A-015

並列プログラミング環境Molatomiumの実行環境軽量化

An Optimization of a Parallel Runtime for Molatomium Programming Environment

境 隆二†
Ryuji Sakai

1． はじめに

Cell Broadband Engine™‡（以降 Cell と略記）向けに最適化された並列プログラミング環境 Molatomium[1]を汎用マルチコア CPU 上で実行すると、実行環境のオーバーヘッドが相対的に大きくなり、性能がコア数に十分スケールしないという問題があった。この問題を解析したところ、実行環境におけるデータ管理と同期処理で多くの時間を費やしていた。そこで、より軽量な並列実行環境を再設計し最適化を行い実用的な性能を得ることができた。

本稿では 2 節で Molatomium のコンセプトとプログラミングモデル、実行モデルを説明し 3 節で実行環境の実装と問題点について記述する。4 節では軽量化設計と最適化について説明し 5 節で性能評価結果を示す。最後に今後の展開と課題について説明する。

2． 並列プログラミング環境 Molatomium

2.1 コンセプト

並列プログラミングは難しいと言われる一方で計算機アーキテクチャの進歩とともに成果をあげてきたのが命令レベル並列モデルである。これは 1 サイクルでより多くの命令を同時に実行するモデルであり、周波数向上のための深いパイプラインを埋めるために高い並列度の命令シーケンスが求められる。命令レベル並列モデルは、

1．コンパイラによる自動並列化が有効
2．タイミングバグが起きない
3．コーディングの工夫で並列性を向上可能

という特徴があり、コンパイラがプログラムの意味を変えないことを保障することによって並列処理の複雑度増大を抑えコーディングの工夫次第で性能を追及できるという実用性から広く使われている。この仕組みは命令をスレッドに多対応付けることでスレッドレベルの並列処理にも応用可能である（図 1）。

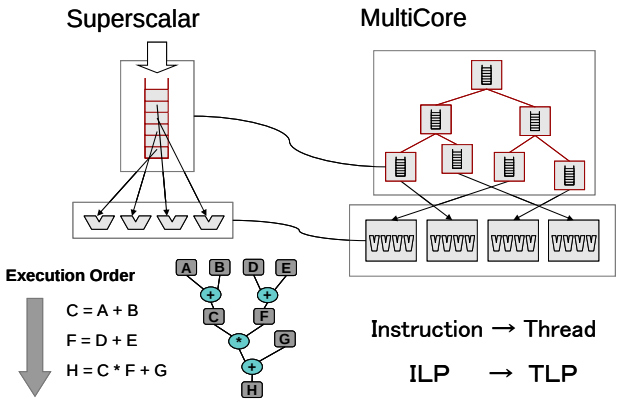

図 1 命令レベル並列からスレッドレベル並列へ

2.2 プログラミングモデル

Molatomium のプログラミングモデルは、実用性重視で設計されている。スレッド内部の逐次処理部分(Atom と呼ぶ)は C/C++言語で記述し、従来と同様に性能を追及可能である。一方、どのようにスレッドを並列で動かすかについては、独自の並列言語である”Mol 言語”によって記述する。原子(Atom)を結合し分子(Mol)を構成するというアナロジーでこのモデルを Molatomium と呼ぶ（図 2）。

Mol 言語は、宣言的な記述と手続き的な記述の両方が可能なハイブリッド言語である。宣言的な部分は関数型言語の考え方を取り入れ関数呼び出しのネスト関係でのみ実行順序が決まる。手続き的部分は上から下に逐次的にコードの解釈を行う。

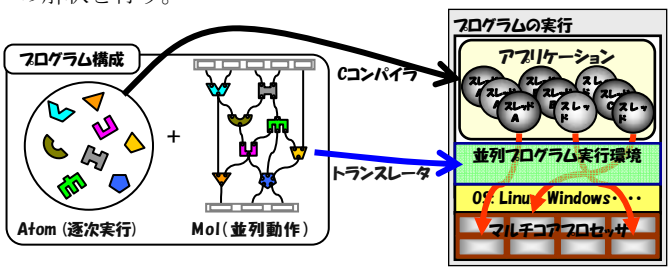

図 2 Moltomium のプログラミングモデル

2.3 実行モデル

並列動作の記述部分に独自言語を採用したのは、C 言語では関数の引数と返り値のみで実行順序を決定することが出来ないからである。プログラムの実行順序が関数の返り値と引数という明示的な依存関係のみによって決定されるというプログラム実行順序の制約を限定したことによりコンパイラと実行環境による並列動作の可能性を広げることが出来た。手続き型の逐次的なプログラム記述部分は、C 言語と同様の構文を採用し、動作モデルもほぼ同じである。違いは、入力データへの値の代入が完了していないオペレーションについては、実行が遅延され、さらに下にあるコードの実行へと処理が先に進むという点である（図 3 のイタリック部分）。

```
edge[i] := run_edge(input[i]);
ccr[i]  := run_ccr(edge[i-1], edge[i], edge[i+1]);
pocs[i] := (i%2==0)
    ? run_pocs_even(ccr[i-1], ccr[i], ccr[i+1], N)
    : run_pocs_odd(pocs[i-1], ccr[i], pocs[i+1], N);
for (i in [0..59]) {
    line[i] = pocs[i];
}
```

図 3 Mol 言語の記述例

† (株)東芝 デジタルプロダクツ＆サービス社

‡ Cell Broadband Engine™ は、株式会社ソニー・コンピュータエンタテイメントの商標

3. 実行環境の実装

3.1 バイトコードインタープリタ

Molatomium の実行モデルを直接動かせるプロセッサは存在しないので、実行モデルを実現するための仮想マシンを実装した。具体的には、各命令や関数のオペランドの値が決定していない場合は実行を遅延させ、後続の命令の解釈実行を進めるというものである。これはスーパースケーラを採用するプロセッサの実行時命令スケジューリングを広範囲にソフトウェアによって実現するものであり、Mol 言語の手続き的な部分での並列動作の仕組みである。

一方、宣言的に記述した部分のプログラム実行は、データの要求駆動で動作する（図 3 のボールド部分）。具体的には遅延式（:=で定義されるデータと式）で定義したデータをロード命令でアクセスする場合、そのエントリのデータが定義済みであれば、通常のデータロードを実行するが、未定義であればその配列のエントリを計算するための関数（遅延式の左辺）が起動される。起動された関数は Mol 言語の関数であれば上記と同様の動作を行うが、Atom であれば対応する C 言語関数を呼び出す。

3.2 実装の問題点

Molatomium の実行環境は、当初 Cell での利用を想定して実装した。Cell は OS が動作する 1 個の Power Processing Element (PPE)と、演算処理を並列に実行する 8 個の Synergistic Processing Element (SPE)で構成されており、各 SPE は 256k バイトのローカルメモリを持つ[2]。SPE で並列処理を行う場合、PPE を介して処理をスケジュールすると、OS が動作の遅延を受け入れなければならない。このため、Cell 向け Molatomium の実行環境では SPE のみで処理をスケジューリングできるようにした。具体的には、バイトコードインタープリタを各 SPE が持ち回りで処理し、実行可能な Atom を準備できれば、Atom の実行に遷移し、続きのバイトコード処理は、残りの SPE に任せるという対称性の高い実装を行った。

また、SPE でのプログラム実行は、コードとデータをローカルメモリへ転送してから実行するので SPE での処理単位はある程度大きくする必要がある。このため、実行環境のオーバーヘッドはこの DMA 転送時間に比べて十分小さければよいので Cell 向けの実装は C++で行いスマートポインタを使って内部データの管理を行った。

このような実装を汎用 CPU のマルチコアでそのまま動かすと、機能的には並列動作が可能であるが小さな Atom で構成された並列プログラムの場合、実行環境のオーバーヘッドが大きいためコアの数に比例した性能が達成できないケースがあった。

4. 軽量化設計と最適化

4.1 軽量化設計の概要

汎用 CPU コアである Intel Core™ i7† で Molatomium のオーバーヘッドを測定したところスマートポインタを介したリファレンスカウント処理とバイトコード実行毎のオブジェクト管理に時間がかかっていることがわかった。そこで軽量化設計では C++ではなく C 言語を使用しスマートポインタを使わず、直接リファレンスカウントを管理するようにした。

また、Atom や遅延オペレータを軽量に実行するコルーチンライブラリを実装して利用するようにした。これに伴い実行環境のバイトコード解釈動作がコア間で必要以上に遷移しないように工夫した。

4.2 最適化

軽量化設計において、さらなる高速化を目指してメモリ管理や共有データへのアクセスにともなう同期コストを減らすために、プロセッサのアトミック命令を使用したロックフリーアルゴリズムを採用した。また、スレッドローカルなデータ領域を確保して同期回数を削減した。

5. 性能評価

4 節で説明した軽量化設計と最適化をしたものを CPU 版の超解像処理に適用して性能評価を行った（図 4）。従来の実行環境（OldMol）では TBB と比べて性能が劣っていたが、今回実装したもの（NewMol）は、TBB よりも高い性能が得られた。これは、超解像処理の各フェーズの並列度が十分でない場合でも、Molatomium では次のフェーズの処理へと進むことが可能な点で優位性があるためと考えられる。今後、より詳細なデータを採取して分析をする予定である。

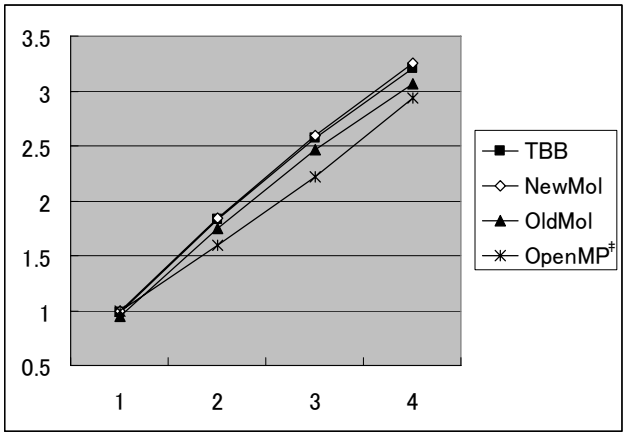

図 4　Core™i7 上での超解像処理の性能比

6. おわりに

今回の汎用 CPU 向け高速化開発は第一段階のものであり、更なる高速化を目指して現在も開発を継続している。Molatomium の実行モデルは局所的には深さ優先でプログラムの依存グラフを探索するのでデータアクセスの局所化が期待できる。今後は、キャッシュサイズの小さい組込み向けのマルチコアプロセッサでも評価を行い、その効果を確認していく予定である。

参考文献

[1] 高山征大他, ”並列プログラミングモデルMolatomium” IPSJ PRO Vol.3 No.1, 2010
[2] F. Pham et al., ”The Design and Implementation of a First-Generation CELL Processor” ISSCC Dig. Tech. Papers, Paper 10.2, pp. 184-185, Feb., 2005.

† Intel、Core™ は、Intel社の商標、登録商標

‡ OpenMPはOpenMP Architecture Review Boardの登録商標