小山工業高等専門学校研究紀要
第 44 号 (2011) 121－126

OpenMPを用いたマルチコア並列プログラム

南斉 清巳*1, 青山 直暉*2

OpenMP Parallel programming on Multi-Core Processor

Kiyomi NANSAI, Naoki AOYAMA

Nowadays, CPU is aiming at improvement in throughput by multi-core-ization instead of improvement in a clock-rate. Although the merit of multi-core CPU is useful enough in the normal use, parallel programming becomes essential in order to pull out the performance of multi-core CPU further. Also in programming education, it will be necessary to take in the approach to the parallel programming, and the practical programming method from now on. We developed the teaching materials for parallel programming which are suitable in order to make students experience parallel programming.

KEYWORDS : parallel programming, multi-core CPU, OpenMP

1. 目的

最近のCPUはクロックの高速化に替えて、マルチコア化により処理能力の向上を図っている。通常の使用においても十分マルチコアCPUのメリットを享受できるが、マルチコアCPUの性能をさらに引き出すためには並列プログラミングが必須となってくる。プログラミング教育においても、今後、並列プログラミングの考え方や、具体的なプログラミング方法を取り入れていく必要があるだろう。ここでは並列プログラミングを簡単に学生に体験させるために適した、並列プログラミング教材を開発することを目的としている。

2. 開発環境

本研究のプログラム開発及び実行使用したハードウェア及びソフトウェアを次に示す。Intel Parallel Studioとは、Windows Visual Studioに統合して使用するC++用のコンパイラ・ツールセットである。これによりコンパイルならびに実行及びデバッグや処理高速化の評価など全てが行える。今回使用したIntel Parallel Studio付属のコンパイラはOpenMP 2.5に対応している。

【使用したハードウェア及びソフトウェア】
○ CPU ：Intel Core i5 (2.67GHz, Quad-Core)
○ Memory：2GB
○ OS：Windows 7 Professional (64bit)
○ OpenMP 2.5
○ 並列化コンパイラ：Intel Parallel Studio
○ Windows Visual Studio 2008 C++

*1 電子制御工学科(Dept. of Electronic Control Engineering) E-mail:nansai@oyama-ct.ac.jp
*2 平成23年3月電子制御工学科卒業

3. OpenMP による並列化の特徴

OpenMp は共有メモリモデルによる並列化を記述する API(Application Program Interface)であり、プログラム中に並列実行や同期を指示する指示文を記述することで、比較的容易に並列プログラムを作成することが出来る。また、コンパイルオプションにより並列化指示文を無視させることが出来る。OpenMP に対応していないコンパイラの場合には並列化指示文は単なるコメントと見なされる。OpenMP の特徴をまとめると下記のようになる。

（1） 並列プログラムが比較的簡単に短いコードで記述できる
（2） 異なるシステム間でソースプログラムが共通化でき移植性が高い
（3） 逐次プログラムを段階的に並列化していくことが出来る
（4） 並列プログラムと逐次プログラムを同一ソースプログラムで管理できる
（5） コンパイラの自動並列化機能よりも高速化が可能である

4. 作成プログラムと評価方法

並列プログラムの入門用として、簡単かつ興味を惹くようなプログラムを取り上げることとした。簡単な並列計算プログラムとして「N 以下の素数の個数を求めるプログラム」を作成した。より実際的で興味を惹きそうなプログラム例として、画像処理プログラムを作成した。評価方法としては、並列化前の実行時間と並列化後の実行時間を計測し、並列化前の実行時間を並列化後の実行時間で割ったものを高速化指数と定義し、どの程度、処理の高速化が図れたのかを確認することとした。

【作成したプログラム】

（1） N 以下の素数の数を求めるプログラム
（2） 画像のネガ化プログラム
（3） 画像のグレースケール化
（4） 画像のエッジ検出（差分法）
（5） 画像のエッジ検出（ゼロ交差法）

5. 結果

5.1 N 以下の素数の数を求めるプログラム

実行してから結果を出力されるまでの時間を並列化前後についてまとめたものを表１に、並列化前を１とした場合の処理速度の違いをグラフに表したものを図１に示す。

なお、N の値が１００万以下の場合は処理時間が１[ms]よりも短いため今回は測定していない。

表１　並列化前後における実行時間[sec]

繰返し回数	10^8	10^9	10^10	10^11
並列化前 T１	0.39	10.12	267.43	7282.77
並列化後 T２	0.11	2.59	68.95	1965.52
高速化指数 T１/T２	3.55	3.91	3.88	3.71

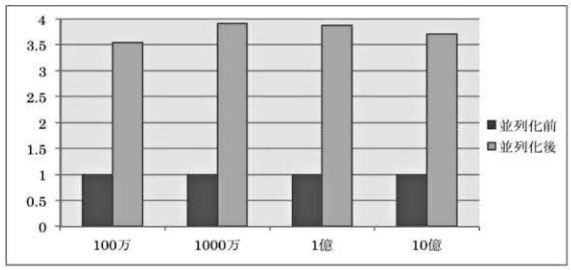

図１　並列化前を１とした場合の処理速度の違い

上記の結果、全体を通して並列化前に比べ４倍近い処理速度の向上が図れていることがわかる。このことは、使用している CPU のコア数が 4 であることと一致している。

5.2 画像処理プログラム

使用した画像の元データの解像度は２３０４×３０７２で、画像容量は２０.２MB である。図２に使用した画像を示す。それぞれのエフェクト処理の実行時間と処理速度の向上比についてまとめたものを以下に示す。

図２　使用した元画像（2304×3024 フルカラー画像）

図４　ネガ化処理後の画像

5.2.1 ネガ化処理プログラム

実行してから結果を出力するまでの時間を並列化前後についてまとめたものを表２に、並列化前を１とした場合の処理速度の違いをグラフに表したものを図３に、ネガ化処理を施した後の画像を図４に示す。

この結果よりネガ化処理については3.39倍ほど高速化出来たものの、ファイルの入出力時間が処理全体の大部分を占めるため、全体として実行時間が並列化前より倍以上かかっている。

表２　ネガ化処理プログラムの実行時間[sec]

	Read Time	Effect Time	Write Time	Total Time
並列化前 T１	4.04	0.78	4.46	9.28
並列化後 T２	10.34	0.23	10.11	20.69
高速化指数 T１/T２	0.39	3.39	0.44	0.45

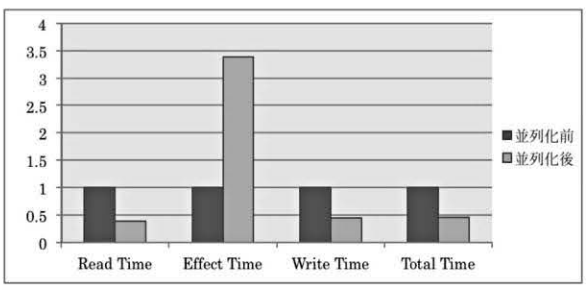

図３　処理速度の違い(ネガ化)

5.2.2 グレースケール化処理プログラム

実行してから結果を出力するまでの時間を並列化前後についてまとめたものを表５−３に、並列化前を１とした場合の処理速度の違いをグラフに表したものを図５に、グレースケール化処理を施した後の画像を図１−６に示す。

この結果よりネガ化処理同様、エフェクト処理については 3.48 倍ほど高速化出来ているが、全体としては並列化した方が却って時間がかかっているのが見て取れる。

表３　グレースケール化処理プログラムの実行時間[sec]

	Read Time	Effect Time	Write Time	Total Time
並列化前 T１	4.01	0.80	4.52	9.33
並列化後 T２	10.17	0.23	10.31	20.72
高速化指数 T１/T２	0.39	3.48	0.44	0.45

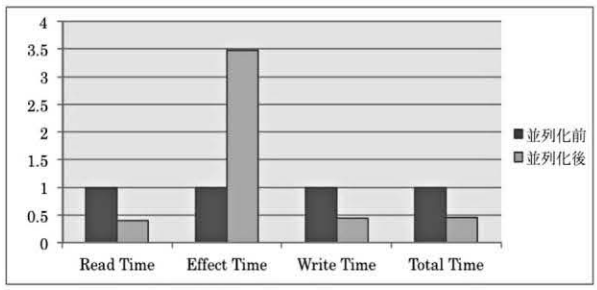

図５　処理速度の違い(グレースケール化)

図６　グレースケール化処理後の画像

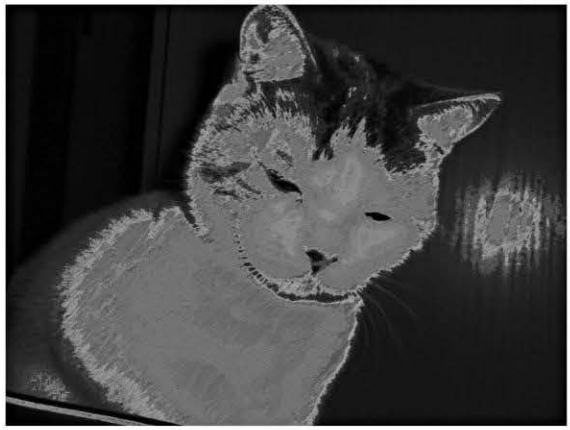

図８　差分法によるエッジ検出後の画像

5.2.3 エッジ検出プログラム（差分法）

実行してから結果を出力するまでの時間を並列化前後についてまとめたものを表４に、並列化前を１とした場合の処理速度の違いをグラフに表したものを図７に、差分法によるエッジ検出処理を施した後の画像を図８に示す。

この結果より前述の二つのエフェクト同様、エフェクト処理については3.32倍ほど高速化できているが、全体としては並列化した方が却って時間がかかっているのが見て取れる。

5.2.4 エッジ検出プログラム（ゼロ交差法）

実行してから結果を出力するまでの時間を並列化前後についてまとめたものを表５に、並列化前を１とした場合の処理速度の違いをグラフに表したものを図９に、ゼロ交差法によるエッジ検出処理を施した後の画像を図１０に示す。

この結果よりエフェクト処理については 3.61 倍ほど高速化できている。また、エフェクト処理時間に対して入出力時間の割合が小さいため、全体としての処理速度も向上している。

表４エッジ検出プログラム（差分法）の実行時間 [sec]

	Read Time	Effect Time	Write Time	Total Time
並列化前 T１	3.99	3.32	4.48	11.79
並列化後 T２	10.41	1.00	10.83	20.72
高速化指数 T１/T２	0.38	3.32	0.41	0.57

表５エッジ検出プログラム（ゼロ交差法）の実行時間[sec]

	Read Time	Effect Time	Write Time	Total Time
並列化前 T１	4.18	858.36	4.71	867.26
並列化後 T２	9.95	237.78	11.70	259.43
高速化指数 T１/T２	0.42	3.61	0.40	3.34

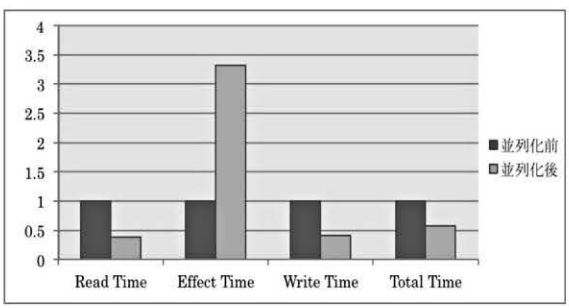

図７　処理速度の比較(差分法によるエッジ検出)

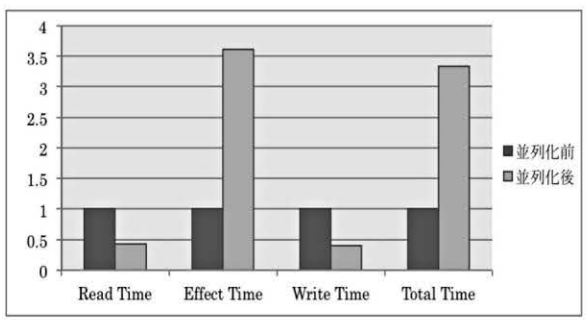

図９　処理速度の比較(ゼロ交差法によるエッジ検出)

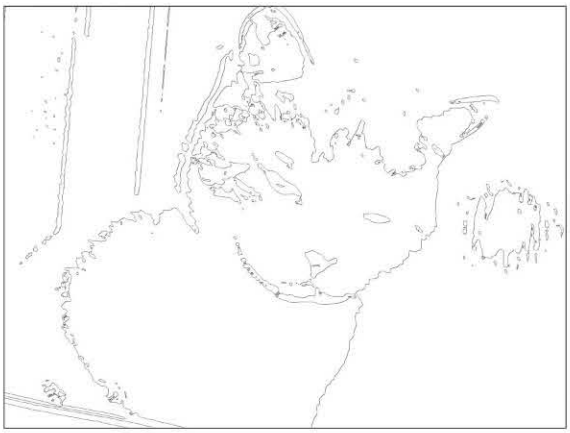

図１０　ゼロ交差法によるエッジ検出後の画像

6. 結論

OpenMP を使用することにより、従来の逐次処理プログラムに並列化指示文を追加していくことで並列化プログラムを書くことができる。始めて並列プログラムを学習するものにとってはわかりやすく興味を示すプログラムを作成することが出来た。また、同一ディスクに対する入出力処理は並列化することによりオーバーヘッドが起こり、却って時間がかかってしまうことが体験できた。今後はより並列化のメリットやデメリットが表れやすいプログラムを作成し、並列プログラム学習教材を充実させていきたい。

参考文献

１）池井　満、林　浩史、田中　智子、「インテル Parallel Studio プログラミングガイド」（株）カットシステム、2009 年

２）菅原　清文、「C/C++プログラマーのための OpenMP 並列プログラミング」、(株) カットシステム、2009 年

３）昌達　啓仁、「詳解　画像処理プログラミング」、（株）ソフトバンククリエイティブ、2008 年

４）インテルコンパイラー　OpenMP 入門、インテル株式会社、2006

【受理年月日　2011年 9月30日】