Vol.2013-BIO-34 No.13
2013/6/28

GPGPUによるタンパク質タンデム質量分析の高速化

小幡 康文[1,a)]　石田 貴士[1,b)]　夏目 徹[2,c)]　秋山 泰[1,d)]

概要：タンパク質タンデム質量分析は生命科学や創薬などの分野で幅広く利用されている．近年は機器の高速化により，時間当たりで得られるスペクトルの量が増加し，更にタンパク質データベースサイズも増加している．そのためスペクトルの解析に高速な計算機が必要となっている．本研究では，質量分析プログラムである CoCoozo を対象に質量分析の高速化を図り，アルゴリズムの改良と，それに加えてマルチスレッド化と GPGPU 化の実装も行った．その結果，プレカーサ情報が有る場合の解析について，従来に比べて 8.9 倍の高速化を実現した．更に，プレカーサ情報が無い場合の解析について，12 コア CPU を用いた場合で従来に比べて 15.9 倍，さらに GPU を用いた場合で，従来に比べて 18.1 倍の高速化を実現した．

キーワード：質量分析，MS/MS，CoCoozo，マルチスレッド化，GPGPU

Acceleration of Tandem Mass Spectra Analysis Software CoCoozo using Multi-core CPUs and Graphics Processing Units

Yasufumi Obata[1,a)]　Takashi Ishida[1,b)]　Tohru Natsume[2,c)]　Yutaka Akiyama[1,d)]

Abstract: Tandem mass spectrometry, a method involving multiple steps of mass spectral selection, is widely used in various biological fields. In recent years, steady improvements have been made with respect to speed, and the number of protein databases available for analysis has rapidly increased. Consequently, computational analysis has become the bottleneck in tandem mass spectrometry. To overcome this problem, we attempted to improve the tandem mass spectrometry analysis software CoCoozo. To accelerate the program, we improved the algorithm and also incorporated utilization of multi-core CPU and GPGPU. As a result, when all mass spectral data files had precursor data, we achieved 8.9-fold speedups compared with the original software. In addition, in the case of no precursor data, by using a 12-core CPU and a GPU card we achieved 18.1-fold speedups compared with the original software.

Keywords: Mass Spectrometry, MS/MS, CoCoozo, Multi-threading, GPGPU

1. 導入

細胞に含まれる全てのタンパク質に加えて，ミトコンドリアや細胞膜などの細胞内小器官，タンパク質機能複合体など，大規模なタンパク質の集合体を網羅的に解析する研究をプロテオミクスと呼ぶ．このプロテオミクスで，サンプル中に含まれるタンパク質を同定する方法として，現在，質量分析が主に用いられる [1]．この質量分析を用いた研究の例として，癌やアルツハイマー病などの病気に特異的なタンパク質の解析・研究 [2], [3] や，タンパク質間相互作用の研究などが挙げられる [4].

現在主流の質量分析は，大まかに 3 つの段階に分けることができる．最初に分析対象となるタンパク質は，いくつかのペプチド鎖へと断片化され，イオン化される．次に，

[1] 東京工業大学 大学院情報理工学研究科 計算工学専攻, Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[2] 産業技術総合研究所 創薬分子プロファイリング研究センター, Molecular Profiling Research Center for Drug Discovery, National Institute of Advanced Industrial Science and Technology
[a)] obata@bi.cs.titech.ac.jp
[b)] t.ishida@bi.cs.titech.ac.jp
[c)] t-natsume@aist.go.jp
[d)] akiyama@cs.titech.ac.jp

そのペプチド鎖の質量電荷比と強度を計測し，そのスペクトルを出力する．このスペクトルを質量分析スペクトルと呼ぶ．そして，質量分析ソフトウェアを用いて，そのスペクトルを解析し，元のタンパク質の同定を行う．現在まで，様々な質量分析の方法が開発されてきたが，近年，広く用いられる方式に，タンデムマス法，もしくはタンデム質量分析（Tandem Mass Spectrometry, MS/MS, MS^2）と呼ばれる方式がある．タンデムマス法は，この質量分析の計測を 2 回，もしくはそれ以上の回数計測を行う手法である．タンデムマス法の利点として，分析対象が混合物質であっても分析可能であるという点が挙げられる．タンデムマス法の中でも，2 段階のものが現在良く用いられている．2 段階のタンデムマス法において，1 回目の計測時に分析対象のタンパク質から断片化・イオン化されたペプチド鎖をプレカーサ（precursor）と呼ぶ．1 回目の計測後，プレカーサは計測結果を基に選別される．そして，2 回目の計測時に，各プレカーサは更に断片化・イオン化され，それぞれの質量と強度が計測される．このプレカーサから断片化されたペプチド鎖をフラグメント（fragment）と呼ぶ．タンデムマス法で得られるスペクトルデータには，フラグメントの質量電荷比（mass-to-charge ratio, m/z）と，それに対応する強度（存在量，inteisity），それに加えて，多くの場合，そのフラグメントの元となったプレカーサの情報も記録されている．

タンデムマス法によって得られる情報は質量分析スペクトルであるため，これを分析し，タンパク質を同定するには何らかのソフトウェアが必要となる．これまで様々なソフトウェアが開発されており，その中でも MASCOT[5] と呼ばれるソフトウェアが現在広く用いられているが，その他にも，SEQUEST[6] や SpectraST[7]，CoCoozo といったソフトウェアがある．これらのソフトウェアは，スペクトルデータとデータベースとの比較や類似度の計算に異なったアルゴリズムを使用しており，それぞれのソフトウェアに特徴がある．たとえば，MASCOT は，統計計算を利用した評価を行い分析を行っているが [5]，SEQUEST では，相互相関スコア（cross-correlation score）を用いて評価を行っている [6]．また，これらのソフトウェアが，タンパク質の一般的なデータベースを利用して検索を行うのに対して，SpectraST では，質量分析スペクトルのデータベースに対して検索を行うという特徴がある [7]．CoCoozo は，産業技術総合研究所及び東京工業大学秋山研究室で 2000 年代中頃から開発されてきた質量分析スペクトル解析システムである．特徴として，適合率（precision）の高い解析を行うことが可能である点や，独自の誤差補正機能を搭載しているという点が挙げられる．CoCoozo は，過去数年間，産業技術総合研究所で実際に利用されている．

近年は，質量分析に関する測定技術の向上の結果，検出されるデータの量も増加し，データの質も上昇している．それに伴い，コンピュータによるデータベースとのマッチング自体が，質量分析においてボトルネックとなりつつある．

その一方で，近年では計算機ノード自体の処理性能も飛躍的に向上している．特に性能向上の面では，CPU のマルチコア化に伴うマルチスレッドによる並列化や，GPU（Graphics Processing Units）を用いた汎用演算，GPGPU（General-purpose computing on graphics processing units）など，様々な高速化技術の開発が大きく貢献している．GPU とは，3D グラフィックスなどの画像処理の計算に特化した集積回路のことである．GPU の処理性能の上昇は目覚ましいものがあり，GPU の画像処理以外の目的での使用が試みられ，汎用演算に利用する技術が開発された．この，GPU を画像処理以外の汎用計算で利用する技術を GPGPU と呼ぶ．この結果，GPU は様々な研究に用いられるようになった．その一例として，天体計算 [8] や高速フーリエ変換 [9] が挙げられる．

本研究では，質量分析スペクトル解析ソフトウェア CoCoozo を対象に，アルゴリズムの改良を行い，ノード内並列化，および GPGPU 化による高速化を図った．CoCoozo は，すでに MPI 化されており，複数ノードを同時に用いて並列に動作することで高速化を図っているが，本研究においては，1 ノードにおける高速化を目指した．

2. CoCoozo

CoCoozo は，産業技術総合研究所及び東京工業大学秋山研究室で 2000 年代中ごろから開発された質量分析スペクトル解析システムである．そのメインプロセスのフローチャートを図 1 に示す．CoCoozo は，このフローチャートの主な処理をデータベース側の各プレカーサに対して行っている．まず，最初にクエリーとなるスペクトルファイルにプレカーサの測定結果の情報があるかどうかを判定する．もし，スペクトルファイルにプレカーサの情報がある場合は，次に “プレカーサ・マッチング” と呼ばれる処理を行う．プレカーサ・マッチングでは，クエリーのプレカーサデータとデータベース側のプレカーサデータの比較が行われる．もし，マッチしていると判断されれば，それに続いて “フラグメント・マッチング” と呼ばれる処理が行われる．これに加えて，クエリーのスペクトルファイルにプレカーサの測定結果の情報が無い場合にも，プレカーサ・マッチングを行うことなく，すぐにフラグメント・マッチングが行われる．フラグメント・マッチングでは，クエリーとなるスペクトルファイルの各スペクトルデータとデータベース側のフラグメントのスペクトルデータの類似度の計算が行われる．ただし，この時，比較されるデータベースのフラグメントは，最初に選択されたデータベースのプレカーサから生成される可能性のあるフラグメントのみである．その後，フラグメント・マッチングの結果を用

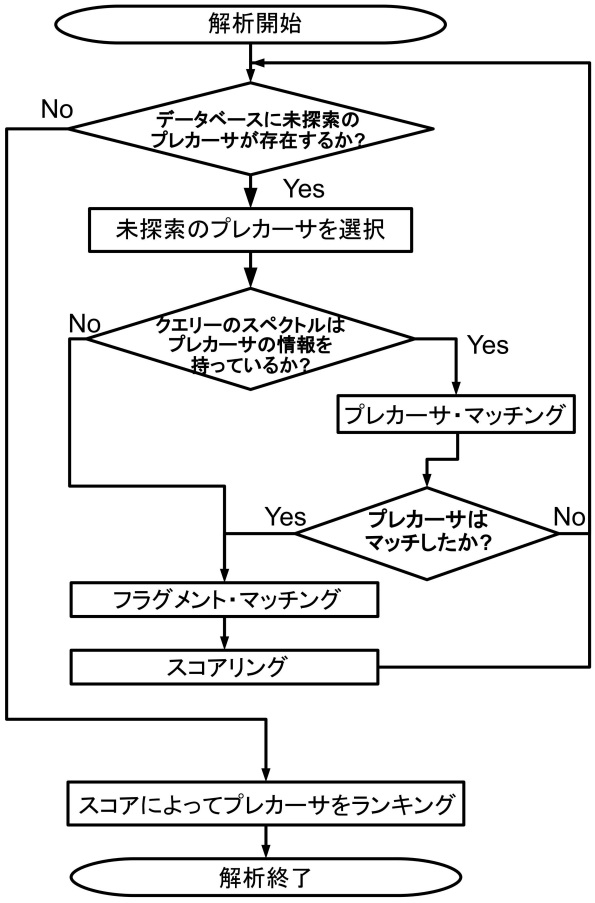

図 1 CoCoozo のメインプロセスのフローチャート（1 スペクトルデータファイル）

Fig. 1 CoCoozo Main Process Flowchart (for a mass spectral data)

いて，データベースのプレカーサに対して，スコア付けが行われる．データベース側の全てのプレカーサに対して，このマッチングとスコアリングを行った後，スコアの値を基にプレカーサをランク付けし，高いスコアであったプレカーサを解析結果として出力する．このメインプロセスの中で主要な処理となるプレカーサ・マッチングとフラグメント・マッチングは類似の処理である．これらの処理は，共にデータベース中のプレカーサ，またはフラグメントの質量が，クエリーのデータとマッチするかを判定する．この際，クエリーとなる質量分析スペクトルに，誤差が含まれている可能性を考慮し，データベース側の質量の両側にトレランスと呼ばれる区間を設け，その区間に入るかどうかを判断している．このトレランスの幅は，質量とトレランスの比率の積によって決まる．プレカーサ・マッチングとフラグメント・マッチングの違いの一つとして，このトレランス幅を決定する比率の違いが挙げられる．プレカーサ・マッチングでは，この比率は固定値が用いられるが，フラグメント・マッチングの場合は，比較対象のスペクトルの強度に依存した可変値が用いられる．加えて，データベース中のフラグメント 1 つに対して，マッチと判定されるのは，トレランスに含まれるスペクトルの内，最も強い強度のスペクトル 1 つのみである．このため，プレカーサ・マッチングはフラグメント・マッチングよりも高速に動作する．

2.1 CoCoozo のボトルネック

まず，CoCoozo のボトルネックとなっている部分を発見するため，2 つの場合について各関数の消費時間を計測した．1 つ目の場合は，全てのスペクトルファイルにプレカーサの測定結果が含まれている場合（プレカーサ情報有りの場合），もう一つの場合が，全スペクトルファイル中の約 10% のスペクトルファイルがプレカーサの測定結果を欠いている場合である（プレカーサ情報無しの場合）．

この計測結果より，CoCoozo において，主に時間を消費する部分が，プレカーサとフラグメントのマッチングの部分であるということが分かった．加えて，プレカーサ情報無しの場合，プレカーサ情報有りの場合と比べて，解析に約 13 倍の時間がかかっていることも分かった．プレカーサ情報有りの場合，プレカーサ・マッチングが全実行時間の内，約 71 %を 占める演算となっている．一方，プレカーサ情報無しの場合，全実行時間の内，最も時間を消費しているのはフラグメント・マッチングで，全実行時間の内，約 85 % を占める計算となっている．このフラグメント・マッチングに要している時間は，全てプレカーサ情報を持たないスペクトルデータに対しての解析のために使用されており，プレカーサ情報が無いファイルの解析は，プレカーサ情報が有るファイルの解析に比べて，非常に長い計算時間を必要としていることが分かった．

3. 提案手法

3.1 マッチングアルゴリズムとスコア初期化のアルゴリズム改良

律速となっているプレカーサ・マッチングとフラグメント・マッチングのアルゴリズムに改良を施す．具体的には，比較されるデータを質量の順にソートし，それを用いて比較を行うように改良を行った．しかし，タンパク質は，翻訳後修飾によって，質量が変化している場合が多々ある．CoCoozo も，いくつかの翻訳後修飾に対応しており，設定によって対応する修飾を変更することができる．その翻訳後修飾に対応するための質量補正によって，ソートが困難なプログラム設計となっていた．そこで，翻訳後修飾に対応したまま，質量に応じてデータをソートすることができるように，プログラムのデータ形式から変更を行い，ソートを実現した．ソートしたデータを用いてマッチングを行うことで，トレランスから外れた場合，すぐにマッチング処理を終了することが可能となる．フラグメント・マッチングでは，トレランスの幅が可変であるため，最大幅のトレランスで終了判定を行うように実装した．これによって，

マッチングにおける比較処理の回数が大幅に減少する．このマッチングの早期終了に加えて，プレカーサ・マッチングにおいては，トレランスの下限よりも小さい質量のプレカーサとの比較にはスキップを導入し，比較回数を更に減少させた．

このマッチングアルゴリズムの改良に加えて，スコアの初期化の処理についても効率化を図った．スコアは，各プレカーサに 1 つずつ割当てられている．そのため，解析が行われる前に，全てのスコアを初期の状態にする必要がある．オリジナルのスコア初期化のアルゴリズムでは，スコアが初期の状態から変更がされたかどうかにかかわらず，全てのスコアを初期化するようになっていた．これは，解析開始時に全てのスコアが初期化されていることを保証するアルゴリズムである．しかしその反面，プレカーサ情報が有る場合は，スコアの値が書き込まれるものはそれほど多くないため，全てのスコアを初期化するのは，非常に冗長な処理となっていた．そこで，変更されたスコアのみを初期化するように変更し，スコア初期化の効率化を図る．

3.2 マルチスレッディング

プレカーサ情報の無いスペクトルデータファイルの解析の際のフラグメント・マッチングとスコアリングの部分をマルチスレッド化し，高速化することを目指した．フラグメント・マッチングとスコアリングの処理は連続した処理で，一連の処理で同じプレカーサから生成されるフラグメントについてのみ処理を行う．そのため，これらの処理は，ほかのプレカーサから生成されるフラグメントに対するフラグメント・マッチングとスコアリングからは独立した処理である．そこで，各スレッドが，それぞれ別のプレカーサから生成されるフラグメントについてマッチング処理とスコアリングを行うように改良する．

今回のマルチスレッド化では，IEEE によって標準化された規格である POSIX.1 [10] に沿ったライブラリである，POSIX thread（pthread）を使用した．

3.3 GPGPU による高速化

マッチング・アルゴリズムの改良後も，プレカーサ情報を持たないスペクトルデータファイルに対する解析は，処理時間の約 70 %を 占める処理であった．そのため，この処理を GPGPU を用いて高速化することを試みた．フラグメント・マッチングにおいて，データベースのあるフラグメントのデータと，クエリーとなるスペクトルデータファイルのフラグメントの 1 スペクトルとの比較処理は，他の比較処理とは独立している．その一方，それぞれの比較処理自体は計算時間のかかるものではない．しかし，この後に続く処理は，他のデータベースのフラグメントとフラグメントのスペクトルとの比較の結果に依存した処理となる．そのため，この部分は，GPU 上で効率的に実行することは難しいと考えられる．

このため，今回 GPGPU を導入するのは，フラグメント・マッチングにおけるデータベース側のフラグメントとフラグメントのスペクトルとの質量の比較，およびフラグメントのスペクトルの強度の比較の部分のみである．しかし，フラグメント・マッチングの可変トレランスは，複数の判断文を含んでおり，GPU で実行すると実行効率を下げてしまうことや，GPU 側に可変トレランスを導入しても，CPU 側の処理が余り減少しないことなどから，GPU では最大幅のトレランスによるマッチングを行う．つまり，GPU 側で，粗いマッチングを行い，その結果を用いて CPU 側で正確なマッチングを少数回行うことで，高速化を目指す．

更に，GPGPU 化に加えて，GPU での処理の後に行われる CPU での処理について，マルチスレッド化を施した．マルチスレッド化した部分は，フラグメント・マッチングの GPGPU を導入していない部分とスコアリングの部分であり，前節で述べたマルチスレッド化の部分と同じ部分である．

GPGPU を実装するにあたり，今回は，NVIDIA 社の提供する GPGPU 開発環境，CUDA（Compute Unified Device Architecture）を使用した．CUDA の version は，2.3 以上に対応するように実装を行った．

4. 結果と考察

4.1 データセットとデータベース

今回使用したデータセットとデータベースは，小規模なデータセットとデータベース（データセット 1・データベース 1）と大規模なデータセットとデータベース（データセット 2・データベース 2）の 2 つである．以下にそれぞれの詳細を述べる．

表 1 データベース
Table 1 Databases used in the evaluation

	データベース 1	データベース 2
タンパク質エントリ	38,415	139,536
プレカーサエントリ	857,298	3,021,877
フラグメントエントリ	26,489,468	132,599,864
タンパク質断片化酵素	リシルエンドペプチダーゼ	

4.1.1 データセット 1・データベース 1

小規模なデータセット 1 の解析対象となるスペクトルデータファイルの総数は，1,486 ファイルである．これらのファイルは，PKL 形式のファイルで事前にフィルタリングが行われている．また，全てのスペクトルデータファイルはプレカーサの計測結果の情報を保持している．そのため，このデータセット 1 に加えて，データセット 1 を複製し，そこからランダムに約 10% に当たる 149 ファイルを選択，人工的にプレカーサ情報を削除したデータセット

1（プレカーサ情報無し）を作成した．

データベース 1 は，小規模なデータベースで，その内容を表 1 に示した．

4.1.2 データセット 2・データベース 2

大規模なデータセット 2 では，解析対象となるスペクトルデータファイルの総数は，10,002 ファイルである．これらのファイルもデータセット 1 と同じく，PKL 形式のファイルで事前にフィルタリングが行われている．また，全てのスペクトルデータファイルはプレカーサの計測結果の情報を保持している．今回は，このスペクトルデータファイルを複製し，その全てのファイルからプレカーサ情報を削除したデータセット 2（プレカーサ情報無し）を作成した．

データベース 2 は，大規模なデータベースで，その内容を表 1 に示した．

4.2 実験環境

実験は，東京工業大学のスーパーコンピュータ TSUBAME2.0 において行った．プログラムは，12CPU コアを搭載した TSUBAME2.0 の Thin ノードで実行した．詳しい実験環境について表 2 に示す．

表 2 実験環境
Table 2 Computing Environment

CPU	Intel Xeon 2.93 [GHz]（6 cores）x 2
Memory	54 [GB]
OS	SUSE Linux Enterprise Server 11 SP1
GPU	NVIDIA Tesla M2050
Compiler	gcc 4.3.4
MPI	OpenMPI 1.4.2
CUDA	CUDA 4.1（64bit）
Profiler	Intel VTune Amplifier XE 2011

時間の計測には UNIX の “`time`” コマンドを使用し，より詳しい解析を行うプロファイリングには Intel VTune Amplifier XE 2011 を使用した．

また，今回の実験においては，一価および一部の二価フラグメントについて検索を行い，N 末端のアセチル化を考慮した解析を行う設定で実験を行う．

4.3 データセット 1・データベース 1 に対する解析の結果

本節では，データセット 1・データベース 1 を用いた場合の実行結果について述べる．

4.3.1 マッチングアルゴリズムとスコア初期化のアルゴリズム改良の結果

図 2 は，全てのスペクトルデータファイルにプレカーサの情報が含まれている場合（プレカーサ情報有りの場合）の実行時間の結果である．アルゴリズム改良によって，オリジナルの CoCoozo と比べて全体で約 8.9 倍の高速化を達成した．特に，律速となっていたプレカーサ・マッチング

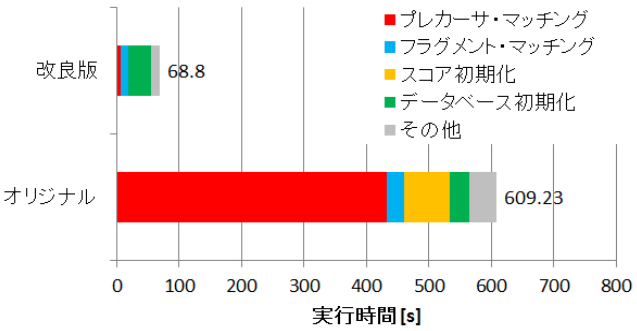

図 2 “マッチングアルゴリズムとスコア初期化のアルゴリズム改良”の結果（実行時間）（プレカーサ情報有りの場合）
Fig. 2 Result of “Improvement of Matching Algorithm and Initialization Process” (execution time) (in the case with precursor).

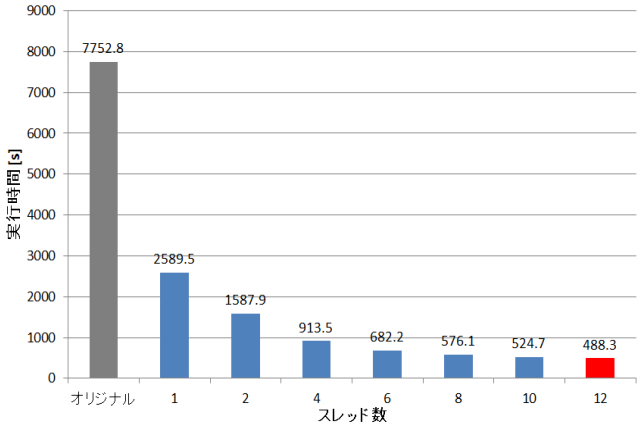

図 3 マルチスレッド化の結果（実行時間）（プレカーサ情報無しの場合）
Fig. 3 Result of “Multithreading” (execution time) (in the case without precursor).

では，オリジナルの CoCoozo と比べて約 65.3 倍の高速化を達成し，フラグメント・マッチングでは 2.5 倍の高速化を達成した．また，スコア初期化については，改良によって，オリジナルから約 483.1 倍の高速化を達成している．これによって，スコア初期化は，その他の短い処理と同程度の時間で処理可能となった．

これより，アルゴリズムの改良は，プレカーサ情報を持つスペクトルデータファイルに対する解析について，非常に大きな効果を発揮したことが示された．

4.3.2 マルチスレッディングの結果

図 3 は，約 10% のスペクトルデータファイルにプレカーサ情報が無い場合（プレカーサ情報無しの場合）のマルチスレッド化についての実行時間のグラフである．図 3 の棒グラフの番号は，マルチスレッド化した部分のスレッド数である．また，マルチスレッド化版の CoCoozo は，アルゴリズムの改良を適用しているものである．そのため，スレッド数が 1 の場合は，アルゴリズムの改良のみを施した場合の実行時間である．このグラフより，プレカーサ情報

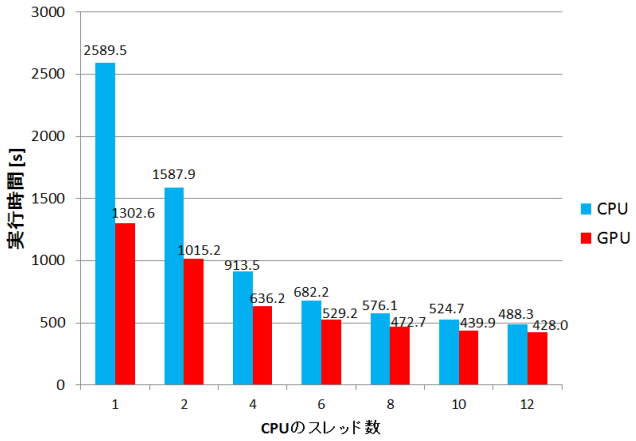

図 4 “GPGPU” 導入による高速化の結果（実行時間）（プレカーサ情報無しの場合）

Fig. 4 Result of “Acceleration by GPGPU” (execution time) (in the case without precursor).

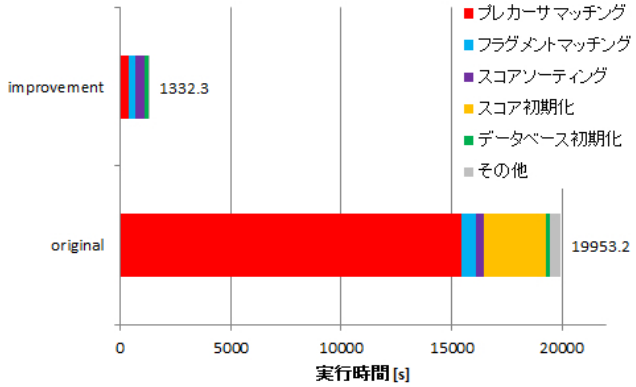

図 5 データセット 2 に対する解析の結果（実行時間）（プレカーサ情報有りの場合）

Fig. 5 Result of Analysis for Dataset2 (execution time) (in the case with precursor).

無しの場合でもアルゴリズムの改良だけで，かなりの高速化が実現できていることが分かる．また，12 スレッドの場合，1 スレッドの場合から約 5.3 倍の高速化を達成した．12 スレッドを使用しているにも関わらず，速度の向上が 5.3 倍と 12 倍を大きく下回った理由は，マルチスレッド化を実装した部分が，プログラム全体のごく一部にとどまっていることが挙げられる．また，もう一つの理由としてスレッドの並列実行度が挙げられる．Intel Vtune Amplifier を用いて，並列度の解析を行うと，12CPU コアを持つ環境で 12 スレッドを使用する設定をしても，並列度のピークは 6 スレッドであった．つまり，12 スレッドが同時に実行されることはほとんどなく，いくつかのスレッドがアイドル状態となっている．そのため，並列化の効果が低くなったと考えられる．

4.3.3 GPGPU による高速化の結果

図 4 は，約 10% のスペクトルデータファイルにプレカーサ情報が無い場合（プレカーサ情報無しの場合）のマルチスレッド化版（“CPU”）と GPGPU 化およびマルチスレッド化版（“GPU”）の実行時間のグラフである．このグラフの “CPU” は，図 3 のグラフと同じものである．CoCoozo にアルゴリズム改良と GPGPU を導入した場合は，アルゴリズム改良のみの場合から，約 2.0 倍の高速化を実現している．特に，GPGPU を一部の処理に導入したフラグメント・マッチングの部分のみを比較すると，GPGPU 導入前のアルゴリズム改良版より約 13.8 倍の高速化を実現している．アルゴリズムの改良と GPGPU に加えて，CPU のマルチスレッド化を実施した場合，CPU 側を 12 スレッドで動作させた場合は，アルゴリズムの改良と GPGPU のみを行った場合と比較して，約 3.0 倍の高速化を実現している．12 スレッドを使用しても，速度向上が 3 倍程度にとどまった理由として，先程と同じく，マルチスレッド化を行った部分が，一部分の処理に留まっていること，そしてスレッド並列度が挙げられる．Intel Vtune Amplifier を用いてスレッド並列度を計測すると，今回は 3 スレッドがピークとなっており，9 スレッド以上が同時に実行されることはほとんどないという状況になっている．この原因としては，GPGPU の使用で，CPU の各スレッドの担当するフラグメント・マッチングの計算量が減少したためであると考えられる．そのため，より多くのスレッドがアイドル状態となってしまい，高速化倍率が伸びなかったと考えられる．最終的に，CoCoozo にアルゴリズム改良・マルチスレッド化・GPGPU を導入することで，オリジナルの場合から約 18.1 倍の高速化を実現した．

4.4 データセット 2・データベース 2 に対する解析の結果

本節では，データセット 2・データベース 2 を用いた場合の実行結果について述べる．

4.4.1 全てのデータファイルにプレカーサ情報が有る場合

図 5 は，データセット 2（プレカーサ情報有りの場合）に対する解析実行時間の結果である．この計測結果より，アルゴリズム改良によって，オリジナルの CoCoozo から全体で約 15.0 倍高速化している．このことから，データセットおよびデータサイズの大きいデータを取り扱う場合，オリジナルの CoCoozo に対して，より効果的な高速化が達成されているといえる．

4.4.2 全てのデータファイルにプレカーサ情報が無い場合

データセット 2（プレカーサ情報無しの場合）に対する解析は，データセットおよびデータサイズの問題で，オリジナル版では長大な実行時間が必要となり，全てのファイルの解析を行うことができなかった．そのため，オリジナル版では 10002 ファイル中，500 ファイルをランダムに選び，その実行時間を利用して，全ファイルを処理した場合の解析時間を推計した．一方，GPGPU を導入し，更に

CPU12 スレッドに改良した CoCoozo では，実行時間が大幅に減少したことで，全てのファイルを通して実行することが可能で，実行時間について計測を行うことができた．実行時間は表 3 のようになる．

表 3 データセット 2 に対する解析の結果（実行時間）（プレカーサ情報無しの場合）

Table 3 Result of Analysis for Dataset2 (execution time) (in the case without precursor).

	実行時間
GPGPU & 12 スレッド	約 30 時間
オリジナル	約 515 時間（推計）

これより，速度向上倍率は約 17.2 倍となる．この速度向上比は，規模の小さいデータベース 1・データセット 1 の解析の場合の速度の向上よりも低いものではあるが，ほぼ同等の速度向上を達成していると考えられる．

4.5 考察

改良後の解析結果については，改良前とほぼ同じ解析結果を出力するように注意したものの，わずかながら差異が発生した．この差異は，スコアの値が多少変わってしまうというもので，オリジナルの解析結果から大きな変化はないものであった．この僅かな差は，マッチングの順番がソーティングによって変わってしまったために起きるものであり，同一強度のフラグメントのスペクトルが，フラグメント・マッチング時に同じデータベースのフラグメントのトレランス内に入ってしまった場合にのみ発生するものである．

5. 結論

本研究では，質量分析スペクトル解析システムである CoCoozo の改良を行い，高速化を達成した．その際まず計算時間の内訳の分析を行い，プレカーサ及びフラグメントのマッチングが律速になっているという計測結果を得た．そこで，本研究では，主にプレカーサ及びフラグメントのマッチング双方のアルゴリズムの変更を行った上で，マルチスレッド化も行い高速化を実現した．ピークファイルに親プレカーサの情報が有る通常の場合において，小規模なデータベース・データセットにおいて約 8.85 倍の高速化に成功した．更に，大規模なデータベース・データセットを使用した場合は，約 15.0 倍の高速化を実現した．一方，クエリであるピークファイルの内，約 10%から親プレカーサの情報を削除した場合において，アルゴリズムの変更によって約 3 倍の高速化を成功し，それから更にマルチスレッド化によって，12 スレッドの場合，1 スレッドの場合から 5.3 倍の高速化に成功した．オリジナルの CoCoozo と比べると，約 16 倍の高速化に成功している．更に，GPU を用いた計算が行えるようにアルゴリズムの変更を行い，改良前の CoCoozo と比べ，約 6 倍の高速化に成功した．更に，GPU 化に加えてマルチスレッド化を適用することにより，CPU12 スレッドで動作させた場合，1 スレッドで動作させた場合から約 3 倍の高速化に成功し，オリジナルの CoCoozo から約 18.1 倍の高速化に成功した．また，これらの高速化による解析結果の変化は少なく，従来までの CoCoozo とほぼ同等の結果をより速く得ることが可能となった．

本研究で実装したこれらの高速化は，解析結果をほとんど変えることなく高速化することに成功しており，より膨大な質量分析スペクトルの解析を行うことを可能とした．この成果は，CoCoozo を使用する上で有用であり，質量分析スペクトル解析をより円滑に行うことを可能とした．

謝辞
本論文を終えるに当たり，質量分析に関して様々な助言を賜りました，CoCoozo の開発者である，産業技術総合研究所 小池 克幸氏，草野 秀男氏，八田 知久氏にお礼申し上げます．また，CoCoozo の開発当時の主たるプログラマとして，様々な疑問に答えていただきました，理化学研究所神戸研究所の藤原 康広氏にお礼申し上げます．

参考文献

[1] W.P.Blackstock and M.P.Weir: Proteomics: quantitative and physical mapping of cellular proteins, *Trends Biotechnol.*, Vol. 17, No. 3, pp. 121–127 (1999).
[2] E.P.Diamandis: Mass spectrometry as a diagnostic and a cancer biomarker discovery tool: opportunities and potential limitations, *Mol Cell Proteomics.*, Vol. 3, No. 4, pp. 367–378 (2004).
[3] D.C.German, P.Gurnani, A.Nandi, H.R.Garner, W.Fisher, R.Diaz-Arrastia, P.O'Suilleabhain and K.P.Rosenblatt: Serum biomarkers for Alzheimer's disease: proteomic discovery, *Biomed Pharmacother.*, Vol. 61, No. 7, pp. 383–389 (2007).
[4] A.C.Gavin, K.Maeda and S.Kühner: Recent advances in charting protein-protein interaction: mass spectrometry-based approaches, *Curr Opin Biotechnol.*, Vol. 22, No. 1, pp. 42–49 (2011).
[5] D.N.Perkins, D.J.Pappin, D.M.Creasy and J.S.Cottrell: Probability-based protein identification by searching sequence databases using mass spectrometry data, *Electrophoresis.*, Vol. 20, No. 18, pp. 3551–3567 (1999).
[6] J.K.Eng, A.L.McCormack and J.R.Yates, I.: An Approach to Correlate Tandem Mass Spectral Data of Peptides with Amono Acid Sequences in a Protein Database, *J. Am. Soc. Mass Spectrom.*, Vol. 5, No. 11, pp. 976–989 (1994).
[7] H.Lam, E.W.Deutsch, J.S.Eddes, J.K.Eng, N.King, S.E.Stein and R.Aebersold: Development and validation of a spectral library searching method for peptide identification from MS/MS, *Proteomics*, Vol. 7, No. 5, pp. 655–667 (2007).
[8] Nguyen, H.: *GPU Gems 3*, Addison-Wesley Professional, Boston (2007).
[9] N.K.Govindaraju, B.Lloyed, Y.Dotsenko, B.Smith and J.Manferdelli: High Performance Discrete Fourier Transforms on Graphics Processors, *the 2008 ACM/IEEE conference on supercomputing*, pp. 1–12 (2008).

[10] IEEE: Information Technology - Portable Operating System Interface (POSIX) - Part1 : System Application Program Interface (API) [C Language] (1996).