

並行システムの検証と実装

平成23年度シラバス

2011年7月15日

国立情報学研究所

トップエスイープロジェクト

代表者 本位田 真一

1. 講座名

並行システムの検証と実装

2. 担当者

磯部 祥尚

3. 本講座の目的

並行システムは複数のプロセスから構成されるシステムであり、各プロセスは他のプロセスと通信しながら同時に動作することができる。その通信方式は、共有メモリ上でメッセージを交換する**共有メモリ通信方式**と、チャンネルを通してメッセージを送受信する**メッセージパッシング通信方式**に大別される（図1参照）。一般に、共有メモリ通信はメッセージパッシング通信に比べて高速であるが、共有メモリ通信による並行システムの動作を理解することは難しく、デッドロックやリソース競合などの不具合が発生する可能性が比較的高い。

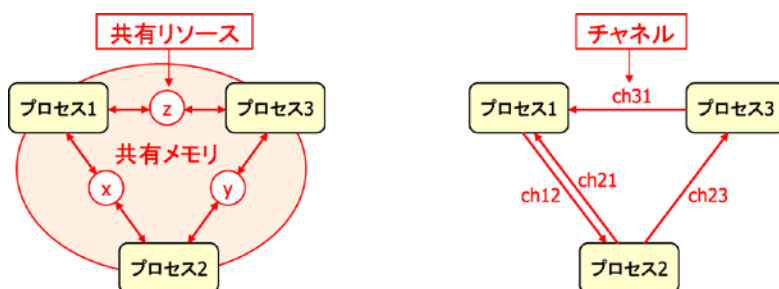


図1 共通メモリ通信（左）とメッセージパッシング通信（右）の構成例

本講座では、信頼性の高い並行システムを構築することを目的として、次の三つのキーワード（CSP, FDR, JCSP）を軸に、CSPによる並行システムのモデル化、FDRによる検証、JCSPによる実装方法について説明する。

- CSP** : 並行システムを記述・解析するための理論 (プロセス代数)
- FDR** : CSP理論に基づく並行システム検証ツール (モデル検査器)
- JCSP** : CSPのプロセスモデルの実装ツール (Javaライブラリ)

JCSPはJavaで同期式（送信と受信が同期する方式）のメッセージパッシング通信（CSPの通信方式）を実装するためのライブラリである。同期式であるためメッセージの取りこぼしがなく、より信頼性の高い並行システムを構築することができる。さらに、JCSPプログラムからCSPの通信部分を抽出し、モデル検査器FDRによってその並行動作を検証することも可能である。尚、本講座では次の用語を用いる。

- CSPモデル** : 並行システムのCSP記述（形式的な記述）
- FDRスクリプト** : モデル検査器FDRで読み込み可能な記述（FDR入力言語）
- JCSPプログラム** : ライブラリJCSPを利用しているJavaプログラム

4. 本講座のオリジナリティ

JCSP は CSP 理論の試験的な実装ではなく、実際のシステム開発に利用できるほどの性能を有している。しかし、残念ながら JCSP プログラミング技法を習得できる機会は極めて少ない。また、JCSP の基礎になっているプロセス代数 CSP やその検証ツール FDR についても、オートマトンやその検証ツール (SPIN 等) に比べると学習する機会は少ない。本講座では、プロセス代数によるモデル化の特徴を紹介しつつ、その特徴を活かした検証方法を解説し、それらに基づく JCSP プログラミング技法を説明する。尚、JCSP (Java) の他に C++CSP(C++), PyCSP (Python), CHP (Haskell) 等の CSP ライブラリや、Google の Go 言語のように言語レベルで CSP に基づく並列処理記述が可能なプログラミング言語も公開されており、本講座の開発手法は Java 以外のプログラミング言語にも有効である。

表 1 既存の講座の問題点と本講座における解

| 既存の講座の問題点 | 本講座における解 |
|---|---|
| 並行システムのモデル化にはオートマトンが使われることが多いが、オートマトン理論では、非決定的な動作を適切にとらえられない場合がある。 | 基礎理論にプロセス代数 CSP を採用することによって、並行システムの動作をより適切にモデル化することができる。 |
| 一般的なモデル検査器 (例 : SPIN, SMV) では、実装をオートマトン、要求を時相論理で記述する。すなわち、実装が要求を満たすかを検証する。このような充足関係では部分的な特性を容易に検証できる利点があるが、その一方で全体の動作がどのようになるのかを時相論理だけで記述することは容易ではない。 | CSP に基づくモデル検査器 FDR では、実装も仕様 (要求) も同じ CSP でモデル化される。すなわち、実装が仕様の詳細化であるかを検証する。並行システム全体の動作と抽象的な仕様が観測的に等しいことなども検証できる。また、この詳細化関係はプロセスを並行合成した場合も保存されるため、部品単位での検証が可能になるなどの利点もある。 |
| 検証されたモデルができて、それを適切に実装する枠組みがなければ高信頼なシステムを実現することはできない。例えば、オートマトンによる検証と共有メモリ通信による実装のギャップは大きい。 | Java のライブラリ JCSP を用いることによって、CSP モデルを比較的簡単に実装することが可能になる。逆に、JCSP プログラムから通信部分を抜き出してその動作を FDR で検証することもできる。 |
| 並行システム理論と並列プログラミングは別々に講義されることが多い。 | 理論 (CSP) → 検証 (FDR) → 実装 (JCSP) の流れを重視し、それぞれの関係に常に注意を払いながら講義を構成している。 |

5. 本講座で扱う難しさ

最近では多くのパソコンがマルチコア CPU を搭載するようになり、そのコア数は 2 個から 4 個に移行しつつある。また、48 コアをワンチップに実装した SCC (Single-Chip Cloud Computer) の動作デモに成功したとの報告もあり (2009 年 12 月)、今後コア数はさらに増えていくと予想される。従来、パソコンの性能向上は主に CPU の周波数を上げることによって行われ、周波数が上がれば、どのようなプログラムの実行速度も自動的に改善されてきた。一方、マルチコア CPU のコア数を増やしても、プログラムが並列処理に対応していなければ、実行速度の改善は望めない。今後のマルチコア CPU の恩恵を受けるには、何らかの方法で並列処理を実装する必要がある。最近では自動的にプログラムを並列化するツールもあるが、より複雑な並列処理を実現するためには、各プロセスの実行のタイミングをプログラマが適切に記述する必要がある。しかし、並列処理ではデッドロックやリソース競合のような不具合が発生する可能性があるため、その全体の動作を適切に予測し、個々のプロセスを制御することは容易ではない。

6. 本講座で習得する技術

本講座では、CSP の通信・並行処理モデル (同期式メッセージパッシング通信) に基づく JCSP プログラミング技術を身につける。共有メモリ通信を使った並行プログラミングと比較し、どのプロセスがどのタイミングで同期 (通信) するのかが分かりやすくなり、より安全で簡潔なプログラムが書けるようになる。

JCSP を使うことでデッドロックのようなバグの発生率を低くすることができるが、それでも複雑な並行動作を完全に把握することは難しい。本講座では、CSP のモデル検査器 FDR による検証技術も身に付け、JCSP プログラムの信頼性をさらに高める方法を習得する。そのために、FDR スクリプトと JCSP プログラムの違いを適切に理解し、FDR スクリプトから JCSP プログラムへの変換ができるようにする。

FDR は CSP 理論に基づくモデル検査器である。本講座では、FDR でどのような等価関係や詳細化関係を検証しているのかを理解できるように CSP 理論についても説明する。これにより、FDR の検証機能を適切に使えるようになるだけでなく、検証結果が偽の場合の反例をより理解できるようになる。

以上、CSP によるモデル化、FDR による検証、JCSP による実装の方法を身につけることにより、信頼性の高い並行システムの開発が可能になる。

7. 前提知識

本講座の受講生は Java プログラミングの基礎知識を有すること。FDR は Linux 上で動作するため、Linux の基本的なコマンドを使えることが望ましい。並列プログラムやプロセス代数についての前提知識は必要ない。

8. 講義計画

本講義では、CSP 理論、FDR 検証、JCSP 実装を個々に教えるのではなく、可能な限り、CSP→FDR→JCSP の流れを重視して教えることを一つの特徴にしている。その特徴を活かせるように、講義計画はそのパターン（CSP→FDR→JCSP）を繰り返して、個々ではなく全体として徐々に詳しい説明をするように構成されている（図 2 参照）。

- 概要

- 第 1,2 回：CSP, FDR, JCSP 概論（並行システムのモデル化、検証、実装の概要）
- 第 3,4 回：CSP 入門（モデル化の基礎）
- 第 5 回：FDR 入門（検証の基礎）
- 第 6,7 回：JCSP 入門（実装の基礎）
- 第 8 回：CSP 理論（並行動作の表現）
- 第 9 回：CSP 理論（並行動作の解析）
- 第 10 回：FDR 検証（検証とデバッグの例）
- 第 11 回：JCSP 実装（共有チャンネル、ネットワークチャンネル）
- 第 12 回：CSP, FDR, JCSP 応用（チャンネル渡し、抽象化）
- 第 13 週：グループ討論（グループ課題設定：並行システム）
- 第 14 週：グループ実習（作業：並行システムのモデル化、検証、実装）
- 第 15 週：グループ発表（発表：並行システムのモデル化、検証、実装）

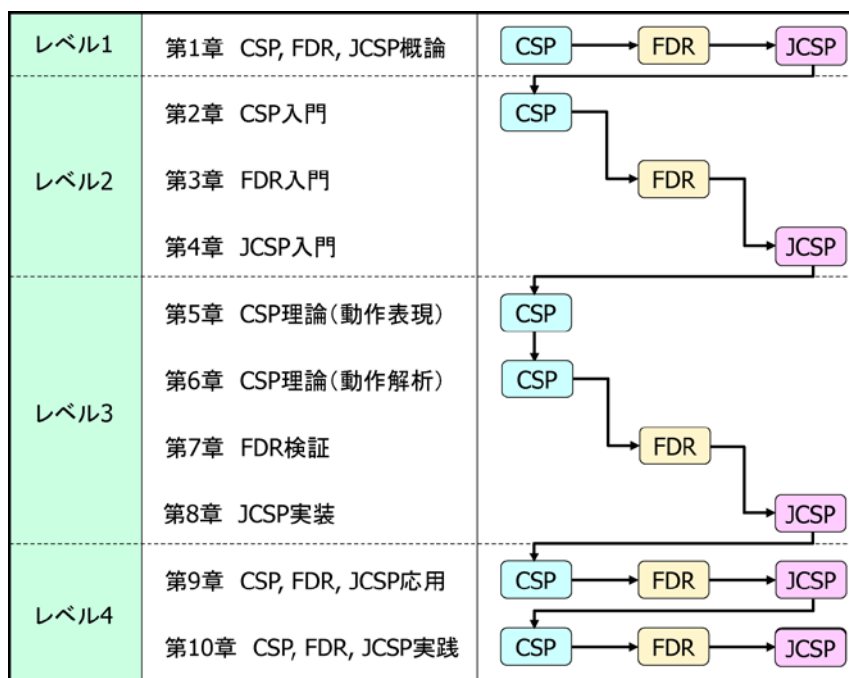


図 2 配布予定の講義テキスト（随時配布）の章立て

- ・ 詳細

第 1,2 回 : CSP, FDR, JCSP 概論

- 簡単な例による、モデル化、検証、実装の概要
- 演習 : 各ツールの動作確認

第 3,4 回 : CSP 入門

- CSP による並行プロセスの記述
- 演習 : CSP モデルの記述

第 5 回 : FDR 入門

- FDR による並行プロセスの記述
- 演習 : FDR によるデッドロック解析

第 6,7 回 : JCSP 入門

- CSP モデルから JCSP プログラムへの変換方法
- 演習 : JCSP による並行プログラミング

第 8 回 : CSP 理論 (動作表現)

- 並行動作の状態遷移図による表現方法
- 演習 : 補助ツール ProBE による遷移の確認

第 9 回 : CSP 理論 (動作解析)

- 並行システムの等価関係や詳細化関係
- 演習 : 詳細化仕様の作成

第 10 回 : FDR 検証

- FDR による詳細化関係の検証とデバッグ方法
- 演習 : 詳細化関係の検証

第 11 回 : JCSP 実装

- 共有チャンネルとネットワークチャンネル
- 演習 : JCSP によるネットワークプログラミング

第 12 回 : CSP, FDR, JCSP 応用

- より高度な検証と実装テクニック
- 演習 : グループ作業の準備

第 13 回 : グループ討論

- グループ発表のための課題 (並行システム) の討論

第 14 回 : グループ実習

- グループ発表のための実習 (モデル化、検証、実装)

第 15 回 : グループ発表

- 最終発表会 (モデル化、検証、実装)

6. 教育効果

本講座を受講することにより、プロセス代数 CSP に基づくモデル化、検証、実装の方法を習得できる。理論から実装までの流れを理解することによって、検証された信頼性の高い並行システムを開発できるようになる。

7. 使用ツール

FDR : CSP のモデル検査ツール

- ・ 使用する上での難しさ
 - 仕様（要求）のモデルの作成
 - 詳細化関係の意味
- ・ 使用上必要なノウハウ
 - 並行動作のモデル化ノウハウ
 - 詳細化関係の使い分けノウハウ
- ・ 選択理由、実用性
 - CSP モデルの検証に最適
 - データの高い表現力

JCSP : CSP のプロセスモデルを Java で実装するためのライブラリ

- ・ 使用する上での難しさ
 - 通信チャンネルの種類と使い分け
 - FDR スクリプト（CSP モデル）との違い
- ・ 使用上必要なノウハウ
 - 並行システム実装ノウハウ
 - FDR スクリプト（CSP モデル）から JCSP プログラムへの変換ノウハウ
- ・ 選択理由、実用性
 - CSP モデルの実装に最適
 - ネットワークチャンネルを利用して分散処理も簡単に実現可能

8. 実験及び演習

CSP, FDR, JCSP に慣れるために、早い段階から小さな例題をもとに演習を行う。最終的には、3~4名程度のグループをつくり、グループごとに考案した並行システムのモデル化、検証、実装を行う。実際に自分で考えた並行システムを検証し実装することで、検証の有効性を実感することができる。

9. 評価

演習課題レポート、グループ作業・発表、出席日数を総合して評価する。

10. 教科書/参考書

平成 19 年度の本講義テキスト（サイエンスによる知的ものづくり「並行システムのモデル化、検証、実装」、本位田真一監修、磯部祥尚著）が参考になるが、図 2 に示したようにその構成や内容を修正している。本講義期間中、講義で使用するスライドの他に、最新版のテキストを随時配布する。その他、CSP, FDR, JCSP の参考文献をあげておく。

- A.W.Roscoe. The Theory and Practice of Concurrency. Prentice Hall, 1998. 代表的な CSP 理論の教科書であり、検証ツール FDR についても説明がある。
<http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf> からダウンロード可能。
- Failures-Divergence Refinement (FDR2 User Manual), 2005. FDR をインストールすると添付されているマニュアル。基本的な操作方法や構文が書かれている。
<http://www.fsel.com/documentation/fdr2/fdr2manual.pdf> からダウンロード可能
- 残念ながら JCSP の教科書的な文献はないが、下記の JCSP のウェブサイトでは JCSP に関する様々な情報を得ることができる。
<http://www.es.kent.ac.uk/projects/ofa/jcsp/>

尚、下記ウェブサイトにて、FDR と JCSP のインストール方法を説明しているので参考にしてほしい。

本講座の補足ウェブサイト：<http://staff.aist.go.jp/y-isobe/pw/topsevic2009/>

付録 1

本講座で習得する並行システムのモデル化、検証、実装の特徴を、図 3 の並行挿入ソートの例 ParSort を用いて簡単に紹介する。初期状態では、この挿入ソートは 2 つのプロセス Manager と TermSort から構成されている。図 3(a)に示すように、これらのプロセスは次の手順で接続されている。

1. TermSort の 2 つのチャンネル dataL と retL を各々 dataR と retR に名前変更する。
2. TermSort と Manager をチャンネル dataR と retR を通じて並行合成する。
3. チャンネル dataR と retR を外部から隠す。

プロセス Manager はチャンネル dataR にデータ (乱数整数) を繰り返し送信し、TermSort はそのデータを受信して順次挿入ソートする。ただし、TermSort は自分がソートするデータ数が上限値 M を超えると、ソート作業を分業するために新しくプロセス Sort を生成する。この動作は Manager から整数が送信される限り続けられるため、その構造は図 3(b)(c)のように徐々に変化する。Manager から終了信号が送信されると、ソート結果がチャンネル retR を通じて Manager に返信され、最後にチャンネル disp を通じて外部に送信される。

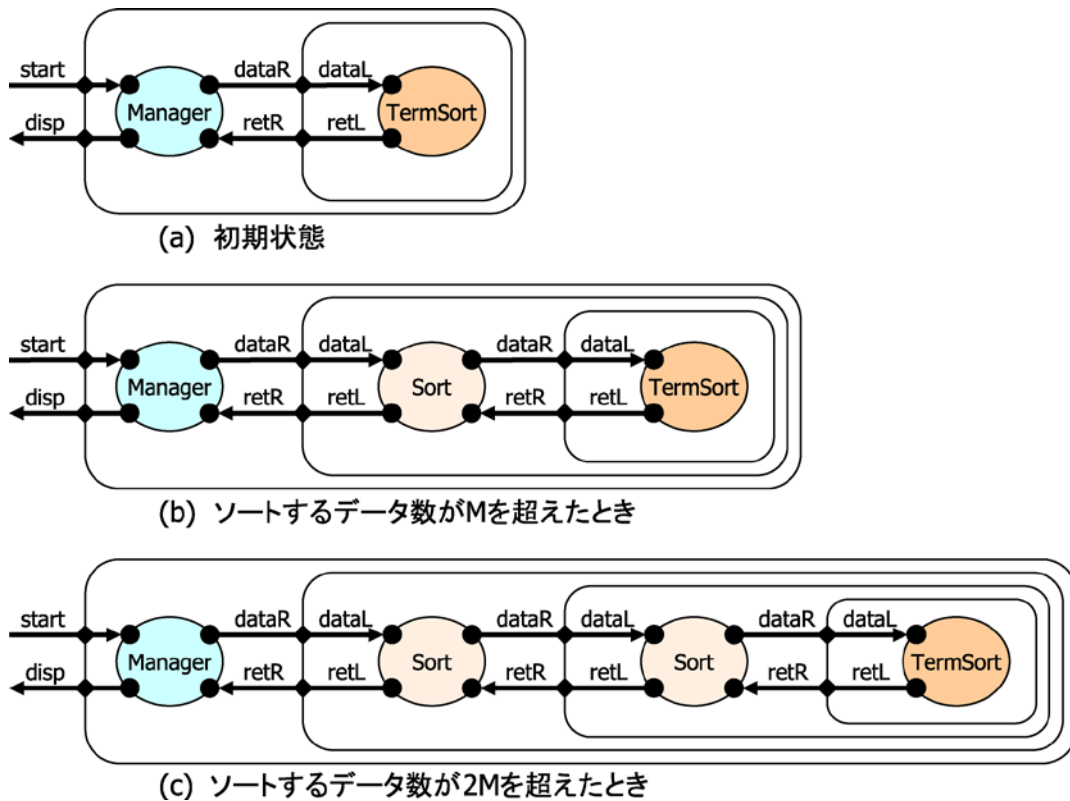


図 3 並行挿入ソート ParSort の構造 (M : 1 つのプロセスが処理する最大データ数)

この並行システムとその仕様の FDR スクリプト (CSP モデル) を作成し、実際に FDR で検証した結果を図 4 に示す (データ数 5, データ変域{0,1}, 各ソートプロセスのデータ数の上限 (M) 2, 検証時間約 2 分、Pentium-M, 1.5GHz, 768MB RAM)。ここで、Spec は start 後に (いつか) 必ず display チャンネルからソートされたリストが出力されることを要求する仕様である。図 4 の中央左の 1 番目の✓が ParSort がデッドロックフリー (DF) であること、2,3 番目の✓が ParSort と Spec は等しくなることを表している。

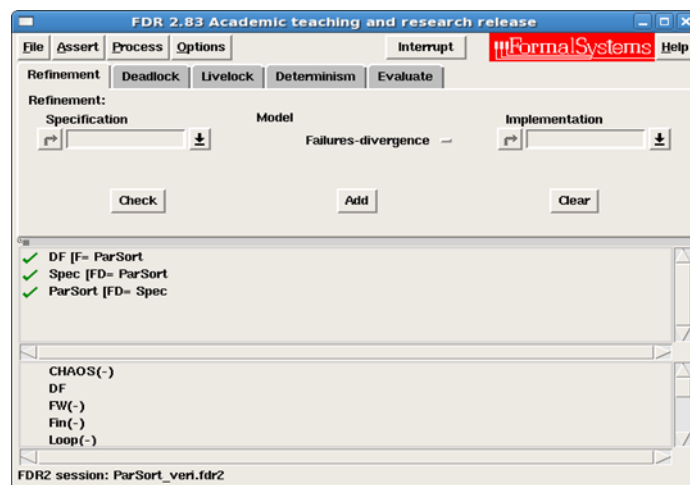


図 4 FDR による挿入ソートの検証結果

また、図 3 の並行挿入ソート ParSort を JCSP で実装し、ソートプロセスの数を 1 から 15 まで増加させた時のソート処理時間 (データ数 400,000) を図 5 に示す。使用したパソコンは 4 コア CPU を搭載している (Intel Core 2 Quad Q9550 CPU, 2.83GHz, 2GB RAM)。プロセス数を増加させることによって 4 つのコアが有効に使われ、ソートプロセス数が 15 個のときは 1 個のときよりも処理時間が約 70%減少している (4 コアであるので理想は 75%の減少)。この挿入ソートの例では、ソート開始直後は TermSort プロセスのみがソート処理を行うため、複数のコアを有効に利用できない。プロセス数を増やして (上限値 M を下げて) 早い段階で処理が新しいソートプロセスが生成されるようにすることによってソートが並列に行われるようになり、その処理時間が短くなる。尚、400,000 のデータを全く並列化せずに (JCSP を使わずに)、逐次処理で挿入ソートしたときの処理時間は 140 秒程度であり、図 5 のソートプロセス数が 1 の場合と大差はなかった。

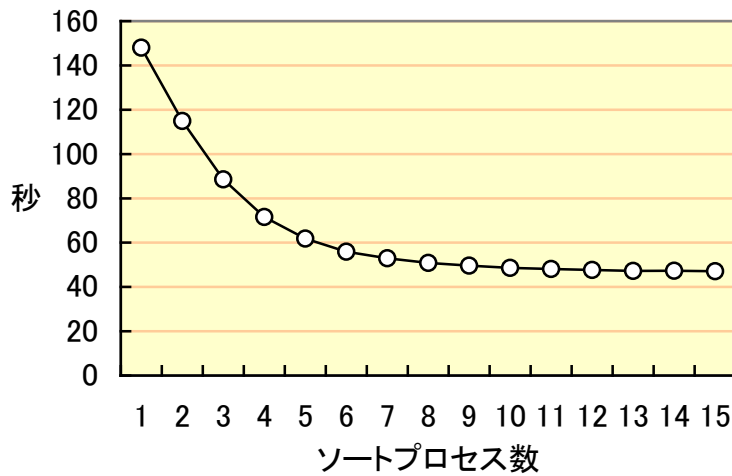
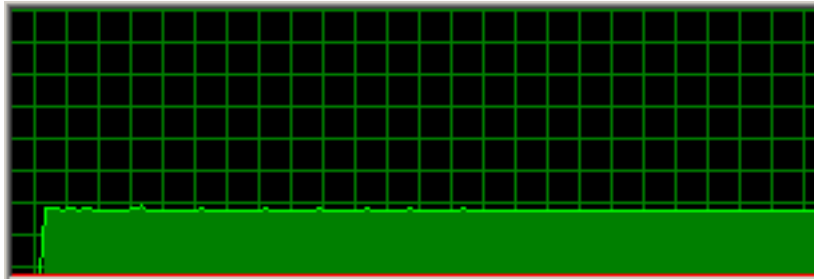


図5 ソートプロセス数とソート時間の関係（4コア CPU、400,000 データ）

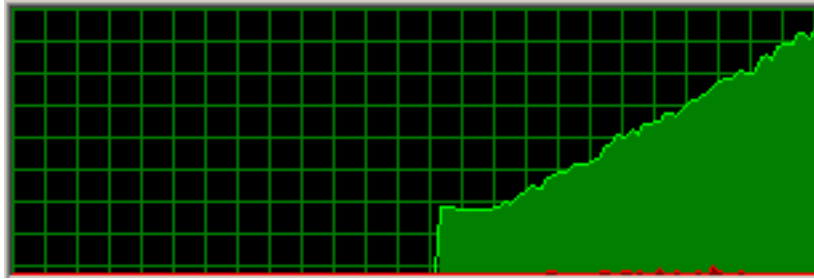
図6に、(a) ソートプロセス数1、(b) ソートプロセス数4、(c) ソートプロセス数15の場合の、各々のソート（400,000 データ）中の CPU 使用率の履歴（タスクマネージャー Process-Explore による）を示す。逐次処理（ソートプロセス数1）では1つのコアしか使用できないため、その CPU 使用率は（4 コアに対して）25%程度になっている。ソートプロセス数が4の場合に CPU 使用率が時間と共に徐々に増加する理由は、ソートするデータ数と共にその処理の並列度が増加するためである。ソートプロセス数15の場合は、早い段階から4つのコアを有効利用できるようになるため、全体の CPU 使用率も増加している。

この挿入ソートの例では、ソートプロセス数とチャンネル数がデータ数に応じて増加し、その構造が動的に変化する。このような並行システムを正しく設計することは容易ではないが、その動作を FDR で検証することによって、実装前にバグを発見することができる。また、JCSP ではその検証済みの CSP モデルを実装できるため、信頼性の高い並行システムを構築することができる。

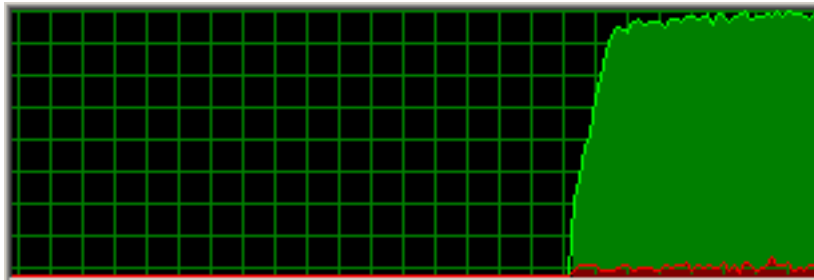
本講座では、ここで紹介した並行挿入ソートの CSP モデル、FDR スクリプト、JCSP プログラムを作成するために必要なテクニックを一通り紹介している。この機会に本講座を受講して CSP, FDR, JCSP を習得してほしい。最後に、CSP モデルを実装可能なライブラリとプログラミング言語の例を表2にまとめておく。



(a) ソートプロセス数 1 (処理時間 148 秒)



(b) ソートプロセス数 4 (処理時間 72 秒)



(c) ソートプロセス数 15 (処理時間 47 秒)

図 6 挿入ソート (400,000 データ) 実行中の CPU 使用率の履歴

表 2 CSP ライブラリとプログラミング言語

| 言語 | ライブラリ | 関連 URL (ダウンロード可能) |
|----------------|------------|---|
| Java | JCSP | http://www.cs.kent.ac.uk/projects/ofa/jcsp/ |
| C++ | C++CSP | http://www.cs.kent.ac.uk/projects/ofa/c++csp/ |
| Haskell | CHP | http://www.cs.kent.ac.uk/projects/ofa/chp/ |
| Python | PyCSP | http://code.google.com/p/pycsp/ |
| Python | Python-CSP | http://code.google.com/p/python-csp/ |
| Go 言語 (Google) | | http://golang.org/ |
| XC (XMOS) | | http://www.xmos.com/jp |