仮想的にCPUコアの追加と実行見積もりが可能なPrismによる

マルチコアにおけるボトルネック検出と対策手法

柴下 哲，
Barry O'Rourke

現状のシングルCPUコア向けのアプリケーションをマルチコアで動作させても，コア数に比例して性能が向上することはまずない．何個のコアを用いれば目的の性能が得られるのかは，マルチコアCPUを搭載した実機のボードで試さなければわからなかった．

本稿では，マルチコア向けソフトウェアを解析・評価する英国CriticalBlue社のPrismを使用し，仮想的にコアを追加した場合の性能見積もりや，データの待ちを解決することによる性能向上の解析などを行う．（編集部）

1．マルチコアにしただけでは高速化は望めない

組み込み機器の高機能化と低消費電力化のためにマルチコア化が注目されています．既存のシングルコア用に書かれたプログラムをそのままマルチコアで動作させても，ほとんどの場合，それだけで高速化は望めません．最悪の場合，一つのコアだけがプログラムを実行し，そのほかでは何も処理を行っていないということもあります．

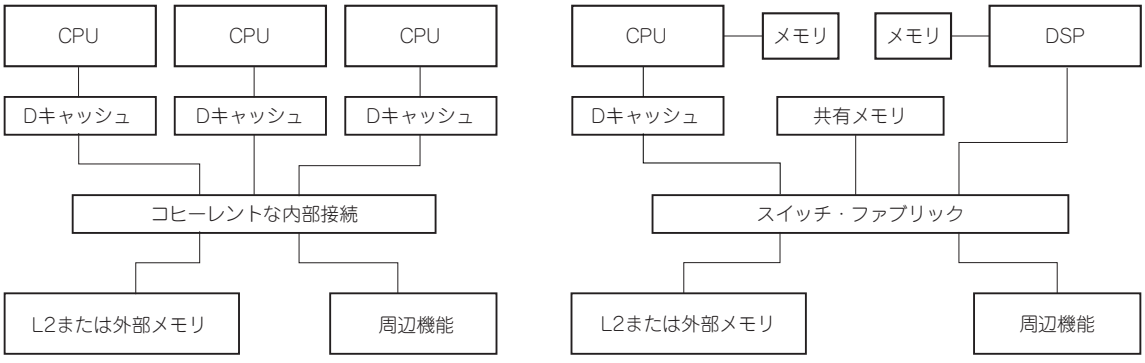
この問題を解決するには一つのプログラムを複数のタスクに分割し，並列実行できるようにしなければなりません．しかし並列実行を可能にすると，資源の待ちによるタスクの待ちやデッドロックなどさまざまな問題が発生します．また，マルチコア化によってどのくらいパフォーマンスが向上するのかは，実際にプログラムを実装して実機のボード上で動かしてからでなければわかりません．

ここでは最初にその概略を説明し，その後にマルチコア用のプログラミングの問題点と解決方法の一例を紹介します．

2．分野ごとのマルチコアのアーキテクチャと問題点

図1のように，パソコンやサーバ，スマートホンなどの通信分野では，同種のプロセッサが共有メモリを持つ，SMP（Symmetric Multi-Processor）型のアーキテクチャが広く普及しています．また組み込み系では，プロセッサとDSPなどを組み合わせたヘテロジニアスなAMP（Asymmetric Multi-Processor）が一般的です（**コラム1**参照）．組み込みの場合は，マルチコアがそれぞれ分担する役割は明確で，データの共有もあまりないため，ソフトウェア開発者はほとんどマルチコアを意識せずにコードを書くことができました．

しかし分担する仕事量が増大し，かつ低消費電力化の要望のために，クロック周波数をあまり上げられなくなると，**図1**のCPUやDSPをそれぞれマルチコア化する必要性が出てきました．この場合，従来は一つのコアで行っていた仕事を分担するので，各ブロックは同種のプロセッサの方が扱いやすく，かつ同じデータをアクセスする必要性も高いので，共有メモリとなります．したがって全体としては



（a）SMP型アーキテクチャの例　（b）AMP型アーキテクチャの例

図1
アプリケーションによるマルチコア・アーキテクチャの違い

KEYWORD——マルチコア，タスク，Prism，データ・レース