプロセスのみが資源をアクセスする
- いわゆるサーバ
- サーバに資源を持たせれば、排他制御の問題は起こらない
  - 何でもかんでも安全というわけではない
    - 例えば、サーバの中から別なサーバを呼び出すようなことをすると、デッドロックが起きるかも知れない
- 広い意味で、モニタを構成しているとも考えることができる
  - メッセージ・パッシングするオブジェクト指向風モニタ

用語

並行計算用語

- 並行と並列
- 並行 concurrent
  - 論理的に同時に計算が行われていると考えられる場合
  - プロセスは、理論的な言葉で、並行とともに用いられることが多い(UNIXは「タスク」というべきものを「プロセス」と名付けた)
- 並列 parallel, simultaneous
  - 実際に同時に複数のものが動作する
  - on the flyという場合もまれにある。(放っておいても動作するハードウェアに対していうことがほとんど)

並行計算用語

- 再入可能 reentrant
  - いくつものプロセスが同時に、呼び出し可能
  - thread safeとはすなわち、再入可能であるということ
  - どうやって実現するのか?

並行計算用語

- 排他制御
  - 複数のプロセスが同時に使用できない資源などを守る
  - OS的には、排他制御と待ちは組である
- 同期と排他は本質的には同じ。排他は同期の問題にも還元可能
- きわどい領域 critical section
  - 排他制御の必要な領域(主にプログラム・コード上)
- 不可分 atomic
  - 基本操作
  - マルチ・プロセス動作するとき、これ以上分割すると、不条理なことが起こる時、不可分であるという
  - 不可分操作 atomic operation
  - 機械語レベルであると、割り込みに邪魔されないということ(単一CPUの時)

並行計算用語

- デッドロック dead lock
  - 資源を取り損なって、システムが止まる
  - 排他制御の失敗による
  - 排他制御の必要な資源の設計のミスによる
    - 一般に、排他的な資源を複数押さえることが可能なシステムでは、起きる
  - 誰が資源を押さえているか、調べる

- ライブ・ロック live lock
  - 広義のデッドロックに含まれる
  - システムとしては生きているが、一部の資源がとれずに、一部の仕事が永遠に遅延(封鎖)される
  - 資源を押さえる手順がレーシング追いかけ状態に陥っている

代表的な排他制御/同期のプリミティブ

- セマフォ (semaphore)
  - セマフォが得られれば、資源の使用権を得られたと考える
  - オランダ人のDijkstra(ダイクストラ)の発案
  - P/V (Wait/Signal)命令が基本
    - Waitでセマフォを得る/Signalでセマフォを返す
    - 腕木信号の用語。しかもオランダ語
  - 1bitのバイナリ・セマフォ
  - カウンティング・セマフォ (カウンタが0でなければ資源を使用可能)
  - mutexはセマフォの一種
- モニタ (monitor)
  - きわどい領域を一つの手続きにし、そこに一人(または許された数)しか入れないように、システムが制御
  - Javaが採用
- ランデブー(rendezvous), バリア(barrier)
  - 全員がそろうまで待つ
- その他
  - 交換変数(Exchange)などもあるが、あまり知られていない。実用性もない
- 待ちには、spin lockなどがある

より低位な排他制御/同期のプリミティブ

- test and set 命令
  - メモリの内容をアトミックに書き換え、書き換え前の状態をテストする
    - セマフォを test and set命令で実現する
      - でも、それがそのまま、プロセスの待ちに使用できるか?
  - Read modified write命令である
    - load-storeアーキテクチャのRISCシステムにそんなものはない
    - CISCには、好んで付けられた
- LL/SC (Load and Link/Store Conditional)命令
  - (近代的な)キャッシュ・メモリの機能
  - 使用法

1) load linkを行って古い値を得る
2) 権利が得られそうか? 得られなければ、もっと違うレベルの待ちへ
3) 更新する値を作る
4) SCで新しい値を書き込む
5) SCの結果が失敗であれば、1へ

  - LLは、キャッシュにCPUコアからのアクセスがあったことを記憶。SCの実行前に、他のCPUコアでLLアクセスがあれば、SCが失敗する
  - キャッシュ同士が、LLアクセスがあったことを通信する

semaphore

- セマフォは、協調型
  - みんながセマフォ・アクセスを正しく行わねばならない
  - セマフォを見ない人がいたら、競合が起こる
  - セマフォを返さない人が居たら、資源は永久封鎖
  - デッドロック解析が難しい(と一般に思われている)
    - 普通の人は解析手法を持たない

monitor

- モニタは、出るときに権利を失うので、たちがいい
- 協調型ではないので、馬鹿な人が居ても、間違いは起きにくい
- 実行時間がかかりすぎるとか、永久封鎖とか、がモニタ中に存在すると、問題
- モニタから他の資源を押さえに行って、押さえられないとか、それが原因でデッドロックとか
  - アプリケーションが馬鹿だと何を使ってもデッドロックする
  - Javaのモニタは、モニタから他のモニタを呼べない(はず)よって、単純なデッドロックは起こらない