Vol.2017-HPC-160 No.21
2017/7/27

メタゲノム解析アプリケーションGHOSTZ-GPUの性能モデリングおよび改善

山川 智史[1]　野村 哲弘[1]　松岡 聡[1]

概要：メタゲノム解析とは微生物の分離を行わず、そのまま DNA を読み取り、配列相同性検索を行う手法である。これにより従来のゲノム解析では得られなかった情報を得られるようになる。反面、配列相同性検索では多くのミスマッチ、ギャップを許容して計算を行うので多大な計算時間が必要となる。GHOSTZ は秋山研究室で開発された高速なメタゲノム解析を行うアプリケーションである。GHOSTZ-GPU は GHOSTZ の処理の一部を GPU 化したものであり、元のプログラムから 5〜7 倍の性能向上を達成している。しかし CPU Thread 数と GPU 数を変えて実行時間の変化を見ると実行時間が CPU Thread 数のみに依存しており、GPU が有効活用されてないと疑わる処理が存在することがわかる。これでは CPU Thread 数,GPU 数の片方を増やしても計算速度の向上を図ることができない問題点がある。
本研究では GHOSTZ-GPU のスケーラビリティを向上させるべく、性能ボトルネックとなっている箇所を明らかにするため、性能モデリングを行った。立てたモデルの実行時間予測の性能は相対誤差 11%以下であった。また実験より CPU-GPU 間での負荷分散が性能を律速する要因であることも判明した。

キーワード：性能モデリング, メタゲノム解析

1. 背景

1.1 メタゲノム解析

ゲノム解析とは微生物を採取、分離、培養し DNA 配列を読み取り、それをデータベース配列との探索にかけるという処理である。この探索では配列同士の文字の違い (ミスマッチ) や読み飛ばし (ギャップ) を許容する曖昧検索を行う必要がある。

メタゲノム解析は採取して得られたサンプルから微生物の分離、培養を行わずに DNA を読み取り、それを配列相同性検索にかけるものである。この手法は従来のゲノム解析では得ることができなかった情報を入手可能になっており有用である。反面、この検索では多くのミスマッチやギャップを許すため、計算時間が非常にかかる。

1.2 GHOSTZ-GPU

GHOSTZ-GPU は東京工業大学秋山研究室で開発されたメタゲノム解析を高速に行うアプリケーションであり、過去に開発された GHOSTZ[3] の一部の処理を GPU 化したものである。GHOSTZ を 12 CPU Thread で実行した時間と比較して GHOSTZ-GPU を 12 CPU Thread, 3 GPU を用いて実行した時間は約 5.8〜7.7 倍の速度となっている。また、同等の感度を達成する条件下でほかのメタゲノム解析アプリケーション (RAPSearch, DIAMOND) と比較しても高速に解析することができる [1]。

1.3 問題点

GHOSTZ-GPU を CPU Thread, GPU 数を変えて実行したときの実行時間の変化を表したものが**図 1** である。この図より、GPU 化によって高速化ができていることがわかる。しかし、GPU 実行時でも CPU Thread に依存して性能が伸びてる箇所が存在する。この部分では CPU 律速になっており、GPU の性能を生かし切れていないと考えられる。これは GPU 数を増やしても性能が CPU に律速されていることを示しており、3 GPU の実行では 12 CPU Thread でも GPU の能力を生かし切れていないという問題点がある。

本研究ではこの問題の原因を明らかにするために GHOSTZ-GPU の性能解析をおこなった。GHOSTZ-GPU のプログラムを処理ごとに分割し、それぞれの処理のパフォーマンス特性を解析した。また解析した結果をもとに各処理のモデル式を立てた。成果として CPU 律速となる原因を特定した。また解析データから得られたモデル式で実行時間をモデル化した。モデル式による全体の実行時間

[1] 東京工業大学

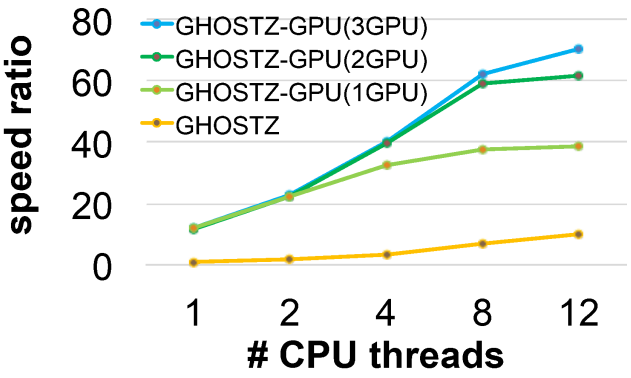

図 1　GHOSTZ-GPU の CPU Thread, GPU 数を変えたときの実行時間

Fig. 1　english

の予測の精度は相対誤差 11%以下であった。また CPU と GPU の間で負荷の不均衡を見出した。

2. モデリング

2.1 処理の分割, 計測

問題点として挙げた GHOSTZ-GPU の挙動を解明するためにプログラムの分割を行った。分割したルーチン毎に実行時間を GPU 数、CPU Thread 数を変えながら計測し、その結果から処理を実行時間一定処理、CPU 依存処理、GPU 処理と特徴づけた。

2.2 モデリングの方法

性能モデルとは入力として CPU 数、GPU 数、入力問題サイズ、メモリバンド幅などのパラメータを受け取り出力として実行時間などを出す数式である。実行時間のモデルを立てて理論的な計算量、メモリアクセス量と比較することで理論値との差異がわかり、最適化へのヒントになる。またモデル式から性能ボトルネックの特定も可能となる。実行時間予測を立てられるとスケジューリングがしやすくなるなどの利点もある。

今回作成するモデルは入力として CPU Thread 数、GPU 数を受け取り出力として実行時間を出すものとする。全体のモデル式をそのまま立てるのは困難であるため、分割、解析した処理それぞれにモデルを立てる。その後全体の実行時間のモデルを分割したモデルの和として定式化する。

2.3 配列相同性検索の流れ

配列相同性検索を行うとき、GHOSTZ-GPU は以下の流れで処理を行う。

(1) 入力クエリとデータベースに対する配列相同性検索の計算 (Presearch)
(2) (1) で得られた計算結果からクエリとデータベースとの紐づけ (BuildResult)
(3) 計算結果の出力 (Write)

この 3 つのステップは逐次的に実行されるため全体の処理時間はこれらの和となる。

2.4 Presearch

Presearch は以下の 3 つの処理で構成される。

(1) スレッドの生成、ループに入る前にデータベースチャンクを一つだけ読み込み (Preprocess)
(2) データベースの読み込み (Preload)
(3) アライメントの計算 (Alignment)

(2) と (3) は並列に実行されるため Presearch の全体の実行時間は (Preprocess にかかる時間)+max{Preload にかかる時間,Alignment にかかる時間 } となる。このように処理を分割していき、それぞれの処理の特徴を把握してモデルを立てていく。処理時間のモデルは以下の 3 つのモデルの組み合わせと和で表現する。

- 入力にかかわらず一定の時間がかかる処理

$$T_1 = C$$

- Thread 数のみに依存する処理

$$T_2 = \alpha / N_{Thread}$$

- Thread 数、GPU 数に依存する処理

$$T_3 = \max\{\alpha / N_{Thread}, \beta / N_{GPU}\}$$

例として BuildResult は CPU Thread 数に依存する処理、Write は入力にかかわらず一定の処理時間、Alignment はここからさらに分割し、CPU Thread 数に依存するモデルと CPU Thread, GPU 数の両方に依存するモデルの和としてかける。

3. 評価

TSUBAME2.5 上でモデル式を作るためのデータ、立てたモデル式の評価を行った。モデル作成のための分割したコードの実行時間の計測には NVIDIA Tools Extension[5] の nvtxRangeStart/Stop 関数と nvprof を用いた。処理の回数が多いものは nvprof のオーバーヘッドが大きくなるため gettimeofday を用いて計測した。計測により生成したモデルと実測値の比較は**図 2** の通りとなる。実測値と予測値との関係はスレッド数が少ないときは予測値は小さく、スレッド数が多いときは予測値は大きく出ている。相対誤差は**図 3** の通りとなる。これより、相対誤差 11%程度で予測がたてられていることがわかる。

図 4 に各条件下での CPU 側の GPU の処理待ちの同期にかかった時間を示す。図 2 と合わせてみると、CPU Thread 数が少ないときはほとんど待ち時間がなく CPU 側がボトルネックになっている。

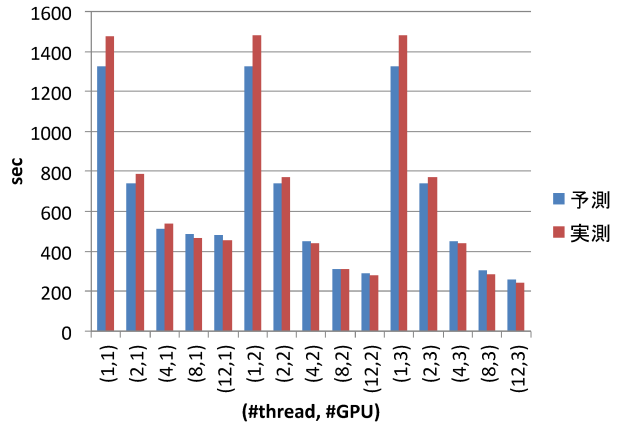

図 2　全体の実行時間の予測値と実測値の比較

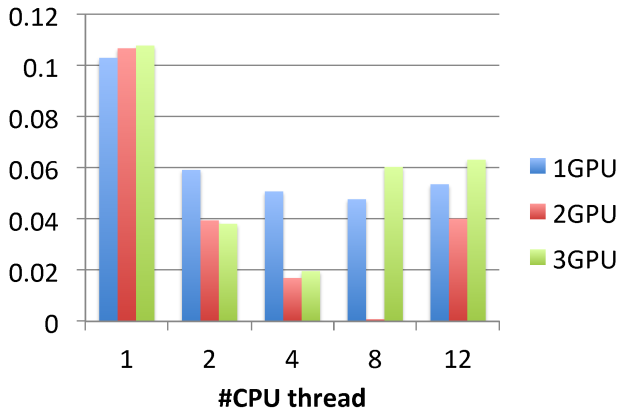

図 3　予測実行時間の相対誤差

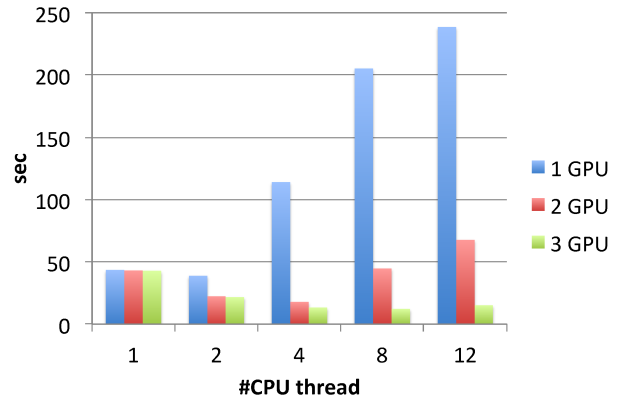

図 4　CPU における GPU 処理待ち時間の変化

4. 考察

4.1 モデル式による予測値と実測値の比較

今回立てたモデル式は Thread 数 1 以外では相対誤差 7%以下で予測できた。しかし Thread 数 1 の時に誤差が大きく出ている。実行時間は Thread 数 1 の時が最も大きいので Thread 数 1 で相対誤差が最も大きいということは絶対誤差も最も大きいということになる。この原因として CPU 依存な処理でモデルの立て方が甘いということが考えられる。今回立てたモデルは CPU 依存な処理の実行時間は Thread 数に反比例するというモデルを立てた。実際には Thread 数で並列化できる部分は限られており、そこの見積もりが甘いのではと考えられる。

5. 関連研究

5.1 メタゲノム解析アプリケーション

本研究では GHOSTZ-GPU[1] の性能解析を行った。メタゲノム解析を行うほかのアプリケーションとして GHOSTZ[3], GHOST-MP[2] がある。両者ともに東京工業大学秋山研究室で開発されたものである。GHOSTZ の特徴として、データベースのクラスタリングを行うことにより高速に配列相同性検索を行えるというものがある。この特徴は GHOSTZ-GPU にも生かされている。GHOST-MP は GHOSTX を分散メモリ環境上で実行できるようにしたものである、これにより並列数を大幅に上げることができ、高速な計算が可能である。一方 GHOSTZ-GPU はノード内並列には対応しているが複数ノードでの並列処理はできない。

5.2 性能モデリング

性能モデリングには以下の 2 つがある。

(1) 実験して得られた実行時間からモデルを作成
(2) メモリアクセス回数、浮動小数点演算回数などアプリの計算処理からモデルを生成する方法

1 の手法によるモデリリングは与えられた環境の元での実アプリの実行時間の推定に役立つ。一方 2 の手法は計算環境が変化しても対応が可能という利点がある [4]。今回対象としたアプリである GHOSTZ-GPU は入力であるクエリ、データベース配列によって演算回数が変わるので 1 の手法を用いてモデルを生成した。

6. まとめと今後の課題

本研究では GHOSTZ-GPU の性能モデリングを行った。得られたモデルの精度は相対誤差で 11%以下であった。モデリングにより CPU-GPU 間の仕事量の不均衡がわかった。

今後の課題として、モデルおよび計測手法の精緻化と、他の計算機でも性能予測を可能とする一般化、対象アプリケーション GHOSTZ-GPU における CPU 依存な処理の GPU 化、モデルから入力 GPU 数 CPU Thread 数に応じて CPU 側にも処理を割り振る手法の開発を行い性能向上をさせることがあげられる。

参考文献

[1] Suzuki S, Kakuta M, Ishida T, Akiyama Y (2016) GPU-Acceleration of Sequence Homology Searches with Database Subsequence Clustering. PLoS ONE 11(8):

e0157338. doi:10.1371/journal.pone.0157338
[2] Zhang Chaojie, Koichi Shirahata, Shuji Suzuki, Yutaka Akiyama, and Satoshi Matsuoka. "Performance Analysis of MapReduce Implementations for High Performance Homology Search" In Proceedings of the HPCS 2015 (2015)
[3] Shuji Suzuki, Masanori Kakuta, Takashi Ishida, Yutaka Akiyama. Faster sequence homology searches by clustering subsequences. Bioinformatics (2014) 31 (8): 1183-1190. doi : 10.1093/bioinformatics/btu780
[4] Hoefler, Torsten and Gropp, William and Kramer, William and Snir, Marc. Performance modeling for systematic performance tuning. SC '11 State of the Practice Reports (2011)
[5] CUDA Toolkit Documentation(online) 入手先 ⟨http://docs.nvidia.com/cuda/profiler-users-guide/index.html⟩ (2017.06.28).