高帯域幅メモリーを搭載したインテル・プロセッサーのパフォーマンス最適化

インテル® Xeon® CPU マックス・シリーズのパフォーマンス・チューニングの詳細

Vamsi Sripathi インテル コーポレーション ソフトウェア・イネーブリング & 最適化エンジニア

量子色力学（QCD）は、素粒子間の強い力の相互作用の理論 / 研究です。格子 QCD は、粒子と力を空間領域と時間領域で離散化した格子として表すことにより QCD 問題を解きます。HotQCD は、高エネルギー物理学の研究コミュニティーで広く使用されている C++ ハイブリッド MPI/OpenMP* 格子 QCD シミュレーション・フレームワークです。

この記事では、インテル® Xeon® CPU マックス・シリーズで最適なパフォーマンスを実現するために HotQCD に適用するパフォーマンス・チューニング手法を説明します。インテル® Xeon® CPU マックス・シリーズとほかのインテル® Xeon® プロセッサーとの主な違いは、高帯域幅メモリー（HBM）の有無です（**図 1a**）。簡単に言えば、HBM は DDR メモリー（シングルスタック DRAM）よりも高いメモリー帯域幅パフォーマンスを実現する 3D スタック DRAM インターフェイスです。インテル® Xeon® CPU マックス・シリーズは（CPU ソケットあたり）最大 56 のコアと、ソケットあたり 8 個の高 DRAM ダイを 4 つスタックした HBM2e を搭載しています。スタックの各 DRAM ダイの容量は 2GB です（ソケットあたり 4 x 8 x 2 = 64GB HBM）。

パフォーマンス解析

HBM を搭載したインテルのプロセッサーには、メモリー（フラット、キャッシュ、HBM のみ）および NUMA（SNC1、SNC4）など、多くの構成モードがあります。各モードの詳細は本題から外れるためここでは詳しく取り上げません。詳細は、「インテル® Xeon® CPU マックス・シリーズ構成およびチューニング・ガイド」（英語）を参照してください。この記事で使用したシステムは、HBM のみメモリーモード（DDR5 なし）と SNC4（サブ -NUMA クラスタリング -4）で構成されています（**図 1b**）。

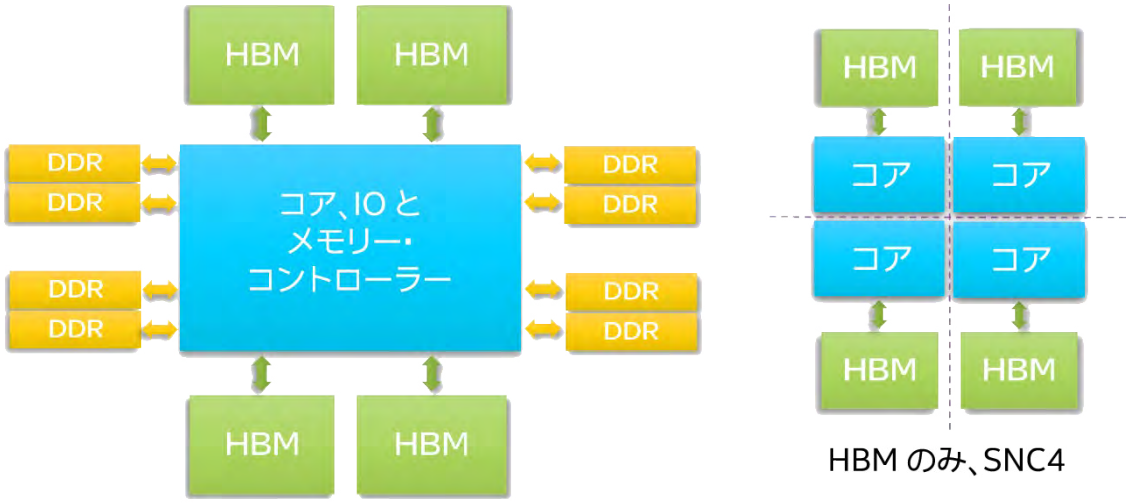
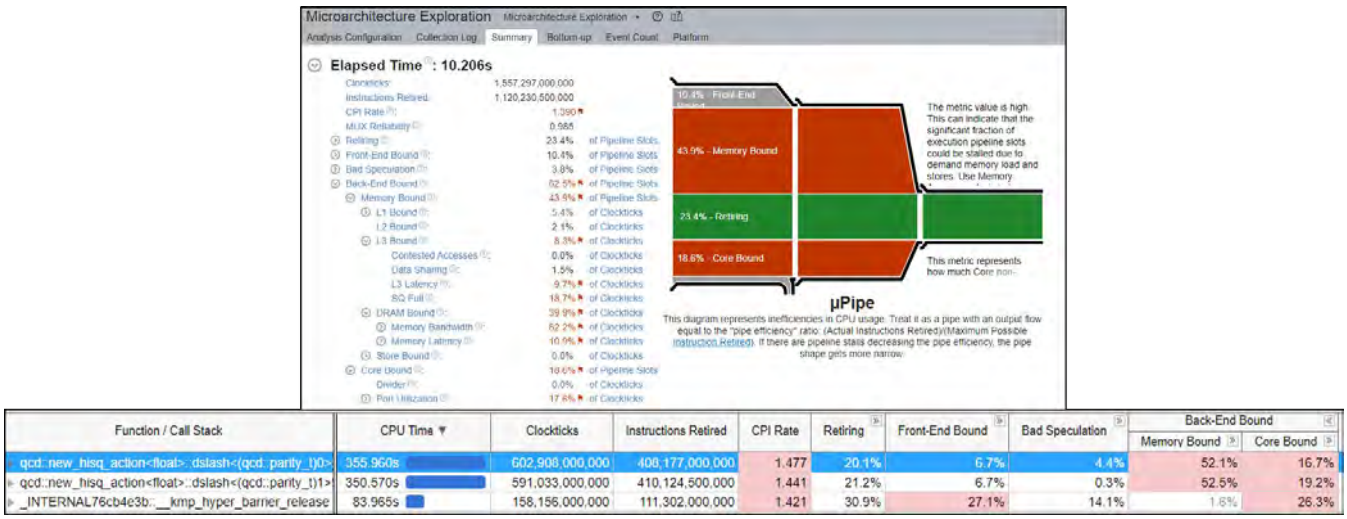


図 1.（a）インテル® Xeon® CPU マックス・シリーズ
（b）インテル® Xeon® CPU マックス・シリーズ（HBM のみ + SNC4 モード）

HotQCD のパフォーマンス・スナップショット [1 つの RHS（右辺）ベクトルを格子サイズ 32^4（x=y=z=t=32）でベンチマーク] は、最も時間を費やしている関数（dslash、合計実行時間の 90% を消費）がメモリー帯域幅依存である（プロセッサーがメモリー操作の待機時間の約 50% でストールしている）ことを示しています（**図 2**）。



Function / Call Stack	CPU Time ▼	Clockticks	Instructions Retired	CPI Rate	Retiring	Front-End Bound	Bad Speculation	Back-End Bound	
								Memory Bound	Core Bound
qcd::new_hisq_action<float>::dslash<(qcd::parity_t)0>	355.960s	602,908,000,000	408,177,000,000	1.477	20.1%	6.7%	4.4%	52.1%	16.7%
qcd::new_hisq_action<float>::dslash<(qcd::parity_t)1>	350.570s	591,033,000,000	410,124,500,000	1.441	21.2%	6.7%	0.3%	52.5%	19.2%
_INTERNAL76cb4e3b::__kmp_hyper_barrier_release	83.965s	158,156,000,000	111,302,000,000	1.421	30.9%	27.1%	14.1%	1.6%	26.3%

図 2. インテル® VTune™ プロファイラーのマイクロアーキテクチャー全般解析を使用したベースラインのパフォーマンス・スナップショット

dslash の OpenMP* 並列領域を**図 3** に示します。格子の各点で、演算子のオーバーロードにより 4 つの密行列 - ベクトル積のセットが実行されます。関数はインテル® AVX-512 組込み関数で完全にベクトル化されていて、スレッド間の同期は行われません。link_std には 2 つの行列 - ベクトル積があり、行列とベクトルはそれぞれ 9 つと 3 つのキャッシュラインで構成され、合計で 4 x 2 x (9 + 3) = 96 のキャッシュラインが読み取られます。link_naik は link_std と似ていますが、行列が 7 つのキャッシュラインのみをロードして構成され、合計で 4 x 2 x (7 + 3) = 80 のキャッシュラインが読み取られる点が異なります。すべてのメモリーアクセスは、無視できる量のデータ再利用でキャッシュラインにアライメントされます。これは、この関数がメモリー帯域幅依存で（FLOP とバイトの比率が約 0.9）、大量の読み取りトラフィックが発生していることを明確に示しています [176(96 + 80)のキャッシュラインがすべて読み取られ、3 つのキャッシュラインのみメモリーに書き込まれます]。つまり、プロセッサーが浮動小数点演算を実行するには、メモリーからデータを継続的に読み取る必要があります。


```
// Baseline/default version
    #pragma omp parallel for schedule(static)
    for ( int i = 0; i < no_sites_bsf; ++i ) {

        const int siteh_bsf = siteh_bsf_list[i];


        color_vector_simd<float_type> v, w;

        #pragma forceinline recursive
        for ( int imu = 0; imu < 4; ++imu ) {

            const new_site_shift site_up = site.shift_eo<new_up>( imu );
            const new_site_shift site_dn = site.shift_eo<new_dn>( imu );

            v += _link_std.get_eo(       site    , imu )          * vec_in.get_shift( site_up );
            v -= _link_std.get_shift_eo( site_dn, imu ).dagger() * vec_in.get_shift( site_dn );
        }

        #pragma forceinline recursive
        for ( int imu = 0; imu < 4; ++imu ) {

            const new_site_shift site_up = site.shift3_eo<new_up>( imu );
            const new_site_shift site_dn = site.shift3_eo<new_dn>( imu );

            w += _link_naik.get_eo(       site    , imu )          * vec_in.get_shift( site_up );
            w -= _link_naik.get_shift_eo( site_dn, imu ).dagger() * vec_in.get_shift( site_dn );
        }


        #pragma forceinline recursive
        vec_out.stream( c_std * v + c_naik * w, siteh_bsf );
    }
```

図 3. HotQCD dslash のコード

メモリー・アクセス・パターン

図 4 は、内部（4 次元格子の各点）ループと外部（格子点）ループの `link_std` ブロックの操作シーケンス（行 1517 と 1518）を示しています。同じ色は連続したメモリー領域を示し、異なる色は（一時的なメモリーアクセスのリクエストと比較して）メモリーの大きなストライドを示します。各行列（9 つのキャッシュライン、CL と表記）は、外部ループ全体で 1,172 要素の一定ストライドで連続したメモリーからロードされます。1,172 の FP32 要素のストライド（1,172 x 32 バイト = 4,688 バイト）により、外部ループの反復ごとに新しい 4KB ページにアクセスすることになります。(内部ループで）ロードされる連続するベクトルも、大きなストライドにより分離されます。しかし、外部ループの反復空間では連続したアドレスストリームを形成します。行列と比較すると、ベクトルは 4KB ページをまたぐジャンプがない分、アクセスパターンは優れています。最後に、各行列 - ベクトル積は 27 のインテル® AVX-512/ZMM レジスターで完全にアンロールされます。行列には 3 x 3 x 2 (複素数の実部と虚数部) = 18 の ZMM レジスターが必要です。ベクトルには、3 x 1 x 2 = 6 つの ZMM レジスターと、内部ループの結果の累積用に 3 つの ZMM レジスターが必要です。

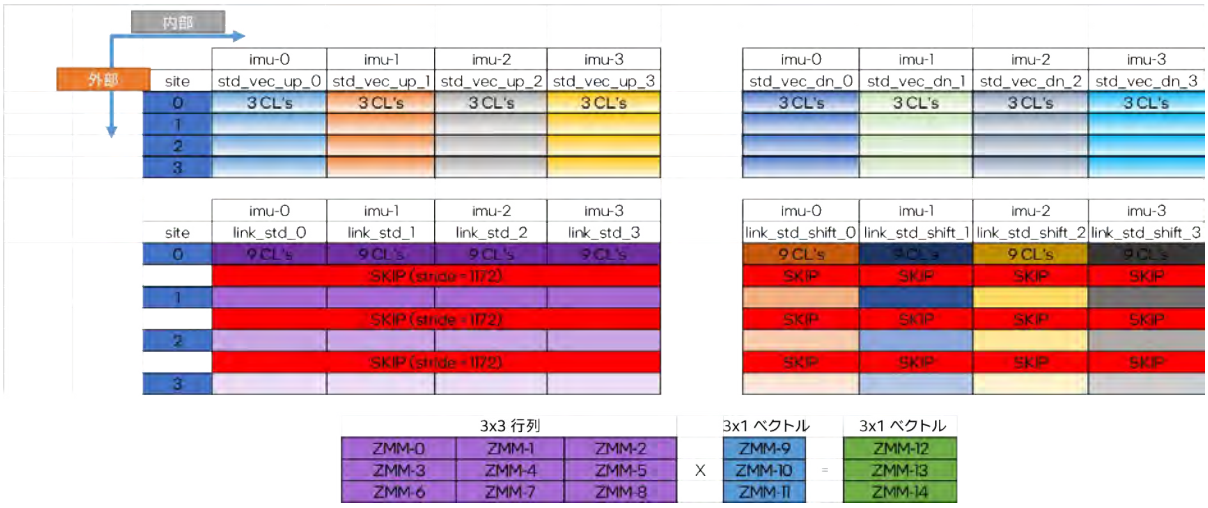


図 4. dslash のメモリー・アクセス・パターン

ソフトウェア・プリフェッチ

インテルのプロセッサーには、ストリーミングとストライド・アクセス・パターンの両方を検出できるさまざまなハードウェア・プリフェッチャー（L1$、L2$）がありますが（詳細は「インテル® 64 アーキテクチャーおよび IA-32 アーキテクチャー最適化リファレンス・マニュアル」を参照）、それらはすべて 4KB のページ境界内でプリフェッチし、ページをまたいでデータをフェッチしません。HotQCD の行列はページ境界を越えるストライドでアクセスされるため、ハードウェア・プリフェッチの効率は低下します。ループ本体に複数のストライド・アクセス・ストリームが存在するとさらに効率が悪化し、ハードウェア・プリフェッチャーにさらなるストレスがかかります。そこで、大きなアクセスストライドの影響を軽減するため、明示的なソフトウェア・プリフェッチを使用した場合のパフォーマンスへの影響を調査することにしました。対応する計算操作でデータが消費される前にメモリーからデータを先にフェッチするプリフェッチ命令（x86 ISA（英語）で利用可能）を発行することにより、ハードウェア・プリフェッチャーの有効性を高めることを目標とします（**図 5**）。

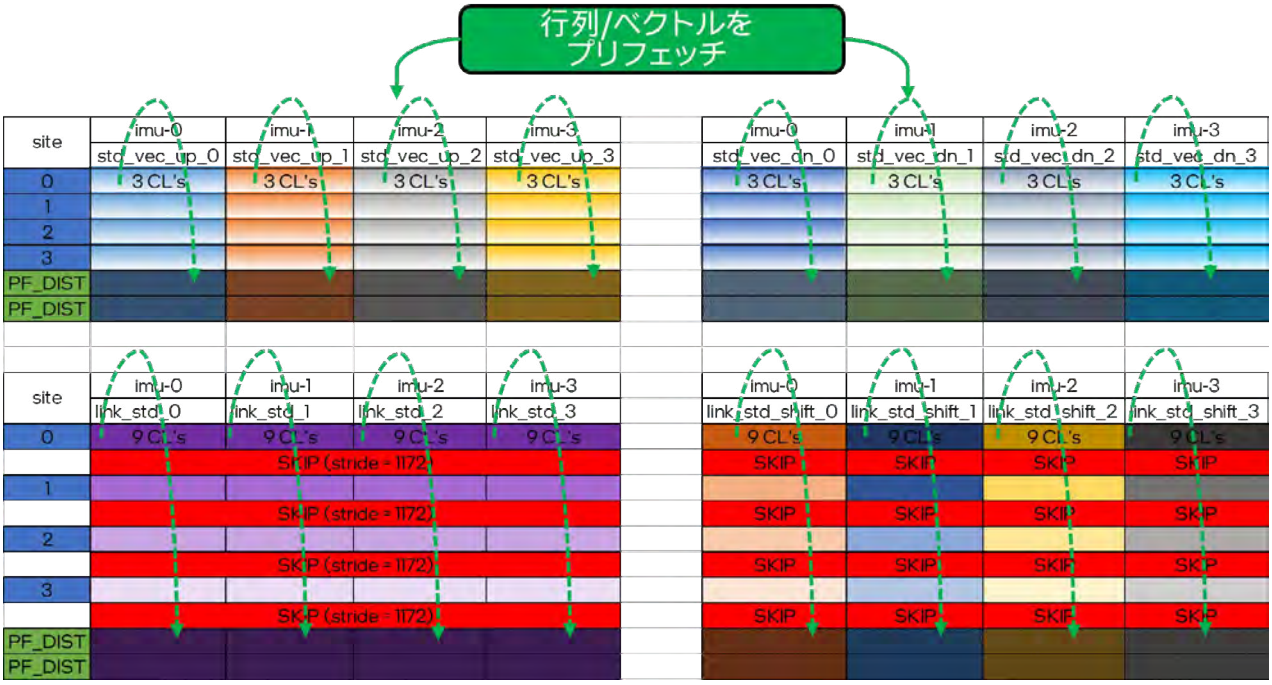


図 5. dslash とソフトウェア・プリフェッチ

ソフトウェア・プリフェッチの有効性は、主に次の 2 つの要因により決まります。

- **プリフェッチ距離：**理想的なプリフェッチ距離は、ワーキングセットのサイズ、ループ本体の命令のレイテンシー、フェッチされるデータの場所（メモリー / キャッシュ）などの要因により決まります。プリフェッチが遠すぎるとプリフェッチされたデータが中間反復ワーキングセットによりキャッシュから退避される可能性がありますが、プリフェッチが近すぎるとメモリー階層のレイテンシーが隠蔽されません。どちらの場合も、プリフェッチ命令で生成される追加のメモリーアクセス要求によりソフトウェア・プリフェッチの有用性やパフォーマンスが低下し、すでに飽和しているメモリー・パイプラインのキューにさらに負担がかかります。
- **キャッシュ階層：**要求されたデータが配置されるキャッシュレベルを制御します。x86 ISA には、L1、L2、および最終レベルのキャッシュ（LLC）にデータを配置できるプリフェッチ命令があり、一時的でないアクセスの下位レベルでのキャッシュ汚染を最小限に抑える追加の制御が利用できます。一部のアプリケーションでは、メモリーから L2 キャッシュへのデータのプリフェッチには長いプリフェッチ距離を使用し、L2 から L1 キャッシュへのデータのプリフェッチには短いプリフェッチ距離を使用する、マルチレベルのプリフェッチ・メカニズムが有益です。

アプリケーションでソフトウェア・プリフェッチを使用する主なメカニズムとして、インテル® コンパイラーのオプションとプリフェッチ組込み関数の 2 つがあります。ここでは、細かい制御が可能な後者を選択しました。**図 6** は、距離とキャッシュ階層が異なる場合の、ソフトウェア・プリフェッチ + ハードウェア・プリフェッチャーのベースライン・パフォーマンス（ハードウェア・プリフェッチャーのみ使用）に対するスピードアップを示しています。10 回先の反復（サイト）から L2 キャッシュにデータをプリフェッチすることで、1.13 倍のスピードアップが得られます。

PF_DIST	キャッシュヒント	
	L1$	L2$
1	0.91	1.06
2	0.89	1.10
4	0.91	1.09
6	0.91	1.10
8	0.89	1.12
10	0.90	1.13
12	0.91	1.10
14	0.91	1.06
16	0.91	1.11
18	0.91	1.10
20	0.90	1.10
22	0.90	1.06
24	0.90	1.11

図 6. ソフトウェア・プリフェッチによるパフォーマンスの向上

メモリーレイアウト

HotQCD の重要な操作は、間接的な反復共役勾配（CG）法を使用して大きな行列の逆行列を計算し、線形システムを解くことです。CG が収束するまでにかかる反復は約 2,000 回です。各 CG 反復で、(前述した) 4 つの行列 - ベクトル積を計算し、次に MPI ランク間でハロー交換を行うことにより、更新されたベクトルを計算します。CG の反復全体にわたり、行列は変更されず、ベクトルのみ更新されます。最初の CG 反復では、行列をパフォーマンスの高い非ストライド / パケット形式にコピーするパターンを活用します。後続の反復では、パックされたバッファーから行列をロードします（**図 7** および**図 8**）。

図 7. dslash のパックド行列レイアウト

図 8. パックド行列レイアウトのコード

連続した行列メモリーレイアウトを利用することにより、内部ループと外部ループをまたぐストライドアクセスを回避します。ハードウェア・プリフェッチャーはネイティブレイアウトの行列（ベースライン）において、4KB を超えるストライドで大きなストレスを受けるため、この処理は特に有益です。CG には約 2,000 回の反復がかかるため、最初の反復のコピーコストは相殺されます。ネイティブレイアウトの行列をパックされたバッファーにコピーする必要があるため、追加のメモリー（テストした問題サイズでは 1GB）を割り当てる必要があります。ただし、HotQCD の合計ランタイム・メモリー・フットプリントは、ソケットあたり 64GB の HBM 容量内に収まるため、これは制限要因ではありません。

図 9 は、ベースラインとパックド・メモリー・レイアウトのインテル® VTune™ プロファイラーのスナップショットを比較したものです。パックド・メモリー・レイアウトでは、命令あたりのサイクル数（CPI）が低くなり（0.4 対 1.4）、メモリーリクエストでストールする実行スロットが少なくなります（28% 対 44%）。

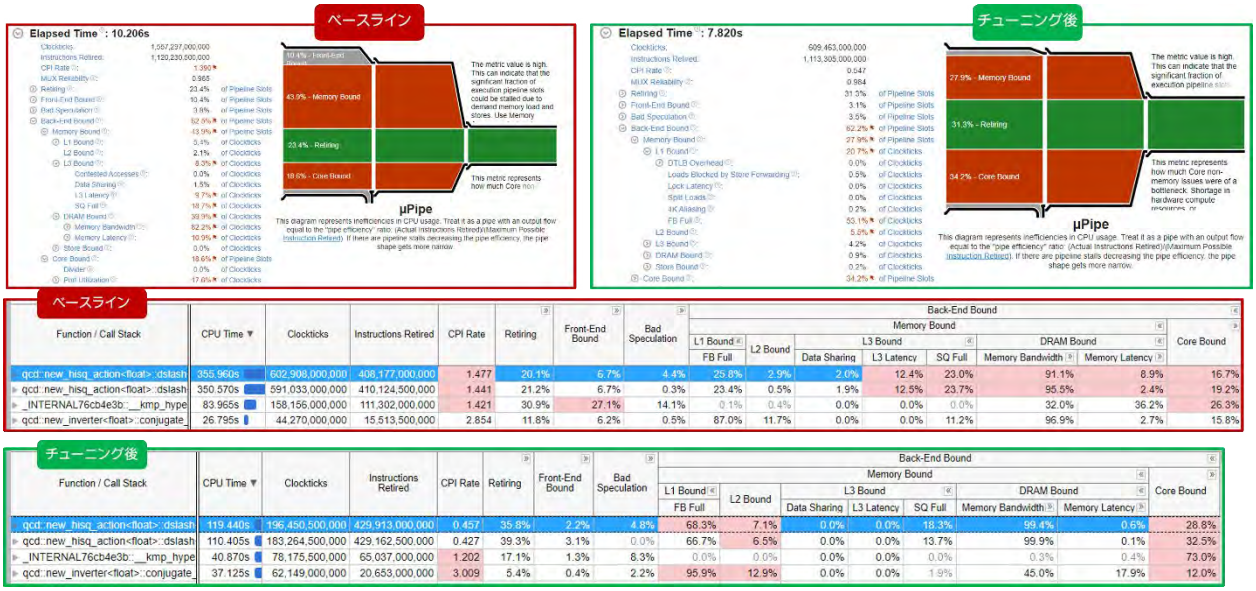


図 9. ベースラインとパックドレイアウトのインテル ® VTune ™ プロファイラーでの比較

パックド・メモリー・レイアウトを採用することにより、HotQCD のパフォーマンスはベースラインから 1.38 倍向上します（**図 10**）。`dslash` カーネルでは 1.54 倍向上しますが、パックドレイアウトがキャッシュ汚染を引き起こし、`dslash` に続く後続のベクトル演算が遅くなるため、ベンチマークの数値をそのまま受け取ることはできません。この最適化を HBM を搭載していないインテル® Xeon® システムに適用すると、パフォーマンスはベースラインから 1.21 倍向上しました（**図 11**）。

コンポーネント	ベースライン	ベースライン + プリフェッチ	パックド形式 + プリフェッチ
dslash	1	1.14x	1.54x
HotQCD	1	1.13x	1.38x

図 10. インテル® Xeon® CPU マックス・シリーズにおける HotQCD の相対的なパフォーマンス向上

	インテル® Xeon® プロセッサー (HBM なし)	インテル® Xeon® マックス・シリーズ (HBM 搭載) によるスピードアップ
ベースライン	1.00	1.66
ベースライン + プリフェッチ	1.14	1.64
パックド形式 + プリフェッチ	1.21	1.88

図 11. インテル® Xeon® プロセッサーとインテル® Xeon® CPU マックス・シリーズにおける HotQCD のパフォーマンス

まとめ

最後に、要点をまとめます。

- HBM を搭載したインテル® Xeon® CPU マックス・シリーズは、同等のインテル® Xeon® プロセッサー (DDR5) と比べてパフォーマンスが大幅に (ベースラインで 1.66 倍) 向上します。
- プリフェッチにより、HBM のパフォーマンスはさらに向上します (HBM ベースラインから 1.13 倍、最適化 + HBM なしから 1.88 倍)。
- プリフェッチのパフォーマンスを適切にチューニングするには、アプリケーションのメモリー・アクセス・パターンを理解することが不可欠です。プリフェッチ命令は低コストですが、使用には注意してください。
- 4KB ページ境界をまたぐ大きなメモリー・アクセス・ストライドは、HBM のパフォーマンスにとって理想的ではありません。ストライドアクセスよりもメモリーの連続したチャンクを読み取ることを優先することで、ベースラインの 1.38 倍から 1.54 倍の最適な HBM パフォーマンスを達成できます。