並列化プログラミング技術研修報告

2010/01/08 湊 真明

昨年の 12 月 11 日と 12 月 22 日に開催された「今から知っておきたい並列化プログラミング実践研修」なるセミナーに参加してきた。そこで得た有益だと思われる情報について記述する。

セミナーは並列化プログラミング、つまりスレッド処理をプログラムに適用する手法を C 言語のサンプルを使って解説するというものだった。無料参加のセミナーで、かつ Intel 社員の方が講師だった為にボトルネックの検出やパフォーマンス測定に逐一 Intel 社製品(オープン価格)が登場したが、ここでは環境・言語に依存する事項や Intel 社製品についてはなるべく触れないように並列化プログラミングの意義と並列化プログラミングの基本的な流れについて、聞いてきたことを簡単にまとめる。

まず、プログラムを並列化することによってどのような恩恵が得られるかだが、CPU 資源を有効に活用する事によって処理時間の短縮が可能となる。CPU 資源の有効活用についてだが、並列化を行なっていないプログラムは CPU のコアを１つしか使用できない。最近はマルチコアの CPU が当たり前になっており、シングルコアの CPU で使用率 100%になる並列化していないプログラムをデュアルコアの CPU で走らせても処理にコアを１つしか使用できず、CPU 使用率は 50%までしか上がらない。残りの 50%は遊んでいる状態になってしまい、非常に勿体無いと言える。Intel の人曰く、これからも CPU のコア数はガンガン増える(32 コア CPU 実用化間近)そうで、プログラムの並列化とそれをサポートする Intel のツールの重要性は高まる一方であるとのことだった。

プログラムの並列化は、以下の流れで行なう。

１．並列化する部分の検討
２．設計
３．コーディング
４．検証
５．結果に不満があれば２に戻る

１の並列化する部分の検討について。まずプログラムのうち、ボトルネックとなっている部分を判明させ、その処理が並列化可能なものであるかどうかを考えなくてはならない。プログラムの並列化には

・各スレッドに全く別の処理を割り当てる
・繰り返し行なう処理をスレッドで分担して行なう

(例：1 から 100 までの数に含まれる素数を取り出す、という処理でスレッド 1 には 1 から 50 までについて素数であるかどうか判定をさせ、スレッド 2 には 51 から 100 までについて素数であるかどうかを判定させる)

という２つのパターンがあるが、各スレッドの処理が終了する順番は決まっていない為、「１つのスレッドの処理結果が別のスレッドの処理結果に影響してはならない」という制約がある。この制約をパスしていないプログラムは並列化を行なえない。

次にそのプログラムは本当に並列化する価値があるかを検討する。並列化によりどれくらいの性能向上が見込め、作業にどのくらいの労力が必要になるかを見極め、労力に見合った成果が得られるかを考える必要がある。

２の設計について。これは各スレッドの負荷がなるべく均等になるように考慮しなければならない。例えば、先程例として挙げた素数判定のプログラムなどは判定対象の数字が大きいほど計算回数が増える。そのため、実際にスレッドを２つ生成して片方に 1 から 50 までを、もう片方に 51 から 100 までを処理させるというように設計すると、1 から 50 までの処理をするスレッドの方が早く終了する。51 から 100 の処理が終了するまでの間、1 から 50 までの処理を割り当てられていたリソースは遊びの状態となるため、結果最良のパフォーマンスが得られない。

３のコーディングについて。プログラムの並列化をしようとすると、どうしてもコードが長く複雑になる傾向があるため、コードを単純化できるツールがあるならどんどん使用していく事が望ましい。セミナーで使用されたのは Open MP という規格で、スレッドに関する処理を比較的単純なコードで実現でき、C、C++、Fortran に対応している。Open MP を使って書かれたソースをコンパイルするためには Open MP 対応のコンパイラが必要になるが、最近は gcc や Visual Studio 付属のコンパイラなどの有名どころが Open MP 対応になってきているとのことだった。

４の検証について。第一に結果が正当なものであるか。スレッドは各自好き勝手に処理を行なうので、処理対象の割り振りや同期のタイミングを適当にすると、すぐに毎回違う結果を返してくるプログラムが誕生してしまう。同じ処理をして毎回結果が違うというのであればわかりやすいが、再現の難しいケースも起こりうるので実行結果からのプログラムの妥当性を立証することはなかなか難しい。

第二に期待した性能向上が得られたか。これはプログラムの処理時間を計測すれば良い。

第三に各コアにきちんと処理の振り分けができているか。並列化したはずなのにパフォーマンスを見ると１つの CPU だけ負荷が高く他の CPU は使用率が低いままだったりするのは、失敗と言える。

期待通りのパフォーマンスが得られていない、もしくは結果の妥当性が損なわれているようであれば設計見直しとなる。

以上がセミナーで聞いてきた並列化プログラミングの流れになる。

セミナー参加のお土産として、表紙が動物なことで有名なオライリーの C++スレッドプログラミング技術書を頂いた。会社の本棚に置いておくので、興味のある方は本社に寄った際に目を通してみてはどうかと思う。

以上