jh200005-NAH

大規模並列計算による格子の最短ベクトル探索の効率化に関する研究

研究代表者 柏原 賢二（東京大学）

概要

最短ベクトル問題 (SVP) の大規模並列化による効率的なアルゴリズムを構築する。SVP は次世代の公開鍵暗号システムである格子暗号の安全性の元になる問題である。プロセスごとに基底簡約し、その情報を共有ファイルシステムを通じて協調計算することにより並列化を行う。

1　共同研究に関する情報

1.1　共同研究を実施した拠点名

東京大学

1.2　共同研究分野

■超大規模数値計算系応用分野

1.3　参加研究者の役割分担

柏原賢二 東京大学 研究のアイディアとコーディング

松田源立 成蹊大学 理論的考察

池上努 産業技術総合研究所 コーディングのサポート

照屋唯紀 産業技術総合研究所 コーディングのサポート

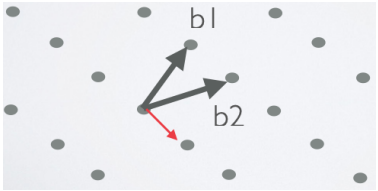
2　研究の目的と意義

最短ベクトル問題 (SVP) の大規模並列化による効率的なアルゴリズムを構築することが研究の目標である。SVP は次世代の公開鍵暗号システムである格子暗号の安全性の元になる問題である。大規模計算機システムを用い、並列プロセスごとに sieving アルゴリズムにより基底簡約し、その情報を共有ファイルシステムを通じて協調計算することにより並列化を行う。格子の最短ベクトル問題とは、与えられた基底行列によって定まる格子点のうち、原点をのぞいて最も短いベクトルを求める問題である。格子の最短ベクトルを求める効率的なアルゴリズムの研究は、次世代の暗号方式である格子暗号の安全性の見積もりに必須である。格子暗号の解読の困難さは、格子の最短ベクトル問題の困難さに基づいている。格子暗号を含む公開鍵暗号システムの安全性は、鍵の長さをどのくらいにとれば、現実的な時間内において、秘密鍵を見つけるのが不可能となるかということで議論できる。しかし、現在では、計算機を大量に借りて並列計算を行うことにより、従来はとても時間が長くかかって解くことが不可能であると思われた計算も、短い時間で解くことができるようになった。格子の最短ベクトルの効率的な並列アルゴリズムについての研究は、非常に大規模な並列計算をする必要がある。われわれの格子の最短問題へのアプローチは、行列の基

底をより簡単なものへと変換していくという格子基底簡約を用いたアプローチである。一時期は、最も効率的なアルゴリズムであったが、近年、Ducas らのグループにより、基底簡約に sieving を組み合わせたアルゴリズムでより効率的なものが開発された。この研究も、sieving と基底変換を組み合わせたアルゴリズムを利用している。

・SVP について

n 次元ユークリッド空間において n 個の線形独立な整数ベクトルの基底の整数結合で作られる点の全体が格子である。格子の各点の位置ベクトルを格子ベクトルと呼ぶことにする。正則な n 次元整数行列 (基底行列と呼ぶ) が与えられると、その行ベクトル (基底ベクトルと呼ぶ) で張られる格子を考えることができる。このような格子が与えられた時に、原点以外で原点に最も近い点を見つける問題が最短ベクトル問題 (Shortest Vector Problem, SVP) である。統計的な議論により、基底行列の行列式から最短ベクトルの長さを推測することができる。最短推測値から、与えられた定数倍の長さ以下のベクトルを求める問題を approximated-SVP(aSVP) と呼ぶ。aSVP の求解の計算困難性は、格子暗号システムなどの公開鍵暗号システムの安全性の根拠になっている。



SVP Challenge は、aSVP 問題のチャレンジサイトとして、2010 年からドイツのダルムシュタット工科大学によって運営されているウェブサイトである。格子の次元ごとの SVP の問題として、基底行列ファイルが公開されている。最短推測値から 1.05 倍以内の長さで、今までエントリーされた中で一番短いベクトルを見つけた時にサイトに登録できる。長さの条件を満たす格子ベクトルをゴールベクトルと呼ぶことにする。一般的に次元が高いほうがゴールベクトルを見つけるのが難しくなる。我々のグループも 150 次元や 152 次元などにおいて、SVP Challenge にエントリーすることができた。当時の最高次元のエントリーであった。現在は、Ducas らのグループにより、GPU も用いた計算によって 180 次元までの解がみつかっている。

・基底変換による SVP のアプローチ

基底行列に行列式が 1 か -1 であるような整数行列をかけても、同じ格子を生成する基底行列が得られる。これを基底変換と呼ぶことにする。基底変換を (何回か) 行うことにより、なんらかの意味でより簡単な基底に変換していくことを基底簡約と呼ぶ。その際に「簡単な」に対して、どのような基準を考えるかで、基底簡約の定義は変わってくるが、大まかには、直交基底ベクトルの長さが短くなる方向に変換することが基底簡約することになる。直交基底ベクトルとは、基底行列をグラムシュミットの直交化 (ただし正規化は行わない) を施して得られる行列の列ベクトルのことである。これは、直交基底ベクトルは整数成分を持つベクトルではないので格子ベクトルではないことに注意する。基底簡約することにより、良い基底 (=より簡約された基底) のほうがサンプリング (基底ベクトルらの整数倍の結合により格子ベクトルを作ること) などによってゴールベクトルのもとまる確率が高くなることが理論的に示すことができる。基底変換によるアプローチは、多くのヒューリスティックな SVP の求解アルゴリズムで利用されており、我々のアプローチも Ducas らのアプローチも基底簡約を利用して

Position	Dimension	Euclidean Norm	Seed	Contestant	Solution	Algorithm	Subm. Date	Approx. Factor
1	180	3509	0	L. Ducas, M. Stevens, W. van Woerden	vec	Sieving	2021-02-8	1.04002
2	178	3447	0	L. Ducas, M. Stevens, W. van Woerden	vec	Sieving	2021-02-8	1.02725
3	176	3487	0	L. Ducas, M. Stevens, W. van Woerden	vec	Sieving	2020-10-13	1.04411
4	170	3438	0	L. Ducas, M. Stevens, W. van Woerden	vec	Sieving	2020-05-12	1.04690
5	158	3240	0	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	vec	Sieving	2021-01-22	1.02311
6	157	3320	0	L. Ducas, M. Stevens, W. van Woerden	vec	Sieving	2019-05-20	1.04906
7	156	3219	0	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	vec	Sieving	2021-01-22	1.01986
8	155	3165	0	M. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. Postlethwaite, M. Stevens, P. Karpman	vec	Sieving	2018-09-18	1.00803
9	154	3200	0	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	vec	Sieving	2021-02-1	1.02258
10	153	3192	0	Martin Albrecht, Leo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, Marc Stevens	vec	Sieving	2018-08-30	1.02102
11	152	3217	0	Kenji KASHIWABARA and Tadanori TERUYA	vec	Other	2018-10-3	1.03313

図 1　SVP Challenge のランキング

いる。

基底簡約をするためには、基底行列から決まる直交基底ベクトルを短くする必要がある。一つの基底ベクトルだけを短くして他のものはそのままの長さを保つことは不可能で、ある index の基底ベクトルを短くすると、後ろの index の基底ベクトルの長さも変化してしまう。それを繰り返すことで基底簡約を行うことになる。よって、ある index の直交基底ベクトルを短くするようなベクトルを見つけることが重要である。通常、基底ベクトルの整数結合を考えることにより、直交基底ベクトルを短くするような新たなベクトルを見つける。このようなアプローチをサンプリングと呼ぶ。どのように基底変換に使うベクトルを見つけるかというサンプリングにおいて、どのような整数結合の集合を取るかで基底簡約に使えるような短いベクトルが見つかるかの確率も変わってくる。我々のグループは、152 次元のゴールベクトルを見つけることに成功したプログラムにおいて、サンプリングを利用し、サンプリングに用いる係数の集合から短いベクトルが求まる確率を計算した。

3　当拠点公募型研究として実施した意義

この公募型課題として研究したことで東京大学および筑波大学の運用する大型並列計算機システム reedbush-H を借りて大規模並列計算を行うことができた。格子の次元が高い SVP 問題を解くのに高性能な大型並列計算機は欠かせない。Ducas らのアルゴリズムは g6k という名前で Github に公開されている。g6k は、大規模な sieving を効率的に行うにあたって、計算ノードごとに非常に大きな共有メモリを必要とする。sieving の大量のベクトルをすべてメモリにいれて、ベクトルの足し引きを計算するためである。それは、reedbush-H の計算機環境でも足りないぐらいのメモリを必要としていた。よって、われわれは、メモリが限定的な計算機環境でも効率的なアルゴリズムを作ることを目指した。並列ノード間で共有ファイルシス

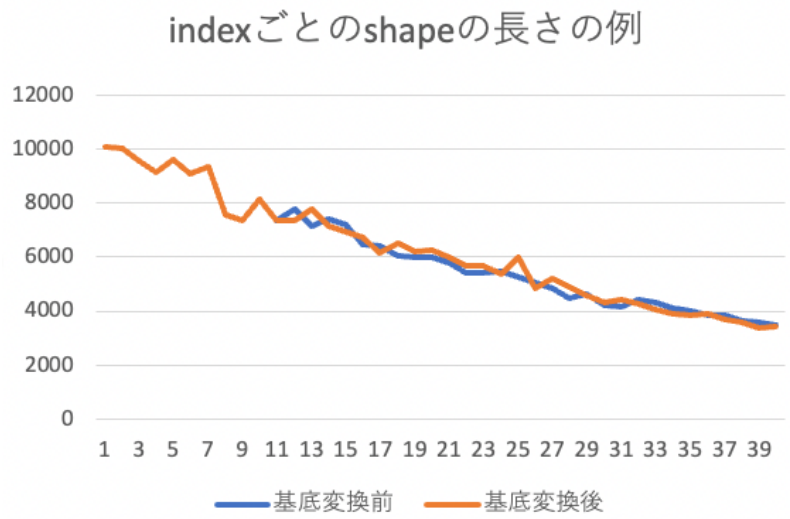


図 2　140 次元の shape(直交基底ベクトルの自乗長さ) の例。基底変換した index の以降がばらばらになる。

テムを使って情報共有するようなシステムを作ることができたが、それだけではまだ大容量メモリを使った計算性能に届いてないのが現状である。

4　今年度の研究成果の詳細

g6k の sieving エンジンを使った新しい基底簡約プログラムを作成したというのが今年度の研究の主な成果である。われわれの新しいプログラムについて記述する。まず、新たに基底簡約プログラムを作った理由を書く。g6k では短いベクトルが見つかった時に、見つかったタイミングで評価関数がよくないなどで基底変換に採用されない場合は、基本的に捨てることになる。あまり長期にわたって、前のほうの直交補空間で短いベクトルが保存されない。それを長時間保持して、あとから適用できるようにしたかったというのがプログラムを改変し始めた主要な動機である。g6k では、sieving による短いベクトルの生成能力は非常にすぐれているが、基底変換につかうベクトルを選ぶ部分は比較的シンプルである。基底変換に使われるベクトルは、各 index に対応する直交補空間において、最短なベクトルのなかから選ばれる。どの最短ベクトルを選ぶかの評価関数は、現在の直交基底ベクトルと見つかった新しいベクトルの射影長さの間の更新比率と index で決まる。その評価関数を用いて、もっともよい index の更新ベクトルを基底変換に用いるベクトルとして選択している。g6k において、基底変換しても、基底変換に使われた index より前の直交補空間の最短ベクトルはそのまま生きているし、後ろのベクトルに関しても変換して短いかどうか調べて使うこともおこなっている。しかし、g6k においても選ばれなかったベクトルの長期的保持に関して、一応行っているが、後述するように細心の注意を払ってないので、基底変換しているうちにベクトルの長さが変わってしまう。

g6k に比べてわれわれのプログラムが改善した点を書く。

・ベクトル選択評価関数変換ベクトルを選ぶときの評価方法も g6k で採用されたものと変更を加えている。g6k では各 index に対応する直交補空間で最も短いもののなかから評価関数

のもっともよいものを選択している。選択に用いる評価関数は、現在の直交基底ベクトルの長さと新しく見つかったベクトルの長さの比率の情報をもとに、index の位置を考慮して計算している。われわれのプログラムでは、各直交補空間でもっとも評価値のよいベクトルを用いて基底変換するところまでは同じである。しかし評価関数として、改善された比率ではなく、改善された幅をもとに評価関数を計算している。すなわち、その index i においての評価値の値は、現在の (index i の直交基底ベクトルの自乗長さ)-(index i の新しいベクトルの自乗長さ) をもとに計算している。さらに、index i のベクトルの評価関数において、index $(i+1)$ での改善幅を計算に取り入れていることが特徴である。われわれは各 index の直交基底ベクトルの長さの更新の比率 (新しいベクトルの自乗長さ/現在の直交基底ベクトルの自乗長さ) をデフォルトで 5 パーセント以上自乗長さが更新されないと更新しなくしている。よって少ない更新幅で更新されることがないようになっている。

・プロセス並列化効率的な基底簡約計算には、並列化は欠かせない。g6k は、簡単なプロセス並列を実行する仕組みも持っているが、それはパラメータを変えたりして多数のプロセスを同時に走らせて実験するためのもので、プロセス間で協力は行わない。g6k では効率的なアルゴリズムを実現するための手法としては、スレッド並列が基本である。スレッド並列は、メモリを共有できるので、大きな sieving で短いベクトルを見つける g6k には合致している。大きな次元の sieving には、大量のベクトルが必要であり、巨大なメモリが必須になる。しかし、われわれのコンピュータ環境では、必ずしもうまく g6k のスレッド並列実行ができなかった。よって、プロセス並列の仕組みを新たに作ることにした。それは、共有ファイルシステムに保存した基底を介して行うことにする。一つのプロセスがよい基底を見つけるとそれを他のプロセスにも共有することで協力関係を築く。ただし、まったく同じ基底を持っていると生成するベクトルも重複が増えてくるので、各プロセスがほどほどの距離を保って計算を進めていけるようにする。

プロセス間の協力方法として基底を共有する以外にたとえば、直交補空間の短いベクトルを共有することも考えられる。以前は、その方法を利用してプロセス間の協力関係を作ったこともある。あるプロセスである直交補空間で短いベクトルを見つけるとすみやかに別のプロセスに共有される。そのときは、ベクトルを共有することにより、前のほうの基底ベクトルは各プロセスである程度、共通化して、後ろのほうの基底ベクトルをばらばらにすることでプロセスごとに異なるベクトルが生成するようにしていた。そのときは、ベクトルの生成方法として Enumeration を採用していたのでそのようにする必要があった。今は、g6k の sieving をベクトル生成に使っているので、まったく同じ基底ベクトルからある程度異なるベクトルが得られるので、後ろのほうの基底ベクトルをばらばらにする必要はない。基底の評価値により、基底をプロセス間で共有するかを決めるので、単に直交補空間ごとに短いベクトルを共有するよりも、大域的な最適化ができる。

基底のよさをはかる評価関数は、基底の shape の長さ (直交基底ベクトルの自乗長さ) を index ごとに重みをかけて足し合わせたものを使う。前の index のほうが重みが重く、指数関数的に後ろにいくほど重みが小さくなっていくような評価関数を用いている。

$$\text{基底の評価値} = \sum_i \Theta^i \text{直交基底ベクトルの長さ}$$

評価関数に加えて、簡約の辞書式順序と呼ぶものも利用している。辞書式順序とは、2 つの基底が与えられたときに、shape の長さを先頭の index から比べていって、最初に短くなった方が小さいとする順序である。基底簡約はかならず辞書式順序が小さくなる方向に行なっていて、基底の置き換えも辞書式順序が小さくなる方向に限って行なっている。それにより、循環することを防いで、計算すれば計算するほど、簡約が進むように作ってある。

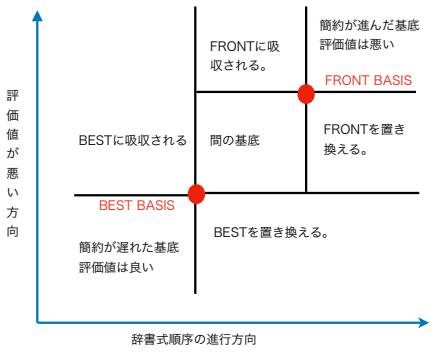
並列プロセス間の協力は次のように行う。各並列プロセスは、それぞれ独自の基底をメモリ上にもっている。最初の起動時には共通であるが、それぞれ独立して sieving を行って、乱数を利用して初期ベクトルを生成しているので結果はプロセスごとに異なるので、それを使って各プロセスで基底変換するとプロセスごとに基底がばらけてくることになる。しかし、それだけでは各プロセス間に協力関係が築けないので、仕組みを用意する。そのときどきで全体で最もよい基底をひとつ共有ファイルシステムに保存して、best basis と呼ぶことにする。並列プロセス間の協力は、基底行列をよい計算結果を出したプロセスのものに置き換えることによって行うことにする。並列プロセスの起動には MPI の仕組みを利用するが、プロセス間の情報交換は MPI 通信を行わずに、共有ファイルシステムを利用する。各プロセスは定期的に共有ファイルシステムに保存されている best basis の情報を読みにいって、簡約の辞書式順序でも基底の評価関数でもともにファイルに保存されているほうが自分のプロセスの基底よりもよい場合に、現在の基底をまったく保存しているものに置き換えてしまう。逆に辞書式順序でも評価関数でも自分の基底のほうがよい場合には、保存しているものを現在の基底に置き換えるということをおこなっている。並列プロセスの情報共有において、基底ベクトルを一部共有することにより行う方式のほかに更新ベクトルの情報共有も一部行なっている。前のほうの index の直交補空間で見つかった最短ベクトルを共有ファイルシステムのファイルに json 形式で保存し、プログラム再開時に読み込めるようになっている。

提案システムでは、並列プロセスごとに基底を持っている。基底の置き換え条件として緩いものを採用することで、ある程度、プロセスごとに基底がバラバラになることを許容している。基底簡約問題においては基底が探索点と思うことができるが、複数の探索点により、探索している形になる。オリジナルの g6k も単一の基底で探索を行うものであるが、十分な効率的な探索を行なっていることからわかるように、基底簡約の問題は、比較的、単一の探索点でも局所最適に落ち込むことがない問題といえる。しかし、提案の並列簡約システムは複数の探索点で探索を行うので、単一の探索点の場合との比較を行うことができる。われわれは、あえて複数探索点のモデルを採用し、各プロセスが自律的に協力関係を築けるようなシステムの構築を目指している。ここで自律的とは、中心になって協力関係をコントロールするような特別なプロセスを置かないということである。

基底の評価関数に加えて、簡約の辞書式順序と呼ぶものも利用している。辞書式順序とは、2 つの基底が与えられたときに、shape の値 (直交基底ベクトルの自乗長さといってもよい) を先頭の index から比べていって、最初に短くなった方が小さいとする順序である。基底簡約はかならず辞書式順序が小さくなる方向に行なっていて、提案システムにおける基底の置き換えも辞書式順序が小さくなる方向に限って行なっている。それにより、計算が循環することを防いで、計算すれば計算するほど、簡約が進

むように作ってある。辞書式順序は全順序なので、基底の集合をを一列に並べることができ、一つの基底よりも辞書式順序で小さい基底は有限個なので、簡約を進めていくことでいつかは最小の基底にたどり着くことになる。

共有ファイルシステムに保存される best basis は、各プロセスの保持している基底と比べて、評価関数もよいし、辞書式順序でも進んでいるときに、各プロセスの基底を置き換えることになる。逆に各プロセスの保持している基底のほうが、best basis よりも、評価関数もよいし、辞書式順序でも進んでいるということになれば、保存されている best basis が、プロセスの保持している基底で上書きされることになる。すなわち、2 つの条件を同時に満たすことが基底の置きかわりの条件となる。よって評価値はよいけど、簡約がすすんでないとか、逆に評価値は悪いけど簡約がすすんでいるとか、どちらがすぐれているか判断できないことがよく起こる。このようなゆるい条件を採用することにより、基底の多様性を保証している。条件としてはゆるいが、簡約も進んでないし、評価値も悪いような基底を排除することができる。また、各プロセスの基底がバラバラになった状態になっても、一つのプロセスの基底が、評価値もとてもよく、簡約も進んでいるようなものになったら、他のプロセスの基底もその基底に置き換えられることになる。



長く計算していると、前のほうを更新したものの評価値がなかなか下がらないプロセスなど、best basis から外れてくる基底も出てくる。それらの間でも協力関係が築けるような仕組みを用意している。best basis に加えてもう一つ情報共有のための基底を利用する方法である。辞書式順序と、基底の評価関数の 2 つの指標で、他の基底に負けないものは複数ありうるので、best basis だけではなく、複数基底をファイルシステムに保持する方法も考えられる。提案システムでは 2 つの基底を保存する方式を実装している。best basis に加えて、front basis と呼ぶ基底を保存している。これは、best basis よりも辞書式の基底簡約は進んでいるけども、評価値は悪いものになっている。これにより、探索枝の分枝においても、プロセス間で協力関係を構築することができる。各プロセスは、まず best basis と比べて、簡約の辞書式順序と評価値の両方がともに上回っているかどうかで、基底の置き換えを行う。保存されているものよりも、辞書式順序はよいけれども、評価値が悪いときに、今度はプロセスが保持している基底を front basis と比べて、best basis と同じように、簡約の辞書式順序と評価値の両方がともに上回っているかどうかで、基底の置き換えを行う。2 つの基底を使うことにより、各プロセスが集まる場所がふたつできて、協力計算から外れるようなプロセスを減らすことができる。

5　今年度の進捗状況と今後の展望

研究計画の段階では、独自の sieving のプログラムを作成する予定で、年度の前半には独自の sieving のプログラムを作成していた。それは GPU の計算も利用したものであった。利用していた並列コンピュータ環境ではノード間でメモリ共有ができないので、独自の sieving のプログラムでは、共有ファイルシステムを利

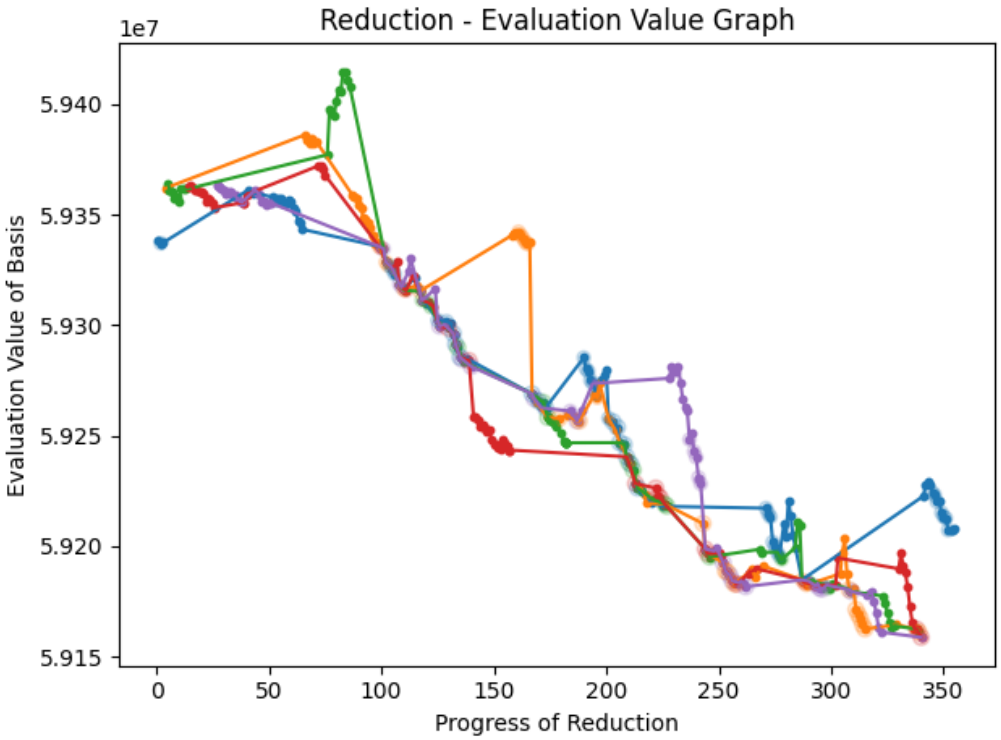


図 3　140 次元で 4 つのプロセスで協調しながら評価値を改善する計算例。横軸は、基底変換の進みぐあいで基底をソートした順番。ぼやけている点は BEST BASIS の場所を表す。

用して、ノード間で情報共有した。index ごとに sieving vector を持つというのが特徴であった。しかし、性能が思ったように上がらなかった。その理由をいまから考えるとベクトル数の不足を補う方法がなかったからなど複数の問題点があった。2020 年度の途中に方向転換して、g6k を改造して g6k の sieving エンジンを利用したプログラムを作ることにした。g6k を元にしたプログラムを作って比較したことで、独自の sieving プログラムは、基本コンセプトの index ごとに sieving vector を持つことと、ファイルシステムを通じて大量のベクトルを共有する方式自体に問題があることがわかった。当初の予定だった並列計算ノードごとのすべてのメモリを利用して sieving することが達成できていないので、自己評価としては達成率は半分程度である。今後も理論的な研究も併せて行なっていきたい。参加研究メンバーの松田氏の発表した国際会議論文も確率的な考察の論文である。

6　研究業績一覧（発表予定も含む）

国際会議プロシーディングス　（査読あり）

Yoshitatsu Matsuda, ‘Optimization of Search Space for Finding Very Short Lattice Vectors’ , Lecture Notes in Computer Science Vol 12231, pp. 149–161, 2020