マルチコア プロセッサのための並列プログラミング技術—Molatomium

簡易な記述でマルチコアの性能を引き出す

パソコンだけでなく組込み機器でも，マルチコア プロセッサが搭載されるようになってきており，マルチコアの性能を引き出すプログラミング技術が注目されています。しかし，多くの並列プログラミング技術は，記述が複雑になりバグ（不具合）が発生しやすい，マルチコアに最適な負荷分散が難しい，などの難点があり，これらを改善することが一般的な課題です。

そこで，東芝は，“Molatomium”（モラトミウム）という並列プログラミング技術を開発しました。これによって，簡易な記述でもマルチコアにおける最適な負荷分散が実現できるようになりました。

Molatomium開発の背景

近年，発熱や消費電力の問題から，プロセッサの動作周波数上昇による性能向上が困難になっており，これに代わる性能向上の方法として，プロセッサの演算装置であるCPUコアを複数実装するマルチコア プロセッサが採用されるようになってきました。

パソコンでは，既に2コア又は4コアの製品が標準になっています。組込み機器である家電製品でも，東芝の液晶テレビ CELL REGZA$_{TM}$ではヘテロジニアス（非対称）なマルチコア プロセッサであるCell Broadband Engine^{TM}（Cell/B.E.^{TM}）(注1)を搭載しており，そのほかの製品でもマルチコア プロセッサの搭載が予定されています。

（注1） Cell Broadband Engine及びCell/B.E.は，（株）ソニー・コンピュータエンタテインメントの商標。

マルチコア プロセッサを使用して高い処理性能を持ったソフトウェアを開発するためには“並列プログラミング”が必要になります。これは，それぞれのコアが並列に動作するように処理を割りふりながら，かつ，それらの処理を協調させて全体のプログラムを実行させるプログラミング技術です。しかし，この手法は，通常のプログラミングと比較して格段に難しいとされています。

まず，個々のコアがプログラムを並列に実行するため，プログラムの実行順序が一定にならず，これを非決定性と言います。そのためバグの再現性がランダムになり，デバッグが困難になります。

また，並列して動作するプログラムを同期させるために排他制御が必要ですが，順序不一致や排他漏れがあると，デッドロックや競合状態を引き起こす原因となります。

更に，動作中のコア個々に処理負荷を正確に見積もるのは困難なため，処理負荷が最適になるようにあらかじめプログラムを記述するのは，かなりの工数を必要とする作業になります。

これらの問題を解決するため，当社は“Molatomium”（モラトミウム）という並列プログラミング技術を開発しました。

Molatomiumのプログラミング モデル

Molatomiumでは，C言語のような手続き的なプログラム表現ではなく，性質を記述する（宣言的）プログラム表現を採用することで，並列プログラムを容易に記述できるようになりました。

Molatomiumの構成を**図1**に示します。AtomというC/C++のプログラムと，それらの依存関係をMolという独自のプログラミング言語で記述したもの，処理系が解釈し実行することで並列処理を実現しています。

（注2） ARMは，英国ARM社の登録商標。

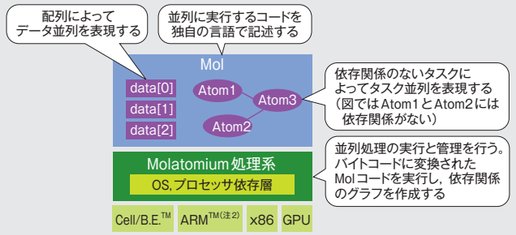

図1. Molatomiumのソフトウェア構成 — 並列処理可能なプロセッサ（下段）のアーキテクチャの違いをMolatomium処理系（中段）が抽象化する。Molの並列プログラム（上段）は，プロセッサの違いを意識せずに配列やタスク（Atom）を組み合わせることで記述できる。

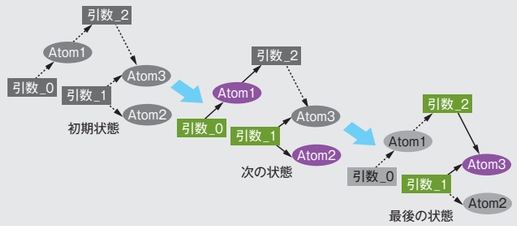

図2. Atomの依存関係に従って処理系が処理を行うようす — 初期状態では，どの引数も決定していないので処理が開始されない。次の状態では，引数_0，引数_1が決定したのでAtom1，Atom2が実行可能になるが，引数_2が決定していないのでAtom3は実行されない。最後の状態では，引数_2が決定したのでAtom3が実行可能になっている。

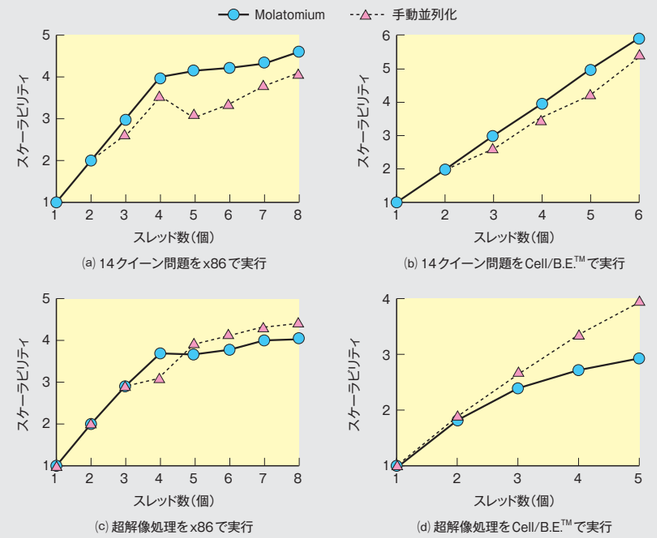

図3. Molatomiumと手動並列化の処理性能比較 — コア数の増加に比例した性能向上（スケーラビリティ）が見られるかを測定した。14クイーン問題ではMolatomiumが高いスケーラビリティを示したが，超解像処理では，手動並列化のほうが高い数値を示した。

Molはタスク並列とデータ並列の両方を記述できます。二つの関数を引数による依存関係がないように記述すると，それらの関数はタスク並列として並列に実行されます。また，互いに依存関係のないデータを配列として記述することで，それらのデータに関する処理をデータ並列として実行できます。

一方，Atomの実体はC/C++で記述する関数です。ここにはプロセッサごとに最適化されたコード，例えばSIMD（Single Instruction Multiple Data）命令などを書くことができます。

処理系は，Molの記述を専用コンパイラによってコンパイルしたバイトコードを解釈し実行します。依存関係のグラフ構造を実行時に動的に構築し，並列実行ができるAtomを個々のコアに割り当てて並列処理を実現します。

依存関係のグラフがどのように実行されるかを**図2**に示します。処理系はデータの依存関係が解決していない処理，つまり引数の決まっていない処理を先送りし，Molのバイトコードの解釈を先に進めます。その後，先送りにした処理の引数が全て決定すると，処理系はその処理を実行します。

また，Molの記述には，プロセッサ固有の記述やコア数を意識した記述は不要です。Atomの並列実行に必要な，プロセッサ固有の同期処理やコア数に応じた負荷分散などは処理系が担当するため，Molを記述するプログラマは意識する必要がありません。

最適化済みのAtomを部品として流用しながら，全体としては並列処理の性能を生かしたいといった場合，Molで依存関係を記述するだけで，高性能の並列プログラミングを得ることができます。

（注3） チェス盤上（8×8）に，8個のクイーンを互いに取られない位置に置くパズルを一般化し，N×Nの盤上にN個のクイーンを置く問題。

Molatomiumの適用例

Nクイーン問題（N=14）(注3)と，実際の製品アプリケーションである超解像アルゴリズムを使用して，Molatomiumのコア数当たりの性能評価を行いました（**図3**）。

Nクイーンの場合，手動で並列化したプログラムよりもMolatomiumで並列化したプログラムが高いスケーラビリティを示しています。

一方，超解像処理では，Molatomium処理系のオーバヘッドが顕在化しています。手動並列化プログラムは，Cell/B.E.^{TM}の専門技術者によって高度に最適化された動的スケジューリングを行っているため，高いスケーラビリティを示しており，Molatomium処理系の性能には，まだ改善の余地があると考えています。

今後の展望

これからは，より多くのコアから成るメニーコア環境や，ネットワーク接続されたコアから成る分散環境への対応が必要になると考えられます。

メニーコアや分散環境では，コアから参照できるメモリへのアクセス速度は必ずしも一定ではないといった問題などがあります。

今後は，Molatomiumの高速化とともに，これらの分散環境でも適切な負荷配分を行えるように，開発を進めていきます。

春木　耕祐
ビジュアルプロダクツ社
コアテクノロジーセンター
エンベディッドシステム技術開発部主務