Cerium による並列処理向け I/O の設計と実装

085726C 古波倉正隆 指導教員：河野真治

1 研究背景と目的

近年、CPU 1 コア当たりのクロック数が頭打ちとなっているため、シングルコアでの処理能力はほとんど上がっていない。それを解決した結果、シングルコアからマルチコアへの移行によって CPU 性能が向上している。しかし、マルチコア CPU を最大限に活かすためには、プログラムの並列度を向上させなければならない。そこで当研究室では、並列プログラミング用フレームワーク、Cerium 及び Cerium Task Manager の開発を行い、提供することによって並列プログラミングを容易にしている。

先行研究による Task の並列化によって、プログラム全体の処理速度は飛躍的に向上しているが [1] 、ファイル読み込み等の I/O 処理と Task が並列で動作するようには実装されていない。

本研究では I/O 処理 と Task が並列に動作するような設計、実装によってプログラム全体の並列度、及び処理速度を上げていく。

2 Cerium Task Manager

Cerium Task Manager は、並列処理を Task 単位で記述する。関数やサブルーチンを Task として扱い、その Task に対して Input Data、Output Data 及び依存関係を設定する。そして、それに基づいた設定の元で Task Manager に管理され実行される。本稿で述べる Input Data とは、検索対象となるテキストファイルのことである。

Cerium Task Manager は PlayStation 3/Cell、Mac OS X 及び Linux 上で利用することができる。

3 I/O を含む Task の概要

ファイルを読み込んで一定の大きさでファイルを分割し (File Read)、それらに対してそれぞれ文字列検索等の処理 (Task) を行う。そしてそれぞれの処理から返されたの結果 (Output Data) を最後に集計をして結果を返す (Result Task)。(図 1)

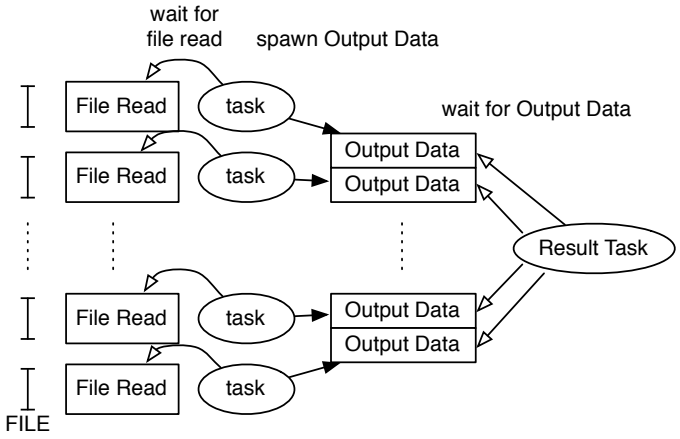

図 1: I/O を含む Task

4 並列処理向け I/O の設計と実装

4.1 mmap での実装の問題点

先行研究では mmap によるファイルの読み込みを行っていた。mmap でファイルを読み込むタイミングは、mmap 関数が呼ばれたときではなく、mmap した領域に対して何らかのアクセスをしたときに初めてファイルが読み込まれる。つまり、分割された Task は文字列検索をすぐに行うのではなく、文字列検索を行おうとした時に初めてファイルが格納される。Task は複数一斉に実行されることが望ましいが、mmap ではそれぞれの Task で読み込みが起こってしまうため、I/O ネックによる Task の待ちが発生する。

4.2 Blocked Read の設計と実装

Blocked Read とは、あるサイズずつで読み込む処理と、それらに文字列検索を行う処理を分離させるための実装方法である。この方法では、読み込み専用の Blocked Read と、文字列検索を行う Task Blocks を別々に生成し処理を行う。Read Task はファイル全体を一度に読み込むのではなく、ある程度の大きさで分割を行い、読み込みされ次第それぞれの文字列検索が行われる。

Task は 1 つずつ起動すると、起動した Task でメモリを圧迫してしまうため、Task を複数まとめたブロック単位で起動を行う。この 1 つのブロックで処理されるテキストファイルを、Blocked Read で読み込んでいき、読み込みが終わったら読み込まれた範囲の Task Blocks を起動する。

もし、Blocked Read で読み込まれる前にその範囲を担当する Task が起動してしまうと、正しい結果が返ってこない。それを防止するために、Task Blocks は必ず Blocked Read が行われてから起動するように wait をかけている。(図 2)

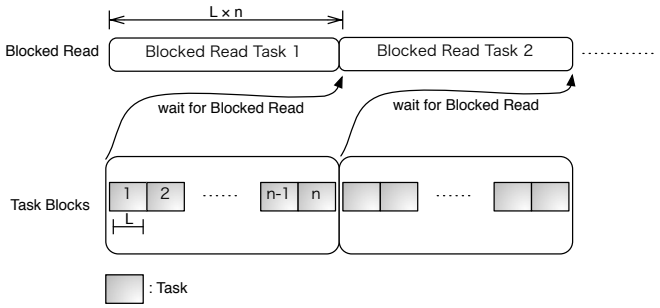

図 2: Wait for Blocked Read

4.3 I/O 専用 thread の実装

Cerium Task Manager では Task 単位で CPU Type の設定を変更することができる。SPE_ANY という Type を設定すると、Cerium Task Manager 側が自動的に CPU を割り振る。しかし、今回の実装でこの Type を使用してしまうと、Blocked Read Task に割り込んで Task が割り振られてしまう問題がある。その問題を解決するために、IO_0 という I/O 専用の thread を実装した。この Thread は I/O を最優先に実行されるようにチューニングを行った。(図 3)

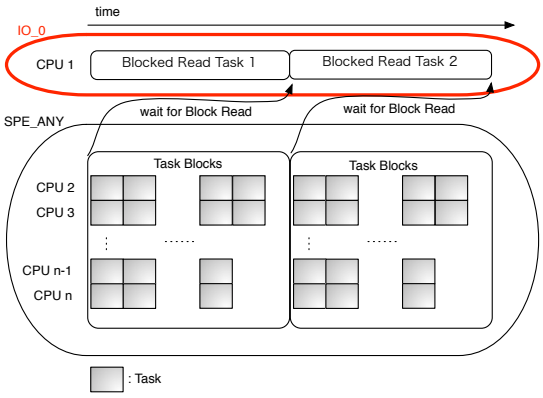

図 3: IO_0 での実装時

5 ベンチマーク

- Mac OS X Mavericks (10.9.1)
- HDD 1TB、Memory 16GB、CPU 2*2.66 GHz 6-Core Intel Xeon
- CPU NUM 12
- 10GB のファイルに対して Booye-Moore String Search をかけ、検索文字列がいくつ含まれているのかカウント
- 測定はファイルの読み込みから結果が返ってくるまでの時間

以下の表 1 に実行結果を示す。

読み込み方法	平均実行速度 (s)
mmap	154.6
一括 Read	114.9
Blocked Read & SPE_ANY	106.0
Blocked Read & IO_0	99.2

表 1: 実行結果

表 1 より、mmap より Blocked Read & SPE_ANY の実行速度が 31% 改善された。また、Blocked Read の CPU Type も SPE_ANY から IO_0 に変更することによって更に 4 % の改善が見られた。これより、I/O を含む並列処理を行う場合は、mmap で実装することによって自動的に読み込ませるのではなく、自分自身で読み込みを制御したほうが速くなることがわかる。

6 まとめ

mmap で I/O を含む Task を実装するとき、mmap した領域に対して何らかの処理が加わった時にしか読み込みが行わないので、それぞれの Task に読み込みを任せてしまうことになる。それを解決する方法として、Blocked Read と Task を並列に行う実装を行った。また、Blocked Read が、他の Task に割り込まれないように改善した結果、35 % の実行速度の改善が見られた。本研究より、I/O を含む Task は、チューニング次第で更に改善する余地があると考える。

参考文献

[1] 金城裕、河野真治、多賀野海人、小林佑亮 (琉球大学) ゲームフレームワーク Cerium Task Manager の改良 情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), April 2011