マルチコア向け ソフトウェア最適化技術

マルチコア時代の並列プログラミング方式

マイクロプロセッサの動作周波数の向上が頭打ちとなり，今後のコンピュータの性能向上は複数のCPUを1チップに集積したマルチコアに期待されています。マルチコアで十分な性能を得るには，プログラムを並列化して，複数のコアで効率よく動作させることが重要となります。そのためには，従来とは異なるプログラムの開発手法が必要です。

東芝は，マルチコア向けの並列プログラムを効率よく作成して高い性能を発揮できるようなプログラミング環境を開発し，大規模で複雑な計算をリアルタイムで処理することによって，新しいアプリケーションを実現します。

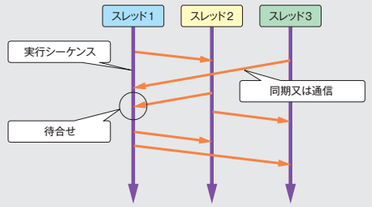

図1. 複数スレッドによるデータのやり取り — 一般的なスレッド処理では，データへのアクセス順序や処理手順の整合性を正しく行えるように，複数のスレッド間で同期や通信を行って並列に処理を進めます。

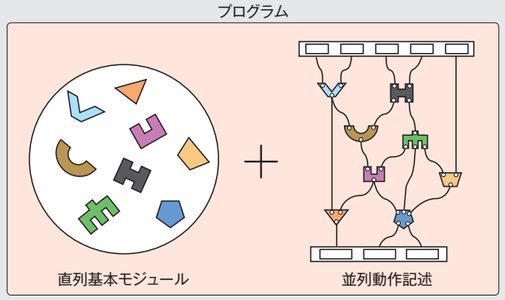

図2. 直列基本モジュールと並列動作記述 — プログラムを，同期や通信なしで最初から最後まで実行する直列基本モジュールと，それらのデータの入出力(定義－参照関係)のつながりを定義する並列動作記述から構成します。

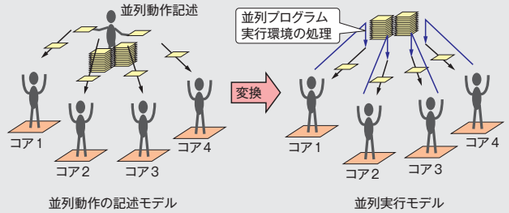

図3. 並列動作の記述モデルと並列実行モデル — プログラムの記述は，並列動作を一連の処理の流れとして記述しますが，プログラム開発時にこの記述をグラフ構造に変換しておくので，プログラム実行時には，各コア上で動作しているスレッドが，次に実行するべきモジュールを自律的に見つけだします。

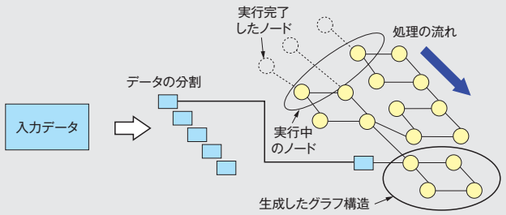

図4. グラフデータ生成情報から動的にグラフデータを生成 — プログラム実行時には，分割した入力データのそれぞれに必要な処理をグラフ構造として生成し，既存のグラフに接続します。

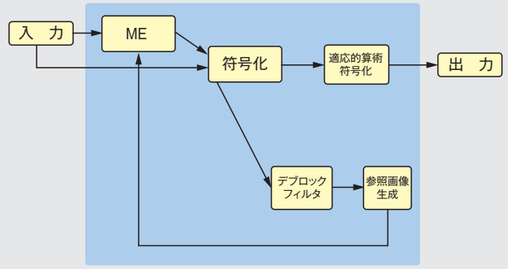

図5. H.264エンコーダのモジュール構成とデータフロー — H.264の機能モジュールごとのブロックと，データの流れを表しています。

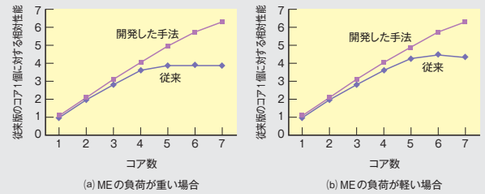

図6. H.264エンコーダの性能測定結果 — この技術をCellプロセッサ上のH.264エンコーダに適用したときの，性能評価結果を示します。

マルチコアにおける技術課題は並列プログラミング

マルチコア向けのプログラム実行モデルとしてマルチスレッド処理があります。スレッドとはプログラムの実行単位で，マルチスレッド処理では，複数並列に動作するスレッドが，メモリ上のデータをやり取りすることによって並列処理を遂行します(**図1**)。

ここで，同期処理が適切でないと，お互いが待合せを行って“デッドロック”に陥ったり，共有データに同時にアクセスしてデータの整合性が崩れてしまうなどの問題が発生します。また，同期の待合せによって，並列性能が十分得られないこともあります。このようなタイミング問題は難しく，修正に手間がかかるため，ソフトウェア開発に多大な影響を及ぼします。

東芝は，平易な並列プログラミング環境及び高効率な並列プログラム実行環境を実現するために，マルチコア向けのソフトウェア最適化技術を開発しています。平易な並列プログラミング環境は，デバッグコストを削減するとともに，性能チューニングのための並列アルゴリズムの検討と変更を容易に行えるようにします。高効率な並列プログラム実行環境は，コアの数に比例して実行性能が高くなるような効率の良い実行環境を提供します。

今後，コア数が増えたときに，プログラムを作り直すことなく性能が向上することが重要です。

プログラムを直列基本モジュールと並列動作記述に分離

コアの数に比例して性能を高めるためには，プログラムの処理単位を細かく分割し，実行順序の制約をできるだけ小さくしておく必要があります。プログラムの分割が十分でないと，コア数が増えた場合に，割り当てる処理が足りなくなるからです。

細かく分割した処理は，開始から終了まで，同期なしで直列実行する直列基本モジュールとして定義します(**図2**)。これには，従来の直列性能の高速化手法を使うことができます。

並列処理は，これらの直列基本モジュールを組み合わせて構成します。この並列動作の記述には関数型言語を使います。各直列基本モジュールの出力データと入力データの“定義−参照”関係を，関数の返り値と引数として定義することで，モジュール間のデータ依存による最低限の実行順序を自然に表現することができます。この並列動作記述は，プログラム開発時に，変換処理(トランスレータ)によってグラフ構造の生成情報に変換します(**図3**)。

プログラム実行時には，並列プログラム実行環境がコア数に応じてスレッドを生成し，入力データをトリガとしてグラフデータの生成情報に基づいて，グラフ構造を生成します(**図4**)。生成されたグラフ構造のノード(○印)は直列基本モジュールを表し，ノードを結ぶエッジはデータの流れを表します。

並列プログラム実行環境は，このグラフが表現する実行順序に従って，コアに処理を割り当てます。実際の動作では，図3に示すように自律的に動作する各スレッドから実行環境が呼び出される形で動作します。呼び出された実行環境は，それぞれのコア上で動作し，直列基本モジュールの完了に伴うグラフ構造の更新を行い，次に実行するべき直列基本モジュールを求めます。

H.264(注1)エンコーダによる評価

このプログラミングモデルと実行環境の有効性を，Cellプロセッサ上で動作するH.264エンコーダに適用して評価しました。

H.264エンコーダは，動画フレームを複数のスライスに分割すれば，容易にスライス単位で並列処理できますが，画質が劣化する問題があります。今回は，スライス分割することなく機能モジュール単位に分割し，各モジュールのデータフローの依存関係に従って並列処理を行いました(**図5**)。

従来は，コア数が一定数を超えると性能が向上しない傾向があり，特に，動き予測処理(ME：Motion Estimation)の負荷の割合が高い場合にその傾向が顕著でした。しかし，この技術を適用することで，コア数に応じた性能向上が達成できることを確認しました(**図6**)。これは，並列プログラム実行環境において，負荷状況に適応して動的に直列基本モジュールの実行割当てを決定できるようになったことと，メモリバッファを効率的に利用できるようになったことの結果です。

(注1) 1セグ，HD-DVDなどに採用された動画圧縮規格。圧縮及びデコードのための計算量が多い。

並列プログラムの生産性向上を目指して

マルチコアは，並列プログラムによって高い性能を得てこそ，その真価を発揮します。今後は，プログラム実行時の負荷状況に応じてパラメータやアルゴリズムを動的に選択する自動チューニング機能の追加や，並列動作記述の変換処理時に各種の最適化処理を盛り込んでいく予定です。

境　　隆二
デジタルメディアネットワーク社
コアテクノロジーセンター
エンベディッドシステムコア技術開発部参事