実行時の情報を使って動的に最適化コードを生成…
使い方から実践例まで

C/C++プログラムの高速化とJITアセンブラ

光成 滋生

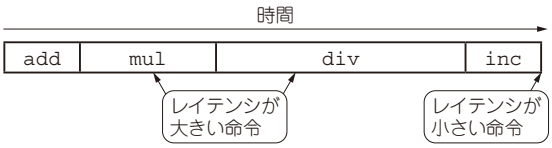

図1　単純に逐次処理する場合，遅い命令の完了待ちが発生する

プログラムは，どのような言語を使う場合でも，書き方によって処理速度が変わります．同じ結果を得るプログラムであっても，最適化によってメモリやCPUタイムといったリソースの使用量に差が出ます．優秀なコンパイラを利用できる現在では，あまりプログラマ主導の最適化を行わないことも増えているかもしれませんが，最適化を施すことで大きな改善を生む場合もあります．

本稿では，プログラムが実行される瞬間に判明する値を最適化に用いる技術JIT（Just in Time）を紹介します．与えられた値に応じたアセンブリ・コードを動的に生成することで，特定の処理の実行効率を高めます．（編集部）

基礎知識1：逐次処理とパイプライン処理

現代のほとんどのCPUはメモリ上に配置された命令列を読み取り，命令を1つずつ順番に実行します．CPUが持つ命令はレジスタの値を1増やすといった簡単なものから，除算や行列演算など複雑なものもあります．それぞれの命令について処理にかかる時間をレイテンシ（latency）と言い，単位はクロック・サイクルで表されます．CPUが命令を逐次処理する場合，複雑な命令の処理が終わるまで，後続の命令は実行を待たなければなりません．

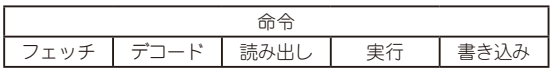

図2　1命令を5つのステージに分割した

図1の場合，divの処理が終わるまでincは処理を始められません．

● 小分けにして並列動作させるパイプライン

そこで多くのCPUでは，命令の実行を複数のステージに分割し，異なるステージを同時に実行できるように工夫されています．これをパイプライン処理と言います．ステージの分割の仕方はCPUごとに異なりますが，ここでは仮に次の5つのステージに分割されているとして説明します（**図2**）．

- 命令の読み込み（フェッチ：FE）
- 命令の解析（デコード：DE）
- オペランドの読み出し（読み出し：LD）
- 演算（実行：EX）
- 結果の書き込み（書き込み：WR）

オペランドとは命令の引数を意味します．それぞれのステージは独立して動作し，異なるステージは同時に実行できます．すると**図1**の逐次処理をパイプライン実行できた場合，**図3**のように処理できます．ただし，**図3**ではadd，mul，div，incの命令はそれ

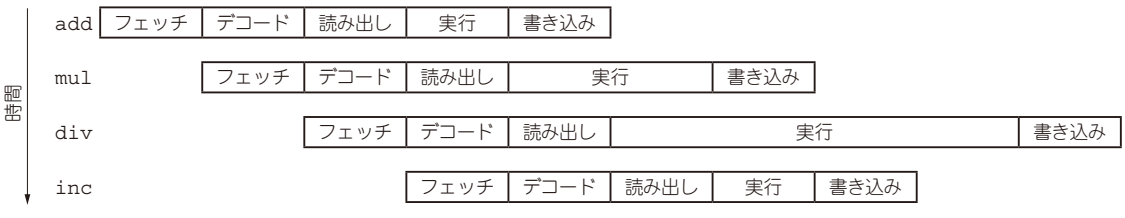

図3　パイプライン処理の実行例