

Cellスピードチャレンジ2007 規定課題参加報告

花岡俊行 (京都大学 情報学研究科)



参加の動機

並列処理を制御するプログラムを書いたことがなかったこと、Cellについて詳しく知らなかったことから、初めは参加するのを躊躇していましたが、しかし、申し込みの締め切り間際に研究室の先輩から勧められ、どこまでできるかは分からないけどやってみようという気になりました。並列処理には興味を持っていたため、並列計算により高速にデータを処理することを競うこのコンテストは、並列処理に実際に触れる良い機会だと考えました。

課題

規定課題では、規定の課題をどれだけ高速に解けるかを競います。今回の課題はソーティングでした。ソーティングと聞くと簡単に思われるかもしれませんが、ソーティングの対象となる要素の長さが固定ではなく、そこがアルゴリズムの実装を少し複雑なものにしています。以下に問題の仕様の概略を示します。

- 対象となるデータのそれぞれの要素は float 1つ分のキーと、float 要素のリスト (図-1 参照)
 - リストの長さはプログラム開始時にパラメータとして与えられる
 - 要素のキーはプログラム開始時には 0 で初期化されている
 - キーの計算法はパラメータとして与えられる (二乗和 or 最大値)
- 使用できるメインメモリには制限があり、以下の2つの場所以外のメインメモリへはアクセスできない
 - 入力データの2倍の容量の配列 (プログラムの開始時に入力データが前半部に格納されている)
 - 容量 16 キロバイトの配列
- ソーティングの終了時点で、メインメモリの配列の前半部にデータがソートされた状態で格納されているようにする
 - 要素には、計算したキーが書き込まれている必要がある

入力データ全体 (float 配列)

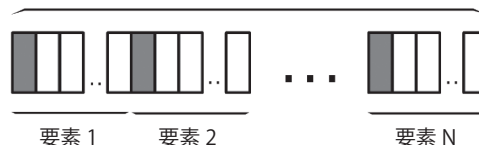


図-1 入力データ

- コンテストは予選と決勝からなり、予選の結果を見て決勝までの間にチューニングをすることができる

課題へのアプローチ

課題を始めるにあたり、まずツールキットに付属のサンプルプログラムを読んでみました。その結果次のようなことが分かりました。

- データを8つの領域に分割し、それぞれをソーティングし、最後にバイトニックソートでまとめている
- それぞれのソーティングにはクイックソートを用いている
- 要素の移動には、要素1つ単位で1回のメモリ転送が行われている
- 7つ使えるSPEのうち、4つしか使っていない

このサンプルプログラムの方法を基に、SPEを7つ効率的に使い、またメモリ転送命令をできるだけ減らすことによって高速化しようと考えました。

まず、全体のデータを16の部分領域に分け、その処理を各SPEに割り振ることによって並列処理を実現します。

また、メモリ転送を減らすために、要素そのものでなく、データのキーとアドレスからなるインデックスを作成し、それをソートするということを考えました。作成したインデックスはメインメモリに格納するのですが、インデックスのサイズはキーとアドレスで float 2つ分になり、一方元の要素はリストの長さの制限により、最小でも float 4つ分のサイズになります。なので、インデックスは元の配列の作業領域を多くても半分しか使わないので、インデックスに対する処理のための作業領域も確

保することができます。この考えを実装したプログラムを予選に提出しました。プログラムの流れは、

1. 全体のデータを16の部分領域に分割し、インデックスを作成し、それぞれをクイックソートでソーティング
2. それぞれのソート済み領域を、バイトニックソートで統合
3. ソートされたインデックスに従ってデータを移動となっています（図-2 参照）。

予選の結果は1位となっていました。配点は決勝の方が多いため安心はできません。結果を詳しく見てみると、リスト長が大きい入力に対しては作成したプログラムがよい成績を残しているのに対し、リスト長が小さなものではあまり結果が芳しくなく、いくつかのチームに大幅に負けているということが分かりました。小さな要素に対してはインデックスを作成しても、データ転送命令にかかる時間を減らすことはできないということが原因です。

とはいえ、決勝までの間に全体の構造を変える余裕はなさそうだったので、方針は変えず、チューニングに専念することにしました。余裕があればデータが小さいときは別に作った処理に任せようと思いました（実際にはできませんでした）。決勝に向けて行ったチューニングの主なものとしては、

- 部分領域のソーティングをマージソートで行うようにした
 - データ転送にダブルバッファリングを用いた
- というものがあります。特に部分領域のソーティングでは、SIMD 命令を活用したソーティングも利用し、高速化に成功しました。SIMD 命令は float4 4 分のデータをまとめて1命令で操作できる命令で、これによりインデックス2つを同時に扱うことができます。

決勝でも幸い1位を獲得できました。結果を見ると、やはりリスト長の大きな入力に強く、小さなものに対しては弱いという傾向は変わっていませんでしたが、決勝に向けて行ったチューニングのおかげでなんとかいい結果を残すことができたのだと思います。

Cell スピードチャレンジはとてもやりがいがありました。高速化するポイントが数多くあり、少しずつ速くなっていくのが楽しかったです。このような機会を設けてくださった関係者のみなさまに感謝したいと思います。ありがとうございました。

（平成19年9月23日受付）

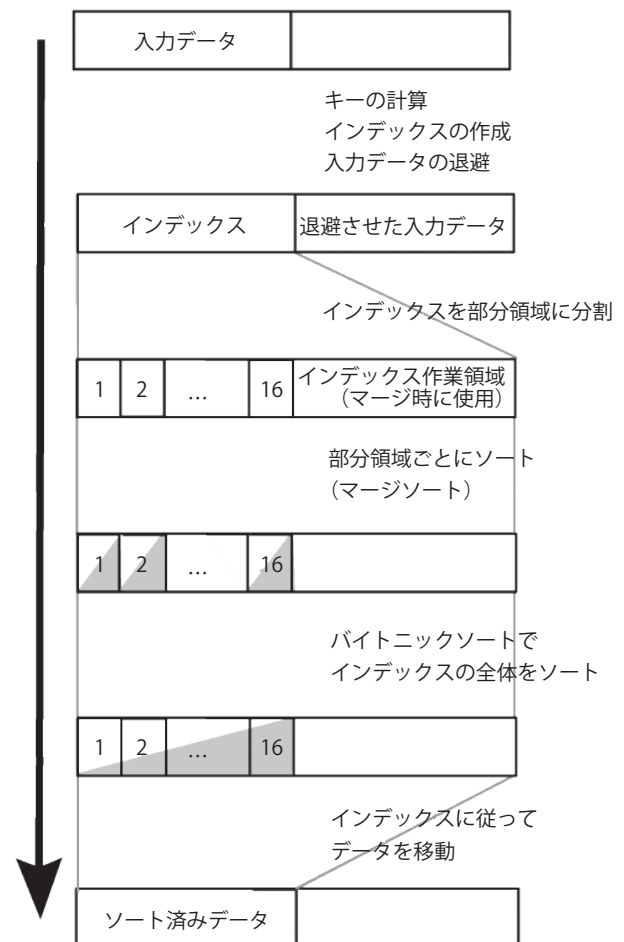


図-2 処理の流れ

花岡俊行
hanaoka@kuis.kyoto-u.ac.jp

平成19年京都大学工学部情報学科卒業、現在、同大学院情報学研究科修士課程在学中。