称号及び氏名	博士（理学）　河南　克也
学位授与の日付	平成 27 年 3 月 31 日
論　文　名	An Efficient Implementation of the Longest Common Subsequence Algorithm with Bit-Parallelism on GPUs
論文審査委員	主査　藤本　典幸 副査　馬野　元秀 副査　瀬田　和久

論文要旨

An Efficient Implementation of the Longest Common Subsequence Algorithm with Bit-Parallelism on GPUs

河南　克也

【研究概要】

本論文の研究テーマは GPU **を用いて並列計算を行うことにより，2 つの文字列の間の最長共通部分列（Longest Common Subsequence，以下 LCS）を高速に求める手法の提案，および提案手法に基づくプログラムの作成と評価実験**である．LCS とは 2 つの文字列の類似度を表す指標の 1 つであり，DNA 配列の比較・文字列のあいまい検索・スペルチェック等に応用されている．

LCS は**部分列（subsequence）**を用いて定義される．部分列は非形式的には，文字列から 0 個以上の任意の数の文字を，順序を入れ替えることなく取り出したものと言える．例えば

文字列 COMPUTER から 1, 4, 5 文字目を取り出すと文字列 CPU が得られるので, CPU は COMPUTER の部分列である．部分列の形式的な定義を与えるために文字列 $S=s_1s_2\cdots s_p$ と文字列 $A=a_1a_2\cdots a_m$ を考える．このとき，S の添字集合 $\{1,2,\cdots,p\}$ から A の添字集合 $\{1,2,\cdots,m\}$ への写像 F が次の条件 C1, C2 を満たすなら，S は A の部分列であるという．

C1: $F(i)=k$ は $s_i=a_k$ の必要十分条件である

C2: $i<j$ ならば $F(i)<F(j)$ である

ただし，空列（長さ 0 の文字列）は任意の文字列の部分列とする．文字列 A の部分列であり，かつ文字列 B の部分列でもある文字列を，A と B の**共通部分列**（common subsequence）と定義する．A と B の共通部分列の中で最長のものを，A と B の LCS と定義する．なお，任意の A, B に対して LCS は必ず存在するが，複数存在することもある．LCS 問題ではいずれか 1 つを見つければ良い.

2 つの文字列 A と B が入力として与えられたとき，A と B の LCS の 1 つを計算する再帰的アルゴリズムが Hirschberg によって提案されている．Hirschberg の LCS アルゴリズムは再帰呼び出し 1 回につき, 動的計画法に基づいて LCS **の長さ**（Length of LCS, **以下** LLCS）を計算するアルゴリズムをサブルーチンとして 2 回呼び出している． 動的計画法に基づいて LLCS を計算するアルゴリズムは $O(mn)$ の時間計算量と $O(m+n)$ の空間計算量で LLCS を計算する．この LLCS 計算の処理が Hirschberg の LCS アルゴリズムを時間計算量的・空間計算量的に支配している部分であるので，この部分を高速化することを考える.

LLCS 計算を高速化する手法としては，ビット並列計算を用いたアルゴリズムが Crochemore らによって提案されている． Hirschberg の LCS アルゴリズムの中の LLCS 計算を Crochemore らのビット並列 LLCS アルゴリズムに置き換えると，コンピュータのワードサイズを w とした場合，$O(mn/w)$ の時間計算量と $O(m+n)$ の空間計算量で LLCS が計算できる（なお，開発・評価実験に用いた環境では $w=32$ となる）．しかし DNA 配列等の比較では例えば 500 万文字以上の文字列を扱うこともあるので，更なる高速化が必要となる.

そこで本論文では LLCS 計算をビット並列アルゴリズムで置き換えた Hirschberg の LCS アルゴリズムを，GPU を用いた並列計算で高速化する手法を提案する．この提案手法は $O(mn/w)$ の時間計算量と $O(m+n)$ の空間計算量で LCS を求める．また，提案手法に基づいた GPU プログラムを実際に作成し，評価実験としてシングルスレッド版の CPU プログラム（CPU の 1 コアのみを使用）・マルチスレッド版の CPU プログラム（CPU の複数コアを使用）・既存 GPU アルゴリズムとの比較を行った.

GPU では再帰呼び出しの回数が最大で 50 回程度と制限されているために，提案手法では再帰呼び出しは CPU 側で処理し, 再帰呼び出しの中で呼び出される LLCS 計算のサブルーチン（Crochemore らのビット並列 LLCS アルゴリズム）のみを GPU 側で処理している．Crochemore らのアルゴリズムではビット列に対する演算として，論理積（AND）・論理和（OR）・否定（NOT）・算術加算の 4 種類を用いる．このうち論理積・論理和・否定はビッ

ト単位の並列性があるので容易に並列化ができる．一方で算術加算には桁上げが存在する．桁上げは下位から上位に伝播する可能性があるため，通常は下位から遂次的に計算していくことになる．しかし桁上げの伝播を待っていると並列性が低くなってしまうため，GPU を用いた計算には適していない．算術加算の並列性を高めるには工夫が必要である．そこで提案手法ではビット列を 1024 ビット毎に部分ビット列に分割し，各部分ビット列を GPU のブロックに割り当て，並列に計算できる箇所を増やすことで LLCS 計算を高速化している．また，各ブロック内で 1024 ビットの算術加算を行うときには**条件求和加算**と呼ばれる，主にハードウェアの加算器の並列化に使われる手法をソフトウェアに適用することで高速化を行っている．

提案手法では高速化のために様々な工夫を行っている．工夫の 1 つは入力文字列長が短くなった場合の対策である．提案手法では再帰呼び出しを繰り返していくにつれ，LLCS 計算に入力として与えられる文字列の長さは短くなっていき，計算時間よりも CPU・GPU 間のデータ転送に必要な時間の方が長くなってしまう．そのような場合には CPU 側で LLCS を計算した方が速いので，適切な閾値を設定して閾値よりも文字列長が短い場合には CPU 側で計算するようにしている．また，CPU・GPU 間のデータ転送は CPU 同士や GPU 同士でのデータ転送よりも時間がかかるので，極力 CPU・GPU 間のデータ転送量を減らす工夫をしている．

本論文の評価実験では CPU に Intel Core i7 920（2.67GHz）を，GPU に GeForce GTX 580 を使用している．提案手法に基づく GPU プログラムは，シングルスレッド版の CPU プログラムに比べて最大で 12.81 倍（入力文字列 A が 1000 万文字，B が 990 万文字の場合）高速に LCS を求めることができた．また，OpenMP（CPU の複数コアを用いた並列計算を行うためのライブラリ）のタスク並列処理を用いて作成したマルチスレッド版の CPU プログラム（使用した CPU は 4 コア）に比べて最大で 4.56 倍（入力文字列 A が 1500 万文字，B が 870 万文字の場合）高速に LCS を求めることができた．なお，Kloetzli らの既存 GPU アルゴリズム（動的計画法に基づいており，ビット並列性は使用していない）に対しては，同一の GPU（GeForce 8800 GTX）を用いた場合に 10.9～18.1 倍高速に動作した．

上記の結果に加えて，提案手法に基づくプログラムの計算時間全体に対する GPU 側の計算時間・CPU 側の計算時間・データ転送時間それぞれの占める割合も測定した．その結果，計算時間全体の 95.53%～98.89%を GPU 側の計算時間が占めており，CPU 側の計算時間は高々3.11%，データ転送に要する時間は高々1.13%であることが分かった．

【学位論文草稿の基礎となる論文】

雑誌：A GPU Implementation of a Bit-Parallel Algorithm for Computing the Longest Common Subsequence, Katsuya Kawanami and Noriyuki Fujimoto, The Information Processing Society of Japan (IPSJ) Transactions on Mathematical Modeling and Its Applications (TOM), Vol.7, No.2, pp.36-44 (2014)

会議録：GPU Accelerated Computation of the Longest Common Subsequence, Katsuya Kawanami and Noriyuki Fujimoto, Proceedings of the International Conference Facing the Multicore Challenge Ⅱ, Lecture Notes in Computer Science, Vol.7174, pp.84-95, Springer-Verlag (Karlsruhe, Germany, 2012)

学位論文審査結果の要旨

文字列から 0 個以上の文字（必ずしも連続していなくともよい）を削除して得られる文字列を、元の文字列の部分列という。２つの文字列 A、B の最長共通部分列（Longest Common Subsequence、以降 LCS）とは、A と B の両方に共通の部分列の中で最も長い文字列をいう。2 つの文字列の LCS を求める計算は DNA 配列の比較などの様々な問題に応用できる。特に DNA 配列を表す文字列は 30 億文字などの非常に長い文字列となるために、単純なアルゴリズムを用いると、その LCS 計算には莫大な計算時間とメモリが必要となる。このため LCS を求める効率のよいアルゴリズムに関する研究が盛んに行われてきた。これらのうち使用メモリ量が現実的なアルゴリズムには次の２つがある。

A1. 動的計画法に基づく Hirschberg の LCS 計算アルゴリズム内の LCS の長さのみを計算する部分を Crochemore らのビット並列アルゴリズムを用いて高速化したアルゴリズム

A2. LCS 計算を４つの部分問題に分割して再帰的に解く Chowdhury らのアルゴリズム

これらのアルゴリズムは、いずれも単一コアの CPU を対象としたアルゴリズムである。

本論文ではアルゴリズム A1 を並列化して GPU（Graphics Processing Unit）を用いて高速化する手法を提案している。アルゴリズム A1 は並列実行が可能なビット毎の論理演算の他に、逐次性が強い算術加算など、GPU での実装に工夫が必要な演算も含んでいる。本論文では特にそれらの演算の GPU 上での効率的な実装方法について論じている。

提案手法は、スレッドブロック間の並列処理は動的計画法の表のブロック単位でのウェーブフロント処理により行い、スレッドブロック内の並列処理は条件求和加算などにより行うものである。さらに本論文では、提案手法に基いて実装した GPU プログラムおよびマルチコア CPU 向けに A1 を並列化したマルチスレッド CPU プログラムを用いた評価実験を行い、提案手法の有用性を実証的に検証し、GPU を用いる提案手法が CPU の 1 コア上で実行した A1 に対しては最大 12.81 倍、CPU の 4 コア上で並列実行した A1 に対しては最大 4.56 倍高速であることを示している。また、Kloetzli らが提案している、GPU を用いて並列化した A2 に対しては最大 18.1 倍高速であることを示している。

以上のように、本研究では２つの文字列の最長共通部分列を従来手法より十分に高速に求める手法の開発に成功している。この手法は多くの応用に適用できる。以上のことから、本委員会は本論文の審査、最終試験の結果に基づき、河南克也氏に博士（理学）の学位を授与することを適当と認める。