完全準同型暗号を用いたゲノム秘匿検索の bootstrap 処理における SSD 適用に向けた検討

辻 有紗　（指導教員：小口 正人）

1　はじめに

近年クラウド上でのデータの機密性を保障したビックデータ解析に向けて, 完全準同型暗号 (以下:FHE) を用いた秘密計算技術の実用化に向けた研究が盛んに行われている. FHE の課題として, 時間空間計算量が膨れ上がることが挙げられるが, その中でもボトルネックは bootstrap [1] と呼ばれる処理である. bootstrap とは暗号文同士を演算する際に蓄積するノイズを取り除くため複数の鍵を用いて暗号化・復号化を行う処理を指し, 実行には膨大なメモリ容量を要する.

FHE は従来, DRAM を多く使用するためクラウドで活用される手法だが, VM や Container の利用によって１台のリソースを分け合うため, 使用効率は 65 %程度にとどまる [2]. その上, DRAM1bit あたりの値段は SSD の約 10 倍とコストが高い. そこで, SSD から随時 IO を行い DRAM 使用量を削減することで, 高効率化・低価格化が可能である. また, 並列性の高いアプリケーションであれば, CPU からの load/store 命令の宛先を, DRAM の代わりに並列実行可能な SSD へ割り当てることで高速化される場合がある.

本研究ではゲノム秘匿検索アプリケーション [3] を対象として, bootstrap を中心にアプリケーション内のそれぞれ の処理の傾向に合わせて SSD を用いることで, 実行にかかるコストを削減し, 高速化可能であるか調査を行う.

2　ゲノム秘匿情報検索

アプリケーションについて述べる. クライアントサーバ 型通信で, クライアントがサーバのゲノムデータベース (PBWT) に対して特定の塩基配列が最長マッチを持つか問い合わせを行う. この時, お互いに持っているゲノム塩基配列や調べたい内容を秘匿しながら行うことが FHE により実現され, 個人情報の漏洩が防がれている. FHE 処理では膨大なメモリ容量が必要な一方で, クラウドでは適当に DRAM が割り当てられないことが多く, スワップ処理が発生する場合がある [2].

3　評価環境

CPU	Xeon E5-2643 v3
	(3.40 GHz × 6 Cores) × 2
L1 (i,d)cache	32KB, 64B/line
L3 cache	20480KB, 64B/line
DRAM	DDR4, 512GB
Docker1	memory 512GB

表 1: 評価環境

ゲノムデータベースについては 1 サンプルあたり 10000 文字のデータを 2184 サンプル用意した. また, プログラム中で bootstrap 処理の割合を大きくすることを目的に, 検索クエリの長さは 1, データベース上の検索開始ポジション数は 1 に設定した. 4 節の評価は全て Docker1 のコンテナ上で行った.

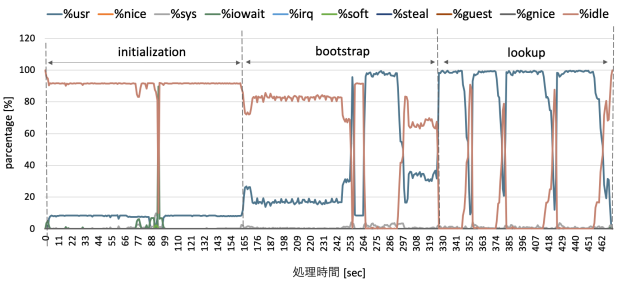

図 1: CPU 処理内訳

4　評価

4.1　アプリケーションの傾向

図 1 に実行時間に対する CPU 処理の内訳を示す. initialization では検索するための準備を行うが, server は client による FHE コンテキストや公開鍵の作成を待機する時間が長いため idle 値が高く推移する. その後, disk から公開鍵や暗号文を読み込むため, iowait も発生している. 続いて bootstrap は Helib ライブラリの recrypt 関数で実装されているが, 前半では処理の多重度が低く CPU の最大の性能を使用していない. 後半は暗号化・復号化処理により引き起こされる CPU の計算処理が重く, usr 値が高い. bootstrap の内部の処理内容について今後調査を行う必要ある. lookup はゲノム配列データベースに対応するテーブルの更新や暗号化されたクエリとの照合を行う. 終始 usr 値が高く推移しており, CPU の計算処理が重いことがわかる.

ここで, 処理全体を通した各値の平均値ではストレージの I/O コストを示す iowait は処理全体の 0.2 %と低く, メモリサイズが大きく swap が発生しない環境ではストレージ IO は問題とならない. また, idle 値が処理全体の 57.51 %を占めており CPU 使用率が低いことから, bootstrap や lookup 処理の割合が低い場合はオーバースペックとなることがわかる. 実際は, PBWT や検索クエリのサイズが大きく, bootstrap や lookup の負荷が大きいため, usr 値の割合が大きい.

4.1.1　予備実験

予備実験として、hyperthread を有効にし, usr 値が高い箇所について, CPU の演算処理とメモリへの load/store 処理の処理の重さを比較した. hyperthread のような仮想的な並列化では計算性能は約 2 倍となるが, メモリへの load/store は逐次処理となる. 実行時間と CPU コストの変化を表 2 に示す. 有効時の方が実行時間が増加し, 特に計算処理が少ない initialization で増加する. これは, 2 スレッドが 1 つのコア上で動作するためタスクが頻繁に切り替わり, コンテキストスイッチの回数が約 2 倍に増加することが原因である. lookup

	無効時	有効時
実行時間 (s)	1167	1636
IPC	2.846	2.75
context switch	118084	138410
cpu migration	170.3	1704
iTLB load 数	15949790	49929917
L3 cache load 数	30810878861	33593106846
L3 cache load misses(%)	5.11	5.27

表 2: hyperthead 有効時・無効時の比較

と bootstrap についても処理時間が増加するが, CPU コストの変化は見られないことと IPC の低下から, メモリへの load/store がボトルネックとなり, 2 倍になった CPU の計算処理性能を活用できていないことがわかる. 従って, メモリへの load/store を並列に行うため, 処理を並列化し, DRAM を並列 IO が可能な SSD に取り替えることが有効である.

4.2 並列化可能性の調査

独立性の高い計算部分を OpenMP を用いてマルチスレッド化し, 実行時間や CPU の負荷の計測を行なった. 図 2 に実行時間の推移と速度向上率を示す. ベンチマークを用いて実験を行ったところ, 計測環境では RAM ディスクへの IO 処理 は, シーケンシャルまたはランダムな読み書きそれぞれについて, 並列数 128 程度でスループットが最大になる. 一方, このアプリケーションについては論理 CPU 数 12 まで並列数に伴い減少し, その後は 増加に転じる. 並列化による CPU コストが変化し, オーバーヘッドが大きいことが原因である.

また, 処理の一部は依存性が高く, 逐次処理となるため, 速度向上効率は一部の非効率部分により律速 される. 今回は initialization の並列度が低く, bootstrap と lookup の並列度は非常に高い. ここで, 実際は検索クエリ長やポジション数は大きい値の場合が多く, FHE 処理特に bootstrap 処理の割合が大きくなる. 図 4 に, 検索クエリ長の変化における bootstrap 処理の割合の増加を示す. 以上から, プログラム内で並列処理が可能な割合は大きく, 並列 I/O 処理による高速化率は高いと予想される.

4.3 並列化による処理コストの変化

並列化による CPU コストの変化が顕著なものの 1 つに context-switch があり, 並列数が 1 増加するごとに処理全体を通して約 10 万回線形増加する. 並列数 12 でも 1 分あたり 850 回程度までしか上らないことから, コンテキストスイッチによるオーバーヘッドは問題にならない. また, 並列数の増加に伴い, L3 キャッシュロードミス率は 5.11 %〜8.53 %, L3 キャッシュストアミス率は 5.16 %〜24.13 %まで上昇する. 並列に行われる方が短期間にアクセスされるメモリ領域が広いためメモリにないデータにアクセスする頻度が上がることが要因である. ここで, L3 load ミス率に比べ L3 store ミス率の上昇具合が大きいことから, 書き込みの方が新しいメモリ領域を使う傾向があり, 読み込みは局所性が高いことがわかる.

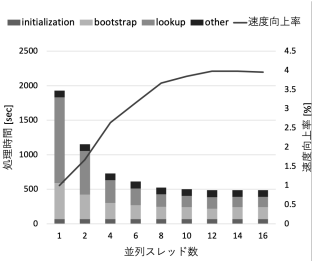

図 2: 並列化による速度向上率

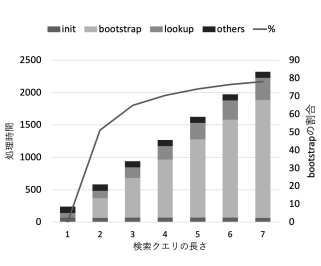

図 3: クエリ長の変化に対する処理内訳の推移

一度にアクセスするメモリ容量をキャッシュサイズと比較するため, 公開鍵や暗号文のサイズについて調査した結果を図 4 に示す. 初めの検索クエリの暗号化に伴いサイズが約 2600 倍ほど増加し, 暗号化文同士の演算が開始する. ここで検索クエリについて一文字ずつ, PBWT と照合し, 続けて bootstrap による暗号文のノイズの削除を行なっている. グラフから, boostrap 処理後には毎回 12MB 程度まで大きくなり, 照合により 5MB 程度に減少することがわかる. 原因については今後調査する必要がある.

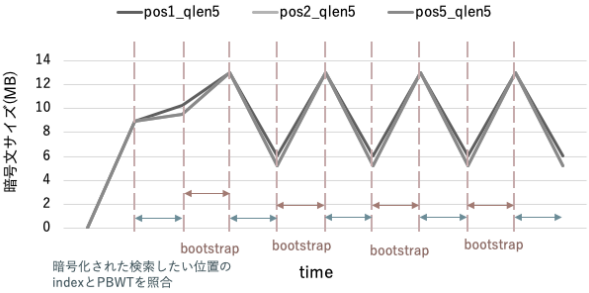

図 4: 処理に伴う暗号文サイズの変化

5 まとめと今後の課題

DRAM と SSD の良い特性を引き出せるようなハードウェアの使い分けやプログラムの改良により, 高速実行できる可能性について調査を行った. メモリへの load/store 処理が重く, 並列処理可能な範囲が広いことから, SSD 並列化が活用できる可能性が高い. 今後は direct IO を用いた SSD による並列処理を実装し, SSD と DRAM について処理状態の比較を行う.

謝辞

本研究の一部は, キオクシア株式会社の支援を受けて実施したものである.

参考文献

[1] Craig Gentry, ”A FULLY HOMOMORPHIC ENCRYPTION SCHEME”, Stanford University, 2019

[2] Muhammad Tirmazi and Adam Barker and Nan Deng and Md.E Haque and Gene Qin and Steven Hand and Mor Harchol-Balter and John Wilkes, ”Borg: the Next Generation”, Fifteenth European Conference on Computer Systems (EuroSys ’20), pp.1-4, 2020

[3] Y. Ishimaki et al., “Privacy-preserving string search for genome sequences with FHE bootstrapping optimization.” 2016 IEEE International Conference on Big Data (Big Data). IEEE. 2016, pp. 3989–3991.