インテル® Xeon® 6（P コア搭載）の HPC アプリケーション向けの構成とチューニング・ガイド

2026 年 4 月

資料番号: 858491
ビジョン 1.4

注意事項:

本ドキュメントはレイアウト調整および校閲を行っておりません。誤字脱字、製品名や用語の表記、レイアウト等の不具合が含まれる可能性があることを予めご了承ください。

この日本語マニュアルは、インテル コーポレーションのドキュメント・センターで公開されている『Intel® Xeon® 6 with P-cores Configuration and Tuning Guide for HPC Applications』の参考訳です。インテル社の許可を得て iSUS（IA Software User Society）が翻訳版を作成した iSUS の著作物です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

著作権と商標について本資料には、開発の設計段階にある製品についての情報が含まれています。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとしないでください。
インテルのテクノロジーを使用するには、対応したハードウェア、特定のソフトウェア、またはサービスの有効化が必要となる場合があります。
本資料に記載されているインテル製品に関する侵害行為または法的調査に関連して、本資料を使用または使用を促すことはできません。本資料を使用することにより、お客様は、インテルに対し、本資料で開示された内容を含む特許クレームで、その後に作成したものについて、非独占的かつロイヤルティー無料の実施権を許諾することに同意することになります。
本資料は、(明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず) いかなる知的財産権のライセンスも許諾するものではありません。
本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。
性能の測定結果はシステム構成の日付時点のテストに基づいています。また、現在公開中のすべてのセキュリティー・アップデートが適用されているとは限りません。構成の詳細は、補足資料を参照してください。絶対的なセキュリティーを提供できる製品またはコンポーネントはありません。
性能は、使用状況、構成、その他の要因によって異なります。詳細については、パフォーマンス・インデックス・サイト (英語) を参照してください。 コスト対効果は異なる場合があります。
「コンフリクト・フリー」とは、当社のデュー・デリジェンスに基づき、コンゴ民主共和国または近隣諸国の武装勢力に直接的または間接的に資金提供または利益をもたらすタンタル、錫、タングステン、または金 (米国証券取引委員会が「紛争鉱物」と呼ぶもの) を含有せず、またはこれらを調達していない製品、サプライヤー、サプライチェーン、製錬所、および精製所を指します。
すべての製品計画とロードマップは、予告なく変更される場合があります。
コード名は、開発中で一般に公開されていない製品、テクノロジー、またはサービスを識別するためにインテルによって使用されます。これらは「商用」の名称ではなく、商標として機能することを意図したものではありません。
インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、知的財産権の非侵害性への保証、およびインテル製品の性能、取引、使用から生じるいかなる保証を含みますが、これらに限定されるものではありません。
(オプションの ENERGY STAR) ENERGY STAR は、米国環境保護庁によって定義されたシステムレベルのエネルギー仕様であり、プロセッサー、チップセット、電源など、すべてのシステム・コンポーネントに関係します。詳細については、エネルギースターのウェブサイトをご覧ください。
(オプションのインテル® サーマル・ベロシティー・ブースト (インテル® TVB)) インテル® サーマル・ベロシティー・ブーストの効果が含まれます。この機能は、プロセッサーが最大温度に対しどれだけ低い状態で動作しているか、またターボ電力バジェットが利用可能かどうかに基づいて、シングルコアおよびマルチコアのインテル® ターボ・ブースト・テクノロジーの周波数よりもさらに高いクロック周波数へ状況に応じて自動的に引き上げます。周波数の上昇幅と持続時間は、ワークロード、プロセッサーの性能、およびプロセッサーの冷却方式によって異なります。
(オプションのインテル® ターボ・ブースト・テクノロジー) インテル® ターボ・ブースト・テクノロジーを利用するには、インテル® ターボ・ブースト・テクノロジーに対応したプロセッサーを搭載した PC が必要です。インテル® ターボ・ブースト・テクノロジーのパフォーマンスは、ハードウェア、ソフトウェア、およびシステム全体の構成によって異なります。使用する PC がインテル® ターボ・ブースト・テクノロジーに対応しているかどうかは PC メーカーに確認してください。詳細については、http://www.intel.com/technology/turboboost (英語) を参照してください。
(推定結果に関する免責事項) 結果は推定またはシミュレーションによるものです。
(サードパーティーのデータに関する免責事項) インテルは、サードパーティーのデータを管理または監査するものではありません。精度を評価するには、他のソースを参照してください。
(オプションの Principled Technologies 社のベンチマーク開示に関する免責事項) インテルは、Principled Technologies 社が運営する BenchmarkXPRT 開発コミュニティーを含む、さまざまなベンチマーク・グループへの参加、スポンサー活動、および技術支援を通じて技術ベンチマークの開発に貢献しています。

本書で参照されている文書番号付きの文書のコピーは、インテル・リソース・アンド・ドキュメント・センターから入手できます。

目次

改訂履歴

改訂番号	説明	日付
1.0	最初のドキュメント	2025 年 6 月
1.1	SPACK のインストール手順を更新し、oneAPI 2025.3 コンパイラーと SPACK バージョン 1.0 を使用するようにしました。	2025 年 12 月
1.2	金融サービス業界 (FSI) 向けアプリケーションのレシピを更新し、SPACK を使用するようにしました。QuantLib 向けのレシピを追加しました。	2025 年 12 月
1.3	oneAPI ツールに関するセクションを更新しました (5 章)。	2026 年 1 月
1.4	SIMULIA PowerFlow 向けのレシピを追加しました 。	2026 年 4 月

1 パフォーマンスおよび最適化に関する免責事項

性能結果はインテルによるテストに基づいており、現在公開中のすべてのセキュリティー・アップデートが適用されているとは限りません。詳細は、システム構成を参照してください。絶対的なセキュリティーを提供できるコンピューター・システムはありません。

インテルは、このドキュメントで言及されている第三者のベンチーマーク・データや Web サイトの設計や実装を管理もしくは監査することはありません。参照先のウェブサイトを訪問し、参照するデータが正確であるかどうかを確認する必要があります。

性能は、使用状況、構成、その他の要因によって異なります。詳細については、
www.intel.com/PerformanceIndex（英語）をご覧ください。

最適化に関する注意事項: インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2（インテル® SSE2）、インテル® ストリーミング SIMD 拡張命令 3（インテル® SSE3）、ストリーミング SIMD 拡張命令 3 補足命令（SSSE3）命令セットに関連する最適化およびその他の最適化が含まれます。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。
注意事項の改訂: 20110804

インテル® アドバンスト・ベクトル・エクステンション（インテル® AVX）は、特定のプロセッサー操作において高いスループットをもたらします。プロセッサーの電力特性が異なるため、インテル® AVX 命令を使用すると、
A） 一部の部品が定格周波数以下で動作し、
B） 一部の部品がインテル® Turbo Boost Technology 2.0 を使用しても、
ターボ周波数が最大にならない場合があります。実際の性能はハードウェア、ソフトウェア、およびシステム構成によって異なります。詳細は、http://www.intel.com/go/turbo（英語）をご覧ください。

インテル® ハイパースレッディング・テクノロジー（インテル® HT テクノロジー）は、一部のインテル® プロセッサーで利用可能です。また、インテル® HT テクノロジー対応システムが必要です。パフォーマンスは、使用するハードウェアおよびソフトウェアによって異なる場合があります。インテル® ハイパースレッディング・テクノロジーの詳細については、こちらをご覧ください。

2 はじめに

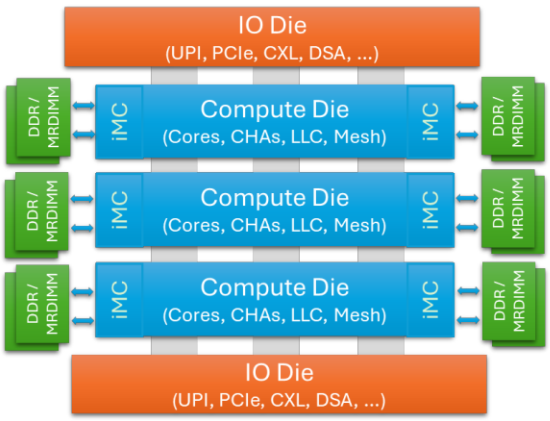
2.1 目的

このドキュメントでは、インテル® Xeon® 6 (P コア搭載、コードネーム: Granite Rapids) プロセッサー上で動作するハイパフォーマンス・コンピューティング (HPC) アプリケーションのチューニングおよび最適化に関するガイドラインを示しています。このガイドには、すべての HPC アプリケーションに関する一般的な BIOS および Linux の構成ガイドラインに加え、厳選された HPC アプリケーションおよびベンチマークのビルドと実行手順 (レシピ) が含まれています。

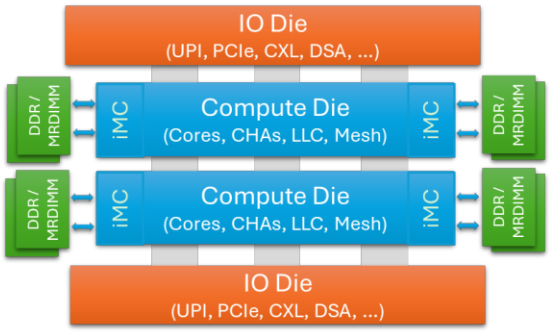
2.2 アーキテクチャー概要

インテル® Xeon® 6 (P コア搭載、コードネーム: Granite Rapids) は、インテル® Xeon® マルチコア・サーバー・プロセッサーの最新製品です。インテル® Xeon® 6900 シリーズ (P コア搭載) は、最大 52 ビットの物理アドレス空間と 57 ビットの仮想アドレス空間をサポートする一方、インテル® Xeon® 6700 シリーズ (P コア搭載) は、最大 52 ビットの物理アドレス空間と 48 ビットの仮想アドレス空間をサポートし、少なくとも 1 つのプロセッサーで構成されるプラットフォーム向けに設計されています。

以下に示すように、6900 シリーズ・プロセッサーは 3 つの演算ダイで構成され、6700 シリーズ・プロセッサーは 2 つの演算ダイで構成されています(6500 シリーズはコア数が少ないため、1 つの演算ダイで構成されます)。1 つの演算ダイには、L1 および L2 キャッシュを備えた演算コア、L3 キャッシュの一部、および統合メモリー・コントローラー (IMC) が含まれます。



6900-series Processor Block Diagram



6700-series Processor Block Diagram

これらのプロセッサーは、Compute Express Link (CXL) 2.0 と統合 I/O (IIO) もサポートしています。ほとんどのプロセッサー・タイプは、最大 96 レーンの PCI Express* 5.0 (32GT/秒) と 8 レーンの DMI をサポートしていますが、R1S プロセッサー・タイプは最大 136 レーンの PCI Express 5.0 をサポートしています。各 IMC は、最大 2 チャネルの DDR5 DIMM をサポートし、チャネルあたり最大 2 個の DIMM を搭載できます。インテル® Xeon® 6900/6700 シリーズ (P コア搭載) は、チャネルあたり 1 つの DIMM で MRDIMM (マルチプレックス・ランク DIMM) もサポートしています。

プロセッサーは複数の IO インターフェイス・モジュールと CHA/コアモジュールで構成され、これらのモジュールは水平方向と垂直方向の相互接続からなるメッシュに接続されており、各相互接続は双方向チャネルで構成されます。プロセッサーの SKU（製品番号）により、モジュールの組み合わせが異なる場合があります。

インテル® Xeon® 6（P コア搭載）は、第 4 世代および第 5 世代で使用された Golden Cove および Raptor Cove コアをベースとした、Redwood Cove コアを搭載しています。コアには、容量 2 MB のミッドレベル・キャッシュ（MLC）が搭載されています。

第 5 世代マイクロアーキテクチャー以降、入出力（I/O）アーキテクチャーが改良されています。PCI Express* (PCIe*) Gen5 は合計で 88～136 レーンあり、これは第 4 世代インテル® Xeon® ファミリーで利用可能な 80 レーンの PCIe Gen5 と比較して増加しています。

インテル® Xeon® 6（P コア搭載）の製品情報および性能結果の詳細は、こちら（英語）をご覧ください。

2.3 追加のチューニングリソース

インテルでは、さまざまなサーバー・ワークロード向けのチューニング・ガイドを公開しています。これらのガイドは、既にワークロードに精通しており、パフォーマンスを向上させるシステム設定を模索しているユーザー向けに作成されています。

データベース、AI フレームワーク、HPC クラスター、ネットワーク・ワークロード・システム向けのチューニング・ガイドは、以下のリンクからオンラインで入手できます:
https://software.intel.com/content/www/us/en/develop/articles/xeon-performance-tuning-and-solution-guides.html（英語）

インテルが開発した最適化ライブラリーとツールキットは、AI、ML、DL フレームワークや多数のプログラミング環境向けに提供されており、さまざまなワークロードのパフォーマンスを向上できます。詳しい情報へのリンクを以下に示します:

最適化ライブラリーとツール
インテル最適化 AI ソフトウェア（英語）
インテル® ディストリビューションの Python （NumPy、SciPy、sckit-learn を含む）（英語）
PyTorch、TensorFlow、MXNet、PaddlePaddle、Caffe などの人気のあるディープ・ラーニング・フレームワーク（英語）
インテル® AI Analytics および OpenVINO™ ツールキットのインテル® Distribution を含むツール、ライブラリー、および SDK（英語） oneAPI（英語）– コンパイラー、ライブラリー、および移植、分析、デバッガーのツールセット。 インテル® リソース・ディレクター・テクノロジー（インテル® RDT）（英語）共有キャッシュおよびメモリーリソースの監視と割り当てを行うフレームワーク。

3 ハードウェアの構成

3.1 目的

本章では、ハードウェアの選択方法と BIOS の設定オプションについて説明します。

3.2 DRAM の選択と構成

インテル® Xeon® 6900 シリーズ（P コア搭載、コードネーム: Granite Rapids）は、6400 MT/秒の DDR5 および 8800 MT/秒の MRDIMM をサポートします。インテル® Xeon® 6700 シリーズ（P コア搭載）は、6400 MT/秒の DDR5 および 8000 MT/秒の MRDIMM をサポートします。

DDR5 では、最高のメモリー帯域幅を実現するため、デュアル・ランク・メモリーの使用を推奨します。

注: 32GB の DDR5 モジュールは、DDR5 モジュールのエラー訂正コード（ECC）ビットの格納およびアクセス方法の違いにより、さらに大容量のモジュールよりもわずかに高い書き込み帯域幅を提供します。

注: MRDIMM は、6900 シリーズと 6700 シリーズの両方のシステムで、可能な限り最高の帯域幅を提供します。同時に、MRDIMM は Granite Rapids システムにおいて、DDR5 よりも低いレイテンシーを実現します（帯域幅の使用率に関わりなく）。

最高のパフォーマンスを達成するには、すべてのメモリーモジュールが同じ仕様、および同じメーカー製であることを確認してください。

3.2.1 確認方法

`dmidecode`（4.3.6 節）および/または `lsmem`（4.3.6 節）を使用して、DRAM の種類と構成情報を確認します。例えば、`dmidecode` コマンドは、システムに搭載されている各 DRAM モジュールの容量、速度（MT/秒）、ランク数、製造元、部品番号/シリアル番号を表示します。

インテル® メモリー・レイテンシー・チェッカー（5.8 節）、STREAM ベンチマーク（6.4 節）、および HPCG ベンチマーク（6.6 節）を使用して DDR のパフォーマンスを検証します。

3.3 BIOS 構成

このセクションでは、HPC アプリケーションのパフォーマンスに関連する BIOS オプションと設定について説明します。特定の BIOS で用意されている BIOS オプションとデフォルト設定は、BIOS の提供元によって異なることに注意してください。

3.3.1 サブ NUMA クラスタリング（SNC）

6900 シリーズのプロセッサーは 3 つの演算ダイで構成され、6700 シリーズのプロセッサーは 2 つの演算ダイで構成されています。1 つの演算ダイには、演算コア（それぞれが専用の L1 および L2 キャッシュを備える）、L3 キャッシュの一部、および 4 つのメモリーチャネルをサポートする統合メモリー・コントローラー（IMC）が含まれます。

BIOS メニューで起動時に選択できるサブ NUMA クラスタリング（SNC）モードを設定すると、各演算ダイを個別の物理 NUMA ノードとして認識させることができます。Birch Stream プラットフォームでは、メニューから SNC を有効/無効にできます:

EDKII から、[Socket Configuration] -> [Uncore Configuration] -> [Uncore General Configuration] -> [SNC] を選択します

インテル® Xeon® 6900 シリーズ（P コア搭載）はプロセッサーあたり 3 つの NUMA ノードをサポートし、6700 シリーズ（P コア搭載）はプロセッサーあたり 2 つの NUMA ノードをサポートします。2 ソケット（プロセッサー）を備えた一般的なシステムでは、6900 シリーズでは合計 6 つの NUMA ノード、6700 シリーズでは合計 4 つの NUMA ノードをサポートします。

SNC モードでは各演算ダイが個別の NUMA ノードとして公開されるため、特定の NUMA ノードのコア上で実行されるソフトウェアは、デフォルトで同じ演算ダイに接続されたメモリーにアクセスできるようになります。これにより、メモリーへのアクセス・レイテンシーが低減され、コアが利用できるメモリー帯域幅が増加します。

SNC は、複数の SNC ノードを活用するため、NUMA に対応したソフトウェアを必要とします。ほとんどの HPC ソフトウェアは NUMA に対応しているため（例えば、MPI プロセス、MPI+OpenMP、またはアンサンブルを使用するなど）、HPC アプリケーションは SNC のメリットを比較的容易に享受できます。例えば、MPI（または MPI+OpenMP）アプリケーションは、各 NUMA ノード上で少なくとも 1 つの MPI プロセス（ランク）を使用する必要があります。同様に、OpenMP アプリケーションでは、ファースト・タッチ・ポリシーを使用して、特定のスレッドで使用されるデータが同じ NUMA ノードに割り当てられるようにできます（たとえば、OpenMP スレッドによってアクセスされるデータ項目が同じ OpenMP スレッドによって初期化されるなど）。

SNC 設定を確認するには、`numactl` ユーティリティー（4.3.2 節）を‘`numactl -H`’コマンドとともに使用します。2 ソケット構成の 6900 シリーズシステムでは 6 つの NUMA ノードが、2 ソケット構成の 6700 シリーズシステムでは 4 つの NUMA ノードが表示されるはずです。さらに、各 NUMA ノードで使用可能なメモリー量をチェックして、メモリーのインストールが対称的であることを確認してください。

アプリケーションが NUMA に対応しているか確認するには、`numastat` ユーティリティーを使用します（4.3.3 節）。アプリケーションが完全に NUMA に対応している場合、‘`numa miss`’カウンターは実行中に増加しません（つまり、アプリケーション実行前と実行後の `numa_miss` 値は同じです）。

3.3.2 ターボ構成

最高のパフォーマンスを得るには、BIOS でターボ機能を有効にする必要があります。ターボ機能により、プロセッサーは基本周波数よりも高い周波数で、TDP（熱設計電力）の制限値まで動作させることがで

きます。

プロセッサーの実際の動作周波数と消費電力を確認するには、turbostat（4.3.4 節）ユーティリティー（例: turbostat -qS）を使用します。

Linux の lscpu（4.3.1 節）ユーティリティーを使用すると、ターボ周波数の最大値と最小値を表示できます。

注: Birch Stream BIOS では、最高のパフォーマンスを得るために、[*Energy Efficient Turbo*]（エネルギー効率ターボ）機能を無効にすることを推奨します。

3.3.3 ファン構成と CPU 温度

6.5 節で説明する High Performance LINPACK（HPL）など、CPU 負荷の高いベンチマークを実行しながら、turbostat（4.3.4 節）を使用して CPU パッケージの温度を確認します。

CPU の温度が低いほど、CPU は効率良く動作します。これは、動作温度が高くなると、リーク電力消費量が増加するためです。

BIOS のファン設定（利用可能な場合）は、最適な空気の循環を実現するため、高パフォーマンスに設定し、デューティサイクル（「PWM オフセット」と呼ばれることもあります）を高い値（90% 以上）に設定する必要があります。Birch Stream プラットフォームでは、BIOS メニューから PWM オフセットを設定できます:

EDKII から [Platform Configuration] -> [Miscellaneous Configuration] -> [Fan PWM offset] を設定します

HPL（6.5 節）のような CPU 負荷の高いアプリケーションの実行中に、プロショットイベント（サーマル・スロットリング・イベント）が発生していないことを確認するため、'dmesg -T'（参照）を使用します。さらに、HPL の実行中は、turbostat ユーティリティーを使用してプロセッサーの温度を監視してください。

3.3.4 レイテンシー最適化モード

レイテンシー最適化モードでは、プロセッサーはアンコア（L3 キャッシュ、メモリー・コントローラー、IO ダイ）の利用率があまり高くない場合でも、アンコアを高い周波数で動作させることができます。これにより、アンコアをあまり利用しないアプリケーションでは、メモリーおよび I/O 操作（UPI 転送を含む）のレイテンシーが低減されます。これは、MPI 通信やメモリー・レイテンシーがボトルネックとなるアプリケーションにおいて有益となる可能性があります。しかし、これは利用率の低いシナリオでは消費電力の増加につながります。

レイテンシー最適化モードは、起動時の BIOS オプションを使用するか、システム起動後に以下のスクリプトを使用して有効/無効にできます（root 権限が必要です）:
https://github.com/intel/pcm/blob/master/scripts/bhs-power-mode.sh（英語）

Birch Stream BIOS の場合、レイテンシー最適化モードはメニューから設定できます:

EDKII から [Socket Config] -> [Advanced Power Mgmt config] -> [CPU - Advanced PM tuning] -> [Latency Optimized mode] を設定します

上記のスクリプトでレイテンシー最適化モードを設定すると、アプリケーションの実行ごとに必要に応じて個別に有効/無効にできるため、柔軟性が高まります。

レイテンシー最適化モードが適用されていることを確認するには、root 権限で以下のコマンドを使用して、各ダイ（IO ダイを含む）のアンコア周波数を確認できます。

```
cat /sys/devices/system/cpu/intel_uncore_frequency/uncore0*/current_freq_khz
```

レイテンシー最適化モードが適用されている場合、システムがアイドル状態または低負荷時であっても、すべてのダイのアンコア周波数は最大になります。IO ダイの最大周波数（例: 2500MHz）は、演算ダイのアンコア周波数（例: 2200MHz）とは異なることに注意してください。

3.4 ネットワーク構成

クラスター環境では、ソケットごとにネットワーク・アダプターをインストールすることで、ネットワーク・アクセスの帯域幅向上とレイテンシーの低減が可能です。レイテンシー最適化モード（3.3.4 節）を使用すると、一部のアプリケーションの MPI 通信レイテンシーを低減できます。

ネットワークのパフォーマンスは、インテル® MPI ベンチマーク（5.3 節）を使用して検証でき、アプリケーションのパフォーマンスは、MPI ライブラリーに付属のユーティリティー（5.3 節）を使用して調整できます。

4 Linux システムの構成

4.1 目的

本章では、HPC アプリケーションのパフォーマンス向上に使用できる、便利な Linux ユーティリティーと Linux オプションについて説明します。

4.2 Linux ディストリビューションとカーネル

最高のパフォーマンスを達成するには、最新の安定版 Linux カーネルの使用を推奨します。最低でも、6.x 系のカーネルを使用する必要があります。このガイドに掲載されているレシピは、カーネルバージョン 6.8 以降を使用してテストされています。

4.3 有用な Linux ユーティリティー

4.3.1 lscpu

この標準 Linux ユーティリティーは、NUMA ノード、コア数、ベース周波数、ターボ周波数、キャッシュサイズ、CPU フラグ（機能）など、高レベルのシステム設定の詳細を表示します。

4.3.2 numactl

Linux ユーティリティーの `numactl` は、システムの NUMA 構成を観察し、特定の NUMA ノードでアプリケーションを実行する場合に使用されます。システムの NUMA 構成を確認するには、`numactl -H` を使用します。以下は、`numactl -H` の出力例です。NUMA ノード数、CPU コア数、各 NUMA ノードのメモリー容量が表示され、その後に各ノードと他のノードの間の距離を表す行列が続いています。詳細は、`man numactl` を参照してください。

```
$ numactl -H
available: 6 nodes (0-5)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
 38 39 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 2
67 268 269 270 271 272 273 274 275 276 277 278 279
node 0 size: 257670 MB
node 0 free: 254438 MB
node 1 cpus: 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 7
4 75 76 77 78 79 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304
 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
node 1 size: 257977 MB
node 1 free: 256622 MB
node 2 cpus: 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110
 111 112 113 114 115 116 117 118 119 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339
 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
node 2 size: 258020 MB
node 2 free: 256790 MB
node 3 cpus: 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
 146 147 148 149 150 151 152 153 154 155 156 157 158 159 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
node 3 size: 258020 MB
node 3 free: 257030 MB
node 4 cpus: 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
 186 187 188 189 190 191 192 193 194 195 196 197 198 199 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414
 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439
node 4 size: 258020 MB
node 4 free: 256942 MB
node 5 cpus: 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479
node 5 size: 257994 MB
node 5 free: 256606 MB
node distances:
node   0   1   2   3   4   5
  0:  10  15  17  21  28  26
  1:  15  10  15  23  26  23
  2:  17  15  10  26  23  21
  3:  21  28  26  10  15  17
  4:  23  26  23  15  10  15
  5:  26  23  21  17  15  10
```

4.3.3 numastat

Linux ユーティリティー numastat は、NUMA メモリー使用量に関するさまざまな統計情報を提供します。特に、以下のコマンドが便利です（詳細は man numastat を参照）。

- numastat -p <process_name は、以下に示すように特定のプロセスのメモリー使用量を表示します:

```
$ numastat -p python

Per-node process memory usage (in MBs)
PID                         Node 0          Node 1          Node 2
----------------  --------------- --------------- ---------------
5132 (tuned)                10.54            1.21            1.21
68100 (python3)             11.88            0.00            0.00
----------------  --------------- --------------- ---------------
Total                       22.42            1.21            1.21
```

- `numastat -m` は、システム全体のメモリー使用量を表示します。
- `numastat`（引数なし）は、NUMA ヒット/ミス（ブートからの累積）を表示します。これは、予期せぬパフォーマンス低下を引き起こす可能性のある NUMA ノードの交差（`numa_miss`）につながるページ割り当てを特定するのに役立ちます。これらの統計はブートからの累積であるため、特定のアプリケーションの実行が NUMA ミスに遭遇したかどうか確認するには、各のアプリケーションの実行前後で `numastat` を実行する必要があります。

```
$ numastat
                           node0           node1
numa_hit              7038233469      7552949520
numa_miss               31491495               0
numa_foreign                   0        31491495
interleave_hit         254896206       254893381
local_node            6905622525      7430407252
other_node             164102439       122542268
```

4.3.4 turbostat

`turbostat` ユーティリティーを使用して、root として（または `setuid` バイナリーとして）実行した場合の、x86 アーキテクチャー・プロセッサーの消費電力、周波数、温度を調べることができます。`turbostat` は、システム冷却の問題を特定するのに役立ちます。例えば、以下はシステムの電力、周波数、温度を表示します。

- `turbostat -qS` `# 両方の CPU の平均値`
- `turbostat -qS --cpu package` `# CPU ごと個別に`

```
$ turbostat -qS
Avg_MHz Busy%   Bzy_MHz TSC_MHz IPC     IRQ     SMI     POLL    C1ACPI  C2ACPI  POLL%   C1ACPI% C2ACPI% CPU%c1  CPU%c6  CoreTmp PkgTmp  PkgWatt RAMWatt PKG_%   RAM_%
1337    49.92   2679    1800    1.68    599693  0       5       669     8110    0.00    0.05    49.92   50.08   0.00    75      75      697.92  172.62  190.03  0.00
1341    49.93   2676    1807    1.70    601152  0       7       1038    8791    0.00    0.07    50.07   50.07   0.00    73      73      702.72  174.93  191.08  0.00
```

4.3.5 htop

`htop` ユーティリティーは、標準 Linux の `top` ユーティリティーのようなツールですが、個々の CPU コア/スレッドの使用状況を視覚的に表示します。これは、NUMA の使用状況や MPI ランク、OpenMP スレッドの配置を特定するのに役立ちます。さらに、システムのメモリー使用量も表示します。

4.3.6 dmidecode、lshw、および lsmem

これらの Linux ユーティリティーを (root 権限で) 使用することで や DDR モジュールなど、搭載されているハードウェア・コンポーネントを検査できます。

4.3.7 lstopo

lstopo ユーティリティーは hwloc ライブラリーの一部であり、標準的なパッケージ・マネージャー（dnf install hwloc など）を使用して、パッケージとしてインストールできます。lstopo ユーティリティー（または lstopo-no-graphics）は、システムのハードウェア・トポロジーを表示します。

4.4 Linux 構成のオプション

このセクションでは、推奨されるさまざまな Linux の設定オプションについて説明します。

4.4.1 透過なヒュージページ（THP）

ほとんどの HPC アプリケーションは THP の恩恵を受けます。これは次のコマンドで有効にできます:

```
echo always >/sys/kernel/mm/transparent_hugepage/enabled
```

4.4.2 ウォッチドッグ・タイマー

Linux カーネル 6.12 より古いバージョンには、4 つ以上の NUMA ノードを持つシステムでクロックソース（TSC）の健全性チェックから発生する誤警報を修正するパッチ（英語）が含まれていません。これにより、クロックソースが低速な HPET タイマーに切り替わる可能性があり、一部のアプリケーションのパフォーマンスが著しく低下する可能性があります。カーネルバージョンが 6.12 より古い場合、この予期せぬ問題を回避するため、次のカーネル・ブート・オプションを使用する必要があります:
tsc=nowatchdog

4.4.3 周波数ガバナー

intel_pstate ドライバー（デフォルト設定）を使用する場合、周波数スケーリングのガバナーを‘performance’に設定してください。

```
/bin/sh -c "for CPU in /sys/devices/system/cpu/cpu[0-9]*; do echo 'performance' >
\"$CPU/cpufreq/scaling_governor\"; done"
```

4.4.4 サブ NUMA クラスタリング（SNC）

（3.3.1 節）で説明されているように、NUMA 対応アプリケーションでは BIOS で SNC を有効にする必要があります。Linux で SNC を有効にするには、追加の手順は必要ありません。‘numactl -H’を使用して、SNC の設定が適切であることを確認してください。

4.4.5 自動 NUMA バランス調整

自動 NUMA バランス調整は、ほとんどの Linux システムではデフォルトで有効になっており、これによりページが 1 つの NUMA ノードから別の NUMA ノードへ自動的に移行される可能性があります。こ

のような自動移行は、NUMA 向けに最適化されたアプリケーションにおいて予期せぬ動作を引き起こす可能性があります。

ページの自動移行が不要な場合は、手動で無効にできます。現在の状況は、以下の方法で確認できます:

```
cat /proc/sys/kernel/numa_balancing
```

4.4.6 スワップ領域

一般的に、ハイパフォーマンス・アプリケーションではスワップ領域を無効にするべきです。スワップはパフォーマンスを著しく (そして予期せず) 低下させる可能性があるためです。スワップ領域は、`/etc/fstab` ファイル内のスワップエントリーをコメントアウトして再起動することで、完全に無効化できます。また、スワップは `swapoff` コマンドを使用して無効にすることもできます（ただし、再起動後に設定が保持されません）。詳細については、‘`man swapoff`’をご覧ください。

4.4.7 GeoPM

GEOPM（Global Extensible Open Power Manager）は、ユーザーと管理者がプロセスセッションの期間中、ハードウェア設定を安全かつ確実に変更できるようにするソフトウェア・フレームワークです。この独自の機能により、ユーザー主導による動的かつパフォーマンスの調整が可能になり、従来のシステムツールでは不可能なエネルギー効率の向上とワークロードの最適化を実現します。

GEOPM は大規模システム（例: Aurora）で実際に運用されており、HPC センターやクラウドプロバイダーによって、ユーザーや管理者が電力、パフォーマンス、持続可能性に関する目標達成に役立てられています。

特に、メモリー負荷の高い HPC アプリケーションにおいて、GeoPM は CPU パッケージの消費電力を削減し、パフォーマンスへの影響を最小限に抑えながらシステム全体の消費電力を低減するのに役立ちます。GeoPM に関する詳細情報はこちら（英語）、メモリー負荷の高いアプリケーションにおける電力管理の具体的な例についてはこちら（英語）をご覧ください。

5 コンパイラー、ライブラリー、ミドルウェア、およびツール

5.1 インテル® oneAPI ツールキット

インテル® コンパイラーおよび開発ライブラリーは、インテル® oneAPI ツールキットに含まれています。基本ツールキットと HPC ツールキットの両方の使用を推奨します。

インテル® oneAPI における最新の最適化により、データセンターとクライアント環境の両方でスケーラブルなハイブリッド並列処理が実現し、リアルタイム画像処理、大規模言語モデルおよび画像生成における推論性能の向上が可能になります。主なイノベーションには、インテル® Core™ Ultra プロセッサーおよび Arc™ GPU 向けの oneMKL、oneDNN、PyTorch の最適化、インテル® Xeon® 6 プロセッサーのパフォーマンス向上、そしてインテル® VTune™ プロファイラーにおける新しい解析ツールなどがあります。その他のアップデートでは、SYCL と Vulkan および DirectX 12 との相互運用性、OpenMP 6.0 および Fortran 2023 コンパイラーの機能強化、CUDA から SYCL への移行の効率化、ハイブリッド並列処理向けの MPI サポートの拡張などが行われており、これらはすべて AI およびデータ集約型ワークロードの効率性、柔軟性、互換性を最大限に高めるように設計されています。

5.2 インテル® oneAPI コンパイラー

インテル® Xeon® 6 (P コア搭載) のサポートは、2023 年後半にリリースされたインテル® oneAPI DPC++/C++ コンパイラーおよびインテル® Fortran コンパイラーのバージョン 2024.0 で導入されました。それ以来、最新バージョンを含む各コンパイラーのリリースでは、このアーキテクチャーに対する継続的な機能強化と最適化が行われてきました。

インテル® コンパイラーは、特定のプロセッサー・アーキテクチャー向けにコード生成を最適化する複数のオプションを提供し、アプリケーションがインテル® Xeon® 6 (P コア搭載) の機能を最大限に活用できるようにします。

5.2.1 ターゲット・アーキテクチャー・フラグ

ターゲット・アーキテクチャー・フラグは、高度なベクトル命令やマイクロアーキテクチャーの最適化など、特定のプロセッサー機能を活用するコードを生成するようにコンパイラーに指示します。これらのフラグを使用することで、互換性のあるハードウェア上でアプリケーションのパフォーマンスを大幅に向上できます。

5.2.1.1 サポートされるターゲット・アーキテクチャー・フラグ

- `-march=graniterapids`
  インテル® Xeon® 6 (P コア搭載、コードネーム: Granite Rapids) 向けに最適化されたコードを生成し、サポートされているすべての命令セットを有効にします。

- `-xgraniterapids`
  `-march` と同様ですが、インテル固有の最適化を有効にする場合があります。コードがインテル®

Xeon® 6（P コア搭載）を搭載したインテル® ハードウェアでのみ実行される場合に使用します。

- `-mtune=graniterapids`
  Granite Rapids 向けにコードを最適化しますが、命令セットの使用を制限しません。インテル® Xeon® 6（P コア搭載）でも良好なパフォーマンスを発揮するポータブルなバイナリーを生成する際に有用です。

5.2.1.2 使用例

```
icx -O2 -march=graniterapids mycode.cpp

ifx -O2 -march=graniterapids mycode.f90
```

最適な結果を得るには、他の最適化フラグ（例: `-O3`、`-ipo`）と組み合わせて使用してください。

5.2.1.3 適切なフラグを選択: ポータビリティー、パフォーマンス、アクセシビリティー

- **最大限のポータビリティー**
  より低いアーキテクチャー・レベルを使用します（例: `-march=core-avx2`）。複数の CPU 間で効率良いバイナリーを作成するには、`-axCOMMON-AVX512,GRANITERAPIDS` などのマルチターゲット・フラグを使用します（注: これによりバイナリーサイズが増加します）。

- **最適なパフォーマンス**
  `-xgraniterapids` オプションを使用すると、インテル® Xeon® 6（P コア搭載）の機能を最大限に活用できます。あるいは、同じマシンでビルドと実行を行う場合は、`-xHost` を使用してください（Granite Rapids では、これは `-xgraniterapids` と同等です）。マルチターゲット・フラグは、複数のプラットフォーム間でのパフォーマンス向上が期待できますが、その代償としてバイナリーサイズが大きくなります。

- **アクセスのしやすさと利便性**
  ビルドマシンのアーキテクチャーに合わせて自動的に最適化するには、`-xHost` または `-march=native` を使用します。

  注: これらのフラグを使用してビルドされたバイナリーは、ビルドしたホストと同じアーキテクチャーを持つマシンでのみ、最適な動作とパフォーマンスが保証されます。異なる CPU 世代間でのポータビリティーはなく、インテル以外のプロセッサーでは動作しない可能性があります。

展開要件と、ポータビリティー、パフォーマンス、バイナリーサイズ間のトレードオフに基づいてフラグを選択してください。

5.2.2 サポートされている主要な命令セット拡張機能

拡張機能	コンパイラー・フラグ / 機能	説明 / 使用例
PREFETCHI0/1	`-mprefetchi`	コードのプリフェッチにより、レイテンシーを低減します。
AMX-FP16	`-mamx-fp16`	AI/ML 向けの半精度行列演算。

注: これらのフラグは通常、`-march=graniterapids` で自動的に有効になりますが、個別に細かく制御することもできます。

5.2.3 自動ベクトル化と AI 最適化

- インテル® コンパイラーは、Granite Rapids をターゲットとする場合、新しい命令セット（例: AMX-FP16）を使用してコードを自動的にベクトル化します。
- 最適化レポートを表示し、ベクトル化を確認するには、`-qopt-report=3 -qopt-report-phase[=vec]` を使用します。
- コンパイラーによる詳細なベクトル化のガイドは、ベクトル化ガイド（英語）で入手できます。

5.2.4 ベスト・プラクティス

- バイナリーは必ずターゲット・ハードウェア上でテストしてください。
- 「不正命令」エラーが発生した場合は、バイナリーが互換性のある CPU 上で実行されていることを確認してください。
- パフォーマンスをさらに向上させるには、プロファイル・ガイドによる最適化（`-prof-gen`、`-prof-use`）を検討してください。

5.2.5 参考文献

- インテル® oneAPI DPC++/C++ コンパイラー・ドキュメント（英語）
- インテル® Fortran コンパイラー・ドキュメント（英語）

5.3 インテル® MPI

インテル® MPI ライブラリーは、oneAPI HPC ツールキットに含まれています。

インテル® MPI ベンチマーク（英語）を使用して、システムの MPI パフォーマンスを確認してください。インテル® MPI ライブラリーには、パフォーマンスを調整するためチューニング・ユーティリティー（英語）もいくつか用意されています。アプリケーションの詳細な MPI 通信の統計は、インテル® APS（5.9 節）を使用して取得できます。

インテル® MPI バージョン 2021.14 以降には、P コアを搭載したインテル® Xeon® 6 向けのチューニングが含まれています。プラットフォームごとの特別なチューニングは MPI ライブラリーとは別のものであり、`tuning_gnr_shm.dat` や `tuning_gnr_shm-ofi.dat` などのチューニングファイルに格納されています。これらのファイルは以下の場所にあります: `$I_MPI_ROOT/opt/mpi/etc` これらのファイルは、インテル® Xeon® 6（P コア搭載）上でインテル® MPI が実行されていると自動的に読み込まれます。チューニングファイルでは、集合アルゴリズムや他の最適化設定のアルゴリズムを設定します。特定のクラスターに合わせてカスタマイズされた、専用のチューニングファイルを作成することもできます。

5.3.1 ノードのトポロジー

具体的なチューニングを行うには、クラスタノードの配置を把握することが重要です。これを理解するには、付属のアプリケーションである `cpuinfo` を使用します。ハイパースレッドを含む、論理コアと物理コアのマッピングが出力されます。これは、P コアを搭載したインテル® Xeon® 6 プロセッサーにおいて重要です。それは、これらのプロセッサーはコア数が異なる NUMA ノードを持つ可能性があるためです。新しい IPL2 ライブラリーはこの状況を自動的に処理しますが、正しいピニングを確認することを推奨します。

`cpuinfo` に加えて、`numactl -H` を実行して NUMA ノードを確認することもできます。NUMA ノード内部の通信は高速であるため、「PIN ドメイン」のスレッド処理は複数の NUMA ノードにまたがってはなりません。

5.3.2 ピニング情報

環境変数 `I_MPI_DEBUG=5` 以上に設定することで、ピニングに関連する情報を収集できます。これにより、MPI プログラムが起動されたときに、論理コアと MPI ランクのマッピングが出力されます。 特にハイブリッド MPI + OpenMP アプリケーションの場合、各ランクの「Pin ドメイン」が表示されます。これは、特定の MPI ランク上でスレッド処理に使用できる論理コアの範囲を示します。新しい IPL2 ライブラリーは、複数の NUMA ノードに関して最適なピンニングを試みます。デスクトップ・コンピューターの場合、P コアと E コアの違いを考慮します。

MPI と OpenMP の両方を使用するハイブリッド・アプリケーションでは、NUMA ノードに特に注意を払う必要があります。

```
export I_MPI_PIN_DOMAIN=numa
```

を使用すると、PIN ドメインが異なる NUMA 領域にまたがらないように強制できます。

5.3.3 暗黙的な MPI_スレッド_SPLIT プログラミング・モデル

環境変数 `MPI_THREAD_SPLIT` を設定すると、インテル® MPI で追加のスレッド機能が有効（英語）に

なります。これは MPI 規格の対象外であるため、デフォルトでは無効になっています。暗黙的モデルは、例えば、追加のスレッドにより帯域幅を増やすことで、MPI_ALLREDUCE のような集合演算を高速化できます。

これは OpenMP ランタイムでのみ機能します。暗黙的モデルを使用する場合、コードの変更は一切必要ありません。詳細については、こちら（英語）をご覧ください。

5.3.4 一方向通信

これは、一方向の MPI_Put および MPI_Get の拡張機能です。インテル® MPI では、これらの両方の機能をホストデバイスと GPU デバイスの両方で使用できます。一方向通信の普及が進まない理由の一つは、データがターゲットランクに到達したことを確認するため、送信元ランクで集団同期を行う必要があるためです。この新しい拡張機能は、MPI_Win_flush などの明示的な同期を必要とせずに、Put および Get 呼び出しの完了をより効率良く検出する方法を提供します。これは MPI 規格の範囲外であり、現時点ではインテル® MPI でのみ利用できます。

機能の詳細については、こちら（英語）を参照してください。

5.3.5 MPI アプリケーションのプロファイル

MPI アプリケーションを分析する出発点は、インテル® アプリケーション・プロファイル・スナップショット（インテル® APS）です（5.9 節を参照）。インテル® APS はインテル® VTune™ パッケージの一部ですが、スタンドアロンの実行ファイルとして単独で使用することもできます。インテル® APS は、トレースを残さずに累積結果を表示します。インテル® APS は軽量であるため、ランク数の多いアプリケーション（例えば 10 万ランク）を実行することが可能です。

5.3.5.1 インテル® APS を MPI アプリケーションで使用

インテル® APS は、MPI 実行ファイルの直前に ”aps” を挿入するだけで簡単に使用できます:

```
mpirun -np N aps <my_executable>
```

ほかに、I_MPI_GTOOL 環境変数を使用することもできます。 MPI プログラムが起動されるシェルやバッチファイルでこの環境変数を設定すると、コマンドラインを変更せずにツールを挿入できるようになります。

```
export I_MPI_GTOOL="aps -r<my_result_dir>:all"

# ここで MPI プログラムを実行
```

解析対象が MPI のみである場合、解析範囲を MPI または MPI + OpenMP に限定できます。これは、APS ハードウェア解析用のドライバーが存在しない場合、特に役立ちます。 これは、以下のオプションを追加することで実現できます:

```
--collection-mode=mpi[,omp]
```

APS の出力は、以下の方法で解析できます:

```
aps -report <my_result_dir>
```

上記を実行すると、ASCII 形式と HTML 形式の出力が生成されます。インテル® APS の詳細については、5.9 節を参照してください。

5.3.5.2 インテル® VTune™ を MPI アプリケーションで使用

gtool（英語）コマンドライン・フラグまたは I_MPI_GTOOL 環境変数を使用して、インテル® VTune™ やその他のプロファイル・ツールを起動することもできます:

```
export I_MPI_GTOOL="vtune -collect hpc-performance -r HPC:0"
```

上記の例では、ランク#0 でインテル® VTune™ を実行し、hpc-performance 解析を行います。この解析は、":0" の代わりに ":all" を使用することで、すべてのランクに対して実行できます。任意のランク番号を指定できます。

その他の例はこちら（英語）に掲載されています。インテル® VTune™ の詳細については、5.10 節を参照してください。

5.4 インテル® oneMKL

インテル® oneAPI マス・カーネル・ライブラリー は、インテル® oneAPI ベース・ツールキットに含まれていますが、こちらからもダウンロードできます。

5.4.1 一般サポート

インテル® oneAPI マス・カーネル・ライブラリー（oneMKL）は、インテル® プロセッサー（インテル® Xeon® 6 プロセッサー（P コア搭載）コード名 Granite Rapids を含む）向けに高度に最適化されています。パフォーマンス結果は、oneMKL のウェブページ（英語）で確認できます。

インテル® Xeon® 6 プロセッサー（P コア搭載）のサポートは、FP16 に AMX を活用できる AVX512_E5 コードパスとともに、2024.0 リリースで初めて導入されました。この機能は、環境変数

```
MKL_ENABLE_INSTRUCTIONS=AVX512_E5
```

を設定するか、

mkl_enable_instructions (MKL_ENABLE_AVX512_E5) API を呼び出すことで有効にできます。

oneMKL 2025.0 以降、AVX512_E5 コードパスは、該当する場合デフォルトで有効になります。

5.4.2 新機能

cblas_gemm_f16f16f32（英語）ルーチンは、スカラー-行列-行列積を計算し、その結果を一般行列のスカラー-行列積に加算します。

```
C:=α*op(A)*op(B)+β*C
```

ここで、A と B は FP16 エントリーを持つ行列であり、C は FP32 エントリーを持つ行列です。この API は、P コアを搭載したインテル® Xeon® 6 プロセッサーにおける FP16 の AMX サポートを活用できます。

ネイティブの半精度ハードウェア命令を持たないプロセッサーでは、行列 A と B は単精度にアップコンバートされ、SGEMM が呼び出されて行列乗算が計算されることに注意してください。

ディスパッチされたコードパスは、MKL_VERBOSE=1（英語）を指定して実行することで間接的に検証できます。以下の出力をご覧ください。

```
MKL_VERBOSE oneMKL 2025 Update 3 Product build 20251007 for Intel(R) 64
architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512)
with support for INT8, BF16, FP16 (limited) instructions, and Intel(R)
Advanced Matrix Extensions (Intel(R) AMX) with INT8, BF16, and FP16, Lnx
1.99GHz lp64 intel_thread

MKL_VERBOSE
GEMM_F16F16F32(N,N,8000,8000,8000,0x7ffed58c1b08,0xe38c13ed040,8000,0xe38b
99da 040,8000,0x7ffed58c1b04,0xe38aa5b5040,8000) …ms CNR:OFF Dyn:1
FastMM:1 TID:0 NThr:32
```

5.4.3 一般的な BKM

5.4.3.1 スレッド/CPU バインド

特にコア数の多いプロセッサーでは、パフォーマンス向上のため、スレッド/CPU バインドを設定することを推奨します。例えば、上記の例を Linux 上のインテル® Xeon® 6972P プロセッサーの最初の NUMA ノード（3.3.1 節）で実行する場合、以下を使用できます:

```
KMP_AFFINITY=compact,1,0 numactl -C 0-31 ./gemm_fp16 8000
```

5.4.3.2 数値再現性

一部のユースケース（例えば、金融サービス、データ分析、科学計算など）では、数値の再現性が求められる。これは、oneMKL を条件付き数値再現性（CNR）モードで実行することで有効にできます。MKL_VERBOSE=1 を設定すると、以下に示すように、CNR モードが有効になっていることと、使用されているコードパスを確認できます:

```
MKL_VERBOSE oneMKL 2025 Update 3 Product build 20251007 for Intel(R) 64
architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512)
with support for INT8, BF16, FP16 (limited) instructions, and Intel(R)
```

```
Advanced Matrix Extensions (Intel(R) AMX) with INT8, BF16, and FP16, Lnx
3.50GHz lp64 intel_thread

MKL_VERBOSE
SGEMM(N,N,8000,8000,8000,0x7ffe6532fda8,0x6acab1db040,8000,0x6ac9bdb6040,8
000, 0x7ffe6532fda4,0x6ac8c991040,8000) …ms CNR:AUTO Dyn:1 FastMM:1 TID:0
NThr:32
```

5.5 インテル® oneDNN（ディープ・ニューラル・ネットワーク・ライブラリー）

インテル® oneDNN は、ディープラーニングのプリミティブ向けに高度に最適化された CPU および GPU カーネルを提供し、PyTorch、TensorFlow、OpenVINO、およびカスタム oneAPI アプリケーションなどのフレームワークの基盤となるパフォーマンス・レイヤーです。インテル® Xeon® 6 プロセッサー（P コア搭載、コード名 Granite Rapids）では、oneDNN は新しいベクトルおよび行列機能を活用する主要な手段であり、AI 推論およびトレーニング・ワークロードにおいて、業界をリードするコアあたりのパフォーマンスと拡張性を提供します。

5.6 インテル® インテグレーテッド・パフォーマンス・プリミティブ（IPP）

インテル® インテグレーテッド・パフォーマンス・プリミティブ（IPP）は、画像処理、信号処理、データ圧縮、コンピューター・ビジョン、および数学演算向けに最適化された関数を提供するソフトウェア・ライブラリーです。このライブラリーには、SIMD 命令やベクトル処理ユニットなどのプロセッサー固有の機能を利用した高性能な実装が含まれます。インテル® IPP は、コードを変更することなく、異なる世代のインテル・プロセッサー間で動作する単一の API を提供します。

インテル® IPP は、ランタイム CPU ディスパッチを使用して、各プロセッサーに最適な関数実装を自動的に選択します。ライブラリーが起動すると、ディスパッチャーはプロセッサーの機能を検出し、それに合った最適化コードを選択します。インテル® Xeon® 6 プロセッサー（P コア搭載）では、IPP が AVX-512 などの命令セットを使用した最適化された関数を自動的に使用し、アプリケーションを再コンパイルすることなく最高のパフォーマンスを実現できます。さらに、IPP は OpenCV、LZ4、Zlib などのサードパーティー・ライブラリーの高速化にも利用できます。 パフォーマンス・ベンチマークの結果については、インテル® インテグレーテッド・パフォーマンス・プリミティブを参照してください。

5.7 インテル® クリプトグラフィー・プリミティブ・ライブラリー

インテル® クリプトグラフィー・プリミティブ・ライブラリーは、暗号化に特化したプログラミング向けの、安全で高速かつ軽量なビルディング・ブロックを提供するオープンソースのソフトウェア・ライブラリーです。このライブラリーは、フットプリントが小さく、高性能かつ安全なアプリケーションを開発できるように設計されており、クロスプラットフォーム API のサポート、FIPS 140-3 への準拠、および機密情報処理関数における定数時間実行を実現しています。

これには、ポスト量子時代に対応したセキュリティー対策、カーネルモードでの互換性、およびストレージ内と転送中のデータを保護するスレッドセーフな設計が含まれています。このライブラリーは、XMSS、LMS、ML-KEM、ML-DSA などのポスト量子暗号アルゴリズムをサポートしており、量子耐性のあるセキュリティー・ソリューションに対するニーズに対応しています。

またライブラリーは、プロセッサー固有の暗号化機能を自動的に検出して利用する、設定可能な CPU ディスパッチ機能を備えています。インテル® Xeon® 6 プロセッサー（P コア搭載）では、暗号化プリミティブ・ライブラリーは、IFMA（整数積和演算）、VAES（ベクトル AES）、PCLMUL（繰り上げなし乗算）などのハードウェア命令を高度に活用します。これらの命令により、暗号処理が大幅に高速化されます。具体的には、IFMA が RSA および ECC 向けの整数演算を高速化し、VAES がベクトル化された AES 処理を実現し、PCLMUL が多項式ベースの暗号関数を強化します。

詳細情報およびベンチマーク結果は、こちら（英語）をご覧ください。

5.8 インテル® Memory Latency Checker（MLC）

インテル® Memory Latency Checker（MLC）（英語）は、さまざまなシステムにおけるメモリーのレイテンシーと帯域幅の測定に使用されるスタンドアロン・ツールです。

MLC はシステム検証において重要なツールです。HPC アプリケーションを実行する前に、MLC を使用して、システムが期待されるメモリー帯域幅とレイテンシーを提供していることを確認する必要があります。完全なレポートを生成するには、引数なして MLC を実行します（例: `./mlc`）。よく使われるコマンドには以下があります（使用可能な引数については、`./mlc --help` をご覧ください）。

```
./mlc --peak_injection_bandwidth
./mlc --bandwidth_matrix
./mlc --latency_matrix
```

5.9 インテル® APS

インテル® アプリケーション・パフォーマンス・スナップショット（インテル® APS）は、アプリケーションのパフォーマンス・スナップショットを提供し、特に MPI 通信の非効率性を解析するのに役立ちます。

具体的には、以下の方法を用いて様々な MPI 統計を解析します:

```
export APS_STAT_LEVEL=5
aps -r <result_dir> <app command>     # アプリを aps で実行
aps-report -t <result_dir>            # MPI 時間を含むレポートを参照
aps-report -mDf <result_dir>          # メッセージサイズを含むレポートを参照
```

詳細情報および使用方法については、上記のリンクをご覧ください。インテル® MPI でインテル® APS を使用する方法については、5.3.5.1 節も参照してください。

5.10 インテル® VTune™ プロファイラー

インテル® VTune™ プロファイラーは、アプリケーションのパフォーマンス・ボトルネックのプロファイルおよび解析に使用されます。インテル® VTune™ プロファイラーの使い方に関する詳細については、以下の資料を参照してください:

- インテル® VTune™ プロファイラーのクックブック
- MPI アプリケーションをプロファイルするレシピまた、5.3.5.2 節もご覧ください。

5.10.1 インテル® Xeon® 6（P コア搭載）に関する詳細な分析

5.10.1.1 AMX_FP16 命令

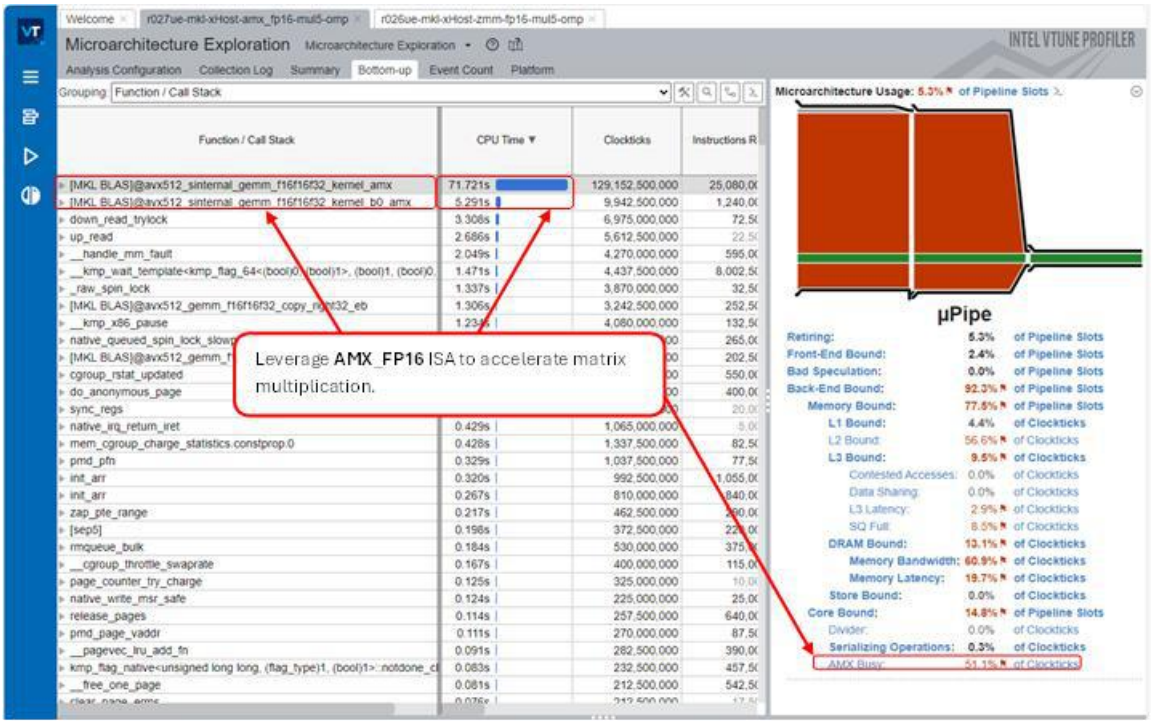
インテル® アドバンスト・マトリクス・エクステンション（インテル® AMX）は、より大きな 2 次元メモリーイメージからの部分配列を表す 2 次元レジスター（タイル）のセットと、タイルを操作できるアクセラレーターの 2 つのコンポーネントからなる新しい 64 ビット・プログラミング・パラダイムです。最初の実装は TMUL（Tile Matrix Multiply）ユニットと呼ばれています。

AMX_FP16 は、半精度浮動小数点数（FP16）のサポートを追加する AMX の拡張機能です。この拡張機能は、効率良い行列演算を必要とする人工知能（AI）やマシンラーニング（ML）のワークロードでは特に有益です。

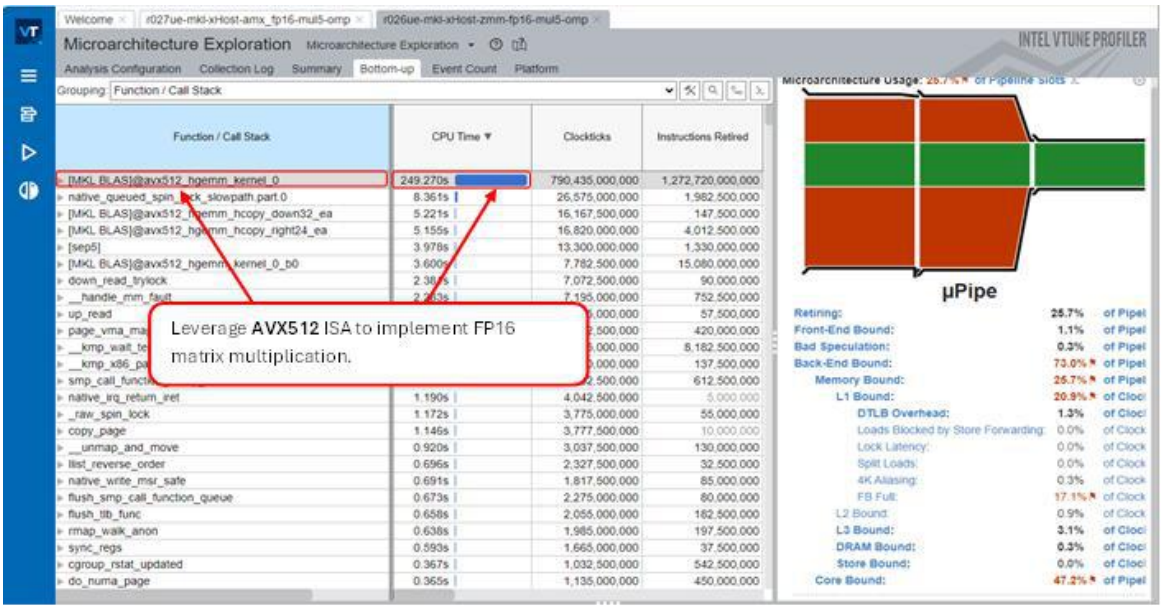
AMX_FP16 は FP16 データ型をサポートすることで、行列乗算のパフォーマンスを向上させます。これは、ディープ・ニューラル・ネットワークにおける学習と推論において非常に重要です。この拡張機能は、インテル® Xeon® 6 プロセッサー（P コア搭載）で初めて導入されました。

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タイプを使用すると、コードで AMX ISA が使用されているかどうか、またこれらの拡張機能にどれだけ CPU 時間が費やされているか識別できます。以下の比較結果から、同じワークロードであれば AMX_FP16 を使用することでパフォーマンスが得られることがわかります。

マイクロアーキテクチャー全般解析タイプを使用して AMX_FP16 の使用状況を特定する方法を示しています。



次の図は、マイクロアーキテクチャー全般解析タイプを使用して、AMX_FP16 以外の利用状況を識別する方法を示しています。



5.10.1.2 コード・プリフェッチ

PREFETCHIT0/1 命令セットは、インテル® Xeon® 6 プロセッサー（P コア搭載）Granite Rapids マイクロアーキテクチャーで導入されました。この命令セットは、IT0/IT1 ヒントを使用してコードを相対アドレスからプロセッサーに近い位置に移動させ、キャッシュミスを減らしてパフォーマンスを向上させます。

IT0（テンポラルコード）— キャッシュ階層のすべてのレベルにコードをプリフェッチします。

IT1（第 1 レベル・キャッシュ・ミスに関する時間コード）— キャッシュ階層の第 1 レベルを除くすべてのレベルにコードをプリフェッチします。

PREFETCHIT0/1 は、RIP 相対アドレスを使用した 64 ビットモードで使用できます。それ以外は、NOP のままです。最適なパフォーマンスを得るには、これらの命令で使用するアドレスは、実際の命令の先頭バイトである必要があります。

```
インテル® C/C++ コンパイラーの組み込み関数: (prfchiintrin.h)
void _m_prefetchit0 (const void* __P)
void _m_prefetchit1 (const void* __P)
```

次のパフォーマンス監視イベントは、PREFETCHIT1 命令によってパフォーマンスが向上する可能性のある命令アドレスを特定できます:

```
FRONTEND_RETIRED.L2_MISS
FRONTEND_RETIRED.UNKNOWN_BRANCH
FRONTEND_RETIRED.LATENCY_GE_128
```

5.10.1.3 分岐ヒント

インテル® Xeon® 6 プロセッサー（P コア搭載）から使用されている Redwood Cove マイクロアーキテクチャー以降、分岐予測器が条件分岐に関する情報を保存している場合、予測器はこの情報を使用して分岐を予測します。分岐予測器が分岐に関する情報を保存していない場合、分岐予測器は命令で提供されるインテル® SSE2 分岐実行ヒント（命令プレフィクス 3EH）を使用します。

ヒントは、予測器に分岐に関する情報が保存されていない場合にのみ使用されます。コンパイラーは、プロファイルに基づく最適化の一部としてヒントを追加することを推奨します。この場合、一方向の実行パスをフォールスルーとして配置することはできません。Redwood Cove マイクロアーキテクチャーでは、ヒントの配置をガイドする新しいパフォーマンス監視イベントが導入されています。

ヒントを慎重に追加することで、分岐予測ミスのペナルティーが減り、プログラムの実行時間を短縮できます。ヒントは、分岐の数が予測器の容量を超えるような大規模なコード・フットプリントのワークロードでは特に役立ちます。

ヒントは JCC にのみ適用され、CXZ および LOOP/LOOPCC には適用されません。少なくとも 1 つの命令プレフィクス 3EH が付いている命令には、分岐するヒントがあります。

次のパフォーマンス監視イベントは、ヒントを追加することで恩恵を受ける分岐を特定するのに使用できます:

- FRONTEND_RETIRED.MISP_ANT: 予測ミスによって実行されなかった、常に分岐しない（Always Not-Taken）条件付き分岐命令。このイベントの発生回数を監視することで、どの分岐命令が頻繁に予測ミスしているか特定できます。これらの分岐にヒントを追加することで、プロセッサーは分岐動作をより正確に予測できるようになり、予測ミスを減らすことができます。

- FRONTEND_RETIRED.ANY_ANT: 常に分岐しない（Always Not-Taken）条件付きの分岐命令がリタイアしました。インテル® VTune™ プロファイラーを使用してこのイベントの発生回数を解析することで、常に分岐しない分岐命令を特定できます。これにより、コードレイアウトを最適化し、不要な分岐予測を削減できます。

- BR_INST_RETIRED.COND: 条件分岐命令の実行回数をカウントします。

5.10.1.4 時間付き PEBS

時間付きプロセッサー・イベントベース・サンプリング（Timed PEBS）は、すべての PEBS レコードに時間を記録します。下記の表に示すように、PEBS レコードの基本情報グループに新しい “リタイアのレイテンシー（Retire Latency）” フィールドが追加され、すべての PEBS レコードに時間情報が追加されます。

オフセット	フィールド名	ビット
0x0	レコード形式	[31:0]
	リタイアのレイテンシー	[47:32]
	レコードサイズ	[63:48]
0x8	命令ポインター	[63:0]
0x10	適用可能なカウンター	[63:0]
0x18	TSC	[63:0]

「**リタイアのレイテンシー**」フィールドには、現在の命令のリタイア（PEBS レコードの「**命令ポインター**」フィールドで示される）から前の命令のリタイアまでの間の、ストールしていないコアサイクル数が報告されます。

MSR IA32_PERF_CAPABILITIES(0x345H).PEBS_TIMING_INFO[17] は、プロセッサーが時間付き PEBS 機能をサポートしていることを示しています。

TMA 5.0 は Granite Rapids 上で時間付き PEBS をサポートしており、時間付き PEBS 機能をサポートするプロセッサーでは、TMA メトリックの一部の計算式でリタイアのレイテンシー（retire_latency）イベント値が使用されるようになります。時間付き PEBS をサポートしていない以前の世代のプロセッサーでは、値は静的であり、ハードウェア設計者によってプロセッサー・ファミリーごとに事前に定義されます。実行環境におけるワークロードの多様性により、リアルタイムで測定されたリタイアメント・レイテンシー値の方がより正確です。したがって、時間付き PEBS を使用する新しい TMA メトリックは、さらに正確なパフォーマンス解析を提供します。

FRONTEND_RETIRED.LATE_SWPF イベントは、時間付き PEBS に基づいており、PREFETCHIT0/1 命令によってトリガーされる進行中の命令フェッチ・キャッシュ・ラインの背後で発生した命令キャッシュ要求ミスの回数をカウントします。

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タイプをカスタマイズしてイベントを含め、プロファイル結果でそのイベントを検索することで、2 のプリフェッチ命令によってトリガーされる進行中の命令フェッチ・キャッシュ・ラインの背後で命令キャッシュミスが発生している場所を特定できます。最後に、インテル® VTune™ プロファイラー GUI でソースコードを表示して、コードレイアウトを最適化できます。例えば、命令キャッシュミスを減らすため、PREFETCHIT0/1 命令の配置を調整できます。

時間付き PEBS に関する詳細はこちら（英語）をご覧ください。

5.11 インテル® パフォーマンス・カウンター・モニター（PCM）

インテル® パフォーマンス・カウンター・モニター（インテル® PCM）は、インテル® プロセッサーのパフォーマンスとエネルギーに関するメトリックを監視するアプリケーション・プログラミング・インターフェイス（API）と、API ベースのツールセットです。詳細はこちら（英語）をご覧ください。

5.12 インテル® PerfSpect

インテル® PerfScpect は、Linux サーバーとそこで動作するソフトウェアを解析・最適化するために設計されたコマンドライン・ツールです。ツールは、こちら（英語）からダウンロードできます。特に、以下のコマンドを使用して、現在のシステム構成の詳細なレポートを生成できます:

```
./perfspect report
```

6 業界標準ベンチマークのレシピ

6.1 目的

このセクションでは、基本的な業界標準ベンチマークのレシピを紹介します。これらのベンチマークは、他のアプリケーションを実行する前にシステムの初期性能を検証するために使用します（「スモークテスト」として）。

6.2 アプリケーション構築の共通 SPACK 環境

このセクションでは、ソースコードからアプリケーションをビルドする際に、このガイドで使用する oneAPI コンパイラーを使用して、共通の Spack 環境をセットアップする方法について説明します。Spack レシピを使用してアプリケーションをビルドする場合、ソースコードからビルドを開始する前に、**この環境をセットアップ**する必要があります。

Spack の導入ガイドのページは、こちら（英語）でご覧いただけます。

```
# GitHub から spack リポジトリー（最新リリースタグ）をクローン
git clone -c feature.manyFiles=true --depth=1 --branch=releases/latest
https://github.com/spack/spack.git

# spack 環境をソース
export SPACK_ROOT=$HOME/spack # point to your SPACK installation
source $SPACK_ROOT/share/spack/setup-env.sh

# Spack バージョンを確認
spack --version

# 最新のインテル® コンパイラー（現在はバージョン 2025.3.0）をダウンロードしてインストール
# 最新の oneAPI コンパイラーはこちら（英語）から入手できます。
spack install intel-oneapi-compilers@2025.3.0

# Spack にコンパイラーが追加されたか確認します。追加されたコンパイラーが表示されます。
# つまり、出力に「[+] intel-oneapi-compilers@2025.3.0」と表示されるはずです。
spack compiler list

# コンパイラーが追加されていない場合は、以下のコマンドを使用します：
# spack compiler find $(spack location -i intel-oneapi-compilers@2025.3.0)

#（必要に応じてシステム提供パッケージを追加します）
spack external find && spack external find -p /usr

# 完了しました。これでパッケージとアプリケーションのインストールに進むことができます。
```

6.3 MPI x OpenMP 解析の共通手順

特定のアプリケーションやワークロードに対する最適な MPI ランク数と OpenMP スレッド数は、プロセッサーの構成によって異なる場合があります。

特定の結果で使用された正確な MPI x OMP の分割方法については、パフォーマンス結果とともに公開されている構成の詳細を参照してください。明らかな分割（例えば、各コアで MPI ランクを実行するなど）が存在する場合、ドキュメントにそれを明記するように努めます。

6.4 STREAM

STREAM は、演算コアが消費するメモリー帯域幅を測定する業界標準ベンチマークです。

STREAM Triad の結果を MLC (5.4 節) を使用して検証することを推奨します。コマンドは以下のとおりです: `./mlc --peak_injection_bandwidth` は、「Stream-triad のような」ミックスを含む、いくつかのリード/ライトをミックスした帯域幅を表示します。

注: STREAM は、通常のストア（RFO ストア）または非テンポラル（NT）ストア（ストリーミング・ストアとも呼ばれる）のいずれかを使用するようにビルドできます。RFO ストアは、最新のインテル® Xeon® プロセッサー世代において、高いパフォーマンスを発揮する可能性があります。アプリケーションは両方のストア方式を使用する可能性があるため、メモリー帯域幅を測定するにはどちらのバージョンもテストすることを推奨します。一般的に、アプリケーションは RFO ストアを使用します。ただし、コンパイラーまたはユーザーが明示的に NT ストアを生成する場合は除きます。

注: STREAM ベンチマークでは、配列サイズを最終レベル（L3）キャッシュ（LLC）のサイズの少なくとも 4 倍にすることを推奨しています。最終レベルキャッシュのサイズは、`lscpu` ユーティリティーを使用して確認できます。しかし、L3 キャッシュの設計上、すべてのキャッシュを無効化することは不可能です。STREAM 配列のサイズを大きくすると、L3 キャッシュの容量が減少します。

6.4.1 ダウンロードの手順

STREAM のソースを `https://www.cs.virginia.edu/stream/FTP/Code/`（英語）からダウンロードします。

6.4.2 Spack のビルド手順

RFO と NT では、それぞれ異なる環境を用意することを推奨します。環境内ではストリーム実行ファイルは 1 つしか存在しないため、バイナリー間の競合は発生しません。

6.4.2.1 非テンポラルストア（NT）バ―ション

```
spack env create stream-nt spack env activate stream-nt
spack add stream cflags="-O3 -xCORE-AVX512 -qopt-zmm-usage=high -qopt-
```

```
streaming-stores=always -mcmodel=medium" +openmp
stream_array_size=600000000 ntimes=2000 stream_type=double
offset=0 %oneapi
spack install
ln -s $(spack location -i stream)/bin/stream_c.exe ./stream_bin

# 実行するには、6.4.3 節のコマンドラインを使用
```

6.4.2.2 通常のストア（RFO）バ―ション

```
spack env create stream-rfo spack env activate stream-rfo
spack add stream cflags="-O3 -xCORE-AVX512 -qopt-zmm-usage=high -qopt-
streaming-stores=never -fno-builtin -mcmodel=medium" +openmp
stream_array_size=600000000 ntimes=2000 stream_type=double
offset=0 %oneapi spack install
ln -s $(spack location -i stream)/bin/stream_c.exe ./stream_bin
# 実行するには、6.4.3 節のコマンドラインを使用
```

6.4.3 実行手順

以下のコマンドラインを使用して実行します:

```
OMP_NUM_THREADS=<NCORES>
OMP_PROC_BIND=spread
OMP_PLACES=threads
OMP_DISPLAY_AFFINITY=true
./stream_bin
```

ここで、NCORES はすべてのソケット（プロセッサー）上のコアの総数です。

注: 以下の代替コマンドラインも使用できます:

```
OMP_NUM_THREADS=<NCORES>
KMP_AFFINITY=verbose,granularity=fine,compact,1,0
./stream_bin
```

最適な結果を得るため、上記のコマンドラインではコアごとに 1 つのスレッドを実行します。

STREAM の出力には、各コンポーネント（Copy、Scale、Add、Triad）の「ベストレート」が表示され、この「ベストレート」がパフォーマンス・メトリックとして機能します。各コンポーネントの「平均時間」が「最小時間」に近いことを確認し、「ベストレート」が外れ値になっていないことを確認してください。

6.5 HPL（ハイパフォーマンス LINPACK）

インテル® Distribution for LINPACK ベンチマーク（英語）は、ハイパフォーマンス LINPACK（HPL）の修正と追加に基づいて作成されています。インテル® oneAPI マス・カーネル・ライブラリー

（oneMKL）のパブリック・ライブラリー・リリース（またはインテル® oneAPI ベース・ツールキットの一部）に、手順書（英語）とともに同梱されています。

6.5.1 ダウンロードの手順

HPL は、5.4 節で説明するインテル® oneAPI マス・カーネル・ライブラリー（oneMKL）で利用できます。

6.5.2 実行手順

一般的な実行手順はこちら（英語）から入手できます。

例えば、プロセッサー（ソケット）あたり 3 つの SNC ノードを持つ単一の 6900 シリーズシステムで実行するには、以下の手順を使用します。

- oneAPI 環境をソースします。以下に例を示します:

```
source /opt/intel/oneapi/setvars.sh
```

- oneMKL に同梱されている mp_linpack ディレクトリーのコピーを作成します:

```
cp -r <MKL_ROOT_DIR>/mkl/benchmarks/mp_linpack /dev/shm/
```

- .txt ファイルの名前を変更し、スクリプトを実行可能にします:

```
cd mp_linpack
mv runme_intel64_prv.txt runme_intel64_prv
mv runme_intel64_dynamic.txt runme_intel64_dynamic
chmod 755 runme_intel64_prv runme_intel64_dynamic
```

- runme_intel64_dynamic を編集し、以下の 3 行を変更します（シングルノードで実行する場合）:

```
export MPI_PROC_NUM=6          # 6700 シリーズのプロセッサーでは 4 を使用します
export MPI_PER_NODE=6          # 6700 シリーズのプロセッサーでは 4 を使用します
export NUMA_PER_MPI=1
```

- パラメーター “-p 3 -q 2 -b 384 -n N” を指定し、HPL.dat で定義されているデフォルト設定を上書きして、runme_intel64_dynamic を実行します。

    - p 倍 q は MPI プロセスの数と等しくなければなりません（6900 シリーズのシステム 1 台の場合は 6、6700 シリーズのシステム 1 台の場合は 4）。
    - b はブロックサイズ Nb を設定します
    - N は 120000 より大きい値で、理想的には N は p*q*Nb の倍数であるべきです。
    - N の値が大きいほど HPL 数は高くなりますが、実行時間も長くなります。理想的には、HPL 実行時に利用可能なメモリーの 70%～80% を消費するように上記のパラメーターを調整します（消費メモリー量を確認するには、4.3.3 節で説明される numastat を使用します）。

例: `./runme_intel64_dynamic -p 3 -q 2 -n 221184 -b 384`

HPL の出力には最終的な Gflops 値が表示され、これがパフォーマンス・メトリックになります。出力結果に “PASSED” と表示されていることを確認してください。

6.6 HPCG

6.6.1 ダウンロードの手順

インテル® Optimized HPCG ベンチマーク用のビルド済み実行ファイルは、5.4 節で説明されているインテル® oneAPI マス・カーネル・ライブラリー（oneMKL）に含まれています。

oneMKL の最上位インストール・ディレクトリーから、CPU 用の HPCG パッケージは、share/mkl/benchmarks/hpcg/hpcg_cpu にあります。パッケージには、readme.txt ファイルと、開始手順の説明が記載された QUICKSTART ガイドが含まれており、ビルド済みの実行ファイルは bin/ フォルダーにあります。

6.6.2 実行手順

このセクションでは、2 基のインテル® Xeon® 6979P プロセッサーを搭載した単一ノードで HPCG ベンチマークを実行する手順を説明します。以下の手順は、クラスター環境、他の CPU モデル、または異なる問題サイズでベンチマークを実行する場合にも簡単に利用できます。

例えば、2 つのソケットのインテル® Xeon® 6979P プロセッサーのノードには、それぞれ 120 の物理コアと 3 つの NUMA ドメインがあり、NUMA ドメインあたり 40 の物理コアに対応します。この例のノードは、ハイパースレッディングがオフ（HT OFF）の場合は 240 の論理コアを持ち、ハイパースレッディングがオン（HT ON）の場合は 480 の論理コアを持つことになります。この例では、ノード上で 12 個の MPI プロセスを実行することを目標としています。つまり、ソケットごとに 6 個の MPI プロセス、NUMA ドメインごとに 2 個の MPI プロセス、そして MPI プロセスごとに 20（HT OFF）/40（HT ON）個の OpenMP（OMP）スレッドを実行することになります。

最高のパフォーマンスを実現するため、各 MPI プロセスを NUMA ノードにバインドします。ベンチマー

クがインテル® MPI を使用して実行されると仮定すると、これは環境変数 I_MPI_PIN および I_MPI_PIN_DOMAIN を適切に設定することによって制御されます。MPI プロセスのピニング関する詳細については、こちら（英語）の記事を参照してください。

以下のスクリプトは、上記の設定に基づいて、ローカル問題サイズ n=384 でベンチマークを 100 秒間実行する方法を示しています（TOP500 への公式提出には少なくとも 1800 秒の実行時間が必要であることに注意してください）。HT は ON であると仮定します。この問題規模では、最適化されたベンチマークにおける最大メモリー使用量は約 692GB となり、物理コア 1 つあたり約 2.88 GB のメモリーが使用されることになります。

```
HPCG_BIN=${PATH_TO_HPCG_CPU_BIN} # oneMKL のパス
                                 # share/mkl/benchmarks/hpcg/
                                 # hpcg_cpu/bin フォルダーには、実行可能な
                                 # バイナリーファイルが格納されています

# HPCG GNR 実行パラメーター
nnodes=1           # ノード数
n=384              # ローカル HPCG ドメインの問題サイズ
runtime=100        # テストでの推奨は 100 秒程度、公式の実行では 1800 秒

ppn=12             # Xeon® 6900 シリーズにおけるノードあたりの MPI プロセス数
                   # (ノードあたり 2 ソケット、ソケットあたり 3 NUMA ドメイン、
                   # NUMA ドメインあたり 2 MPI プロセスを想定した場合)

# MPI プロセスあたりの OMP スレッド数
nthr=40             # HT ON
#nthr=20            # HT OFF

# MPI プロセスの総数
let nranks=${ppn}*${nnodes}

# 各 MPI ランクに従属する OMP スレッドの数を設定
export OMP_NUM_THREADS=${nthr}

# MPI プロセスを NUMA ドメインにバインド
export I_MPI_PIN=yes
export I_MPI_PIN_DOMAIN=numa

# 該当する場合は、MPI ファブリックを選択
export I_MPI_FABRICS=shm:ofi

# 事前済みベンチマークの AVX-512 バージョンを実行
mpiexec.hydra -genvall -n ${nranks} -ppn ${ppn}
${HPCG_BIN}/xhpcg_avx512 -n${n} -t${runtime} --run-real-ref=1
```

HPCG を実行すると、最後に “Final Summary” というテキストを含む出力ファイルが生成されます。最終的な GFLOP/s 値と実行時間が表示されます。また、最終的な GFLOP/s 値を標準出力に書き込みます。

7 HPC アプリケーションのレシピ

7.1 目的

このセクションでは、厳選された HPC アプリケーション向けのレシピについて説明します。

7.2 共通 SPACK 環境のビルド

アプリケーションを Spack を使用してソースからビルドする必要がある場合は、**6.2 節で説明されているように、共通の oneAPI Spack 環境を設定してください。**

7.3 Altair® RADIOSS®

7.3.1 ダウンロードの手順

Altair® RADIOSS®<バージョン> インストーラーは、ここから（英語）ダウンロードできます。ダウンロードには Altair One アカウントが必要です。

hwSolvers<バージョン>_linux64.bin インストーラー・バイナリー を入手したら、Altair RADIOSS を ALTAIR_PATH にインストールします。デフォルトの場所は ALTAIR_PATH=${HOME}/<バージョン> です。

```
/path/to/hwSolvers<version>_linux64.bin -i silent -
DUSER_INSTALL_DIR=${ALTAIR_PATH} -DACCEPT_EULA=YES
```

ベンチマークのモデルデータ:

- neon1m: リンク（英語）
- t10m: リンク（英語）

次に例を示します:

```
wget
https://openradioss.atlassian.net/wiki/download/attachments/47546369/
Neon1m11_2017.zip?api=v2 -O neon1m.zip
mkdir neon1m && pushd neon1m && unzip ../neon1m.zip && popd
```

7.3.2 実行手順

7.3.2.1 環境設定

```
# インテル® MPI などの最新のインテルのライブラリーが必要な場合は、oneAPI からソースコードを取得してください
source /opt/intel/oneapi/setvars.sh # Altair Radioss exports
export ALTAIR_LICENSE_PATH=<port@license_server>
export PATH=${ALTAIR_PATH}/altair/scripts/:$PATH
export ALTAIR_HOME=${ALTAIR_PATH}/altair
export
LD_LIBRARY_PATH=${ALTAIR_HOME}/hwsolvers/common/bin/linux64:${ALTAIR_HOME}
/hws
olvers/radioss/bin/linux64:${ALTAIR_HOME}/hwsolvers/radioss/lib/linux64:${
LD_L IBRARY_PATH}
export RADFLEX_PATH=${ALTAIR_HOME}/hwsolvers/common/bin/linux64
export RAD_CFG_PATH=${ALTAIR_HOME}/hwsolvers/radioss/cfg
export RADIOSS_PATH=${ALTAIR_HOME}/hwsolvers/radioss/bin/linux64 # インテル® MPI を使用した Radioss バージョン 2024.1 の例
export RADIOSS_STARTER_EXE=${RADIOSS_PATH}/s_2024.1_linux64
export RADIOSS_ENGINE_EXE=${RADIOSS_PATH}/e_2024.1_linux64_impi
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export I_MPI_HYDRA_BOOTSTRAP=ssh
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS="" # MPI SSH ブートストラップにおける Slurm の問題を回避
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PIN_ORDER=bunch
export I_MPI_PIN_CELL=core
export OMP_STACKSIZE=400M
export I_MPI_CBWR=1
export I_MPI_PIN_RESPECT_HCA=0
export I_MPI_HYDRA_TOPOLIB=ipl2
```

7.3.2.2 モデルの設定

```
# モデル: neon1m
export ALTAIR_STARTER_INPUT=NEON1M11_0000.rad
export ALTAIR_ENGINE_INPUT=NEON1M11_0001.rad # モデル: t10m
export ALTAIR_STARTER_INPUT=TAURUS_A05_FFB50_0000.rad
export ALTAIR_ENGINE_INPUT=TAURUS_A05_FFB50_0001.rad # メッシュを分割
pushd /path/to/model/
${RADIOSS_STARTER_EXE} -i ${ALTAIR_STARTER_INPUT} -np ${tasks_mpi}
```

7.3.2.3 モデルの実行

```
mpirun -bootstrap ssh -hosts-group ${SLURM_NODELIST} -np ${tasks_mpi} -ppn
```

```
${tasks per node} ${RADIOSS_ENGINE_EXE} -i ${ALTAIR_ENGINE_INPUT}
```

7.4 Ansys CFX

7.4.1 ダウンロードの手順

Ansys の全製品は、https://download.ansys.com （英語）からダウンロードできます。ダウンロードは登録済み/ライセンスユーザーのみが利用できるため、最初にアカウントを登録して認証する必要があります。

ダウンロード・ウェブサイトでは、最新リリース版と旧リリース版をダウンロードできます。必要なバージョンとプラットフォームを選択し、フルパッケージをダウンロードするか、選択した製品のみをダウンロードするか選択ができます。”ISO イメージ” ではなく、”プライマリー・パッケージ” をダウンロードすることをお勧めします。”プライマリー・パッケージ” は、Windows プラットフォームでは zip ファイル、Linux では圧縮された tar ファイルになります。 ファイルをディレクトリーに展開し、そのディレクトリーに移動して、INSTALL スクリプトを実行します。

```
./INSTALL -install_dir /path/to/your/desired/Ansys/location/ -silent
```

例えば、Ansys 2025R1 リリースでは、
/path/to/your/desired/Ansys/location/v251/CFX にインストールされます。
次期リリース 2025R2 ではバージョン番号が 252 となるため、CFX は次の場所にインストールされると予想されます:

```
/path/to/your/desired/Ansys/location/v252/CFX
```

すべてのベンチマークは、特定の CFX インストール環境で実行するため、適切に設定する必要があります。

7.4.2 実行手順

CFX を実行するには、ユーザーはライセンス情報を提供する必要があります。インストール時にライセンス情報を入力します。インストール時にこの手順を実行すれば、CFX の実行中に追加情報を提供する必要はありません。

インストール時にライセンス情報が入力されていない場合、ユーザーは環境変数 ANSYSLMD_LICENSE_FILE を通じてライセンス情報を提供する必要があります。正しいライセンスサーバーを port@machine_name の形式で指定していることを確認してください。

コアごとに MPI ランクを実行することをお勧めします。144 ランクの単一ノードで特定のワークロード（例:perf_LeMansCar_R16.def）を使用して CFX を実行するには、次のコマンドラインを使用します:

```
env LM_LICENSE_FILE=<port@machine> ANSYSLMD_LICENSE_FILE=<port@machine>
CFX5RSH=ssh <ANSYS_BASE_DIR>/v251/CFX/bin/cfx5solve -solver
```

```
<ANSYS_BASE_DIR>/v251/CFX/bin/linux-amd64/ifort/solver-mpi.exe -par-local
-part 144 -def <WKLD_DIR>/perf_LeMansCar_R16.def
```

144 ランクずつを持つ 4 つのノードで特定のワークロードを使用して CFX を実行するには、次のコマンドラインを使用します:

```
env LM_LICENSE_FILE=<port@machine> ANSYSLMD_LICENSE_FILE=<port@machine>
CFX5RSH=ssh <ANSYS_BASE_DIR>/v251/CFX/bin/cfx5solve -solver
<ANSYS_BASE_DIR>/v251/CFX/bin/linux-amd64/ifort/solver-mpi.exe -par-dist
node1*144,node2*144,node3*144,node4*144 -part 576 -def
<WKLD_DIR>/perf_LeMansCar_R16.def
```

ワークロードが大きい場合 (1000 万セル以上)、コマンドラインに **-part-large -large** を追加してください:

```
env LM_LICENSE_FILE=<port@machine> ANSYSLMD_LICENSE_FILE=<port@machine>
CFX5RSH=ssh <ANSYS_BASE_DIR>/v251/CFX/bin/cfx5solve -solver
<ANSYS_BASE_DIR>/v251/CFX/bin/linux-amd64/ifort/solver-mpi.exe -par-local
-part 144 -part-large -large -def <WKLD_DIR>/perf_Airfoil_50M_R16.def
```

パフォーマンスを確認するには、*.out ファイル (例: perf_LeMansCar_R16.out) 内のソルバーの実行時間を確認します:

```
$ grep “Solver w” *.out
```

7.5 Ansys Fluent

7.5.1 ダウンロードの手順

Ansys の全製品は、https://download.ansys.com (英語) からダウンロードできます。ダウンロードは登録済み/ライセンスユーザーのみが利用できるため、最初にアカウントを登録して認証する必要があります。

ダウンロード・ウェブサイトでは、最新リリース版と旧リリース版をダウンロードできます。必要なバージョンとプラットフォームを選択し、フルパッケージをダウンロードするか、選択した製品のみをダウンロードするか選択ができます。”ISO イメージ” ではなく、”プライマリー・パッケージ” をダウンロードすることをお勧めします。”プライマリー・パッケージ” は、Windows プラットフォームでは zip ファイル、Linux では圧縮された tar ファイルになります。 ファイルをディレクトリーに展開し、そのディレクトリーに移動して、INSTALL スクリプトを実行します。

```
./INSTALL -install_dir /path/to/your/desired/Ansys/location/ -silent
```

一般には、製品バージョン固有のディレクトリーを含む /opt/ansys にインストールすることをお勧めします:

例えば、Ansys 2025R1 リリースでは、**/opt/ansys/fluent251** にインストールできます。次期リリース 2025R2 ではバージョン番号が 252 となるため、Fluent は次の場所にインストールされると予想

されます: **/opt/ansys/fluent252**

特定の Fluent インストール環境でベンチマークを実行するには、すべてのベンチマークをインストール・ディレクトリー内で適切に設定する必要があります。

7.5.2 実行手順

Fluent を実行するには、ユーザーはライセンス情報を提供する必要があります。インストール中にライセンス情報を入力できます。インストール時にこの手順を実行すれば、CFX の実行中に追加情報を提供する必要はありません。

インストール時にライセンス情報が入力されていない場合、ユーザーは環境変数 ANSYSLMD_LICENSE_FILE を通じてライセンス情報を提供する必要があります。正しいライセンスサーバーを port@machine_name の形式で指定していることを確認してください。

input.jou で指定されたカスタマー・ワークロードと hostfile で指定されたノード情報を使用して、96 の MPI ランクでインテル・プラットフォームを使用して Fluent を実行するには、以下を使用します:

```
<FluentRoot>/bin/fluent 3d -g -ssh -i input.jou -t96 -mpi=intel
-platform=intel -cflush -cnf=hostfile
```

最適な結果を得るには、物理コアごとに 1 つの MPI ランクのみを使用することをお勧めします。ハイパースレッディングは使用しないでください。

インストール・ディレクトリーに適切な設定がされている場合、以下のコマンドラインで標準ベンチマークを実行できます。例えば、標準ベンチマーク sedan_4m を実行するには、以下を使用します:

```
<FluentRoot>/bin/fluentbench.pl sedan_4m -ssh -t96 -mpi=intel
-platform=intel -cflush -nosyslog -noloadchk -cnf=hostfile
```

正常に実行されると、<ケース名>-<ランク番号>.out という名前の結果ファイルが生成されます。以下の BASH スクリプトを使用して、主要なパフォーマンス・メトリックである対応するソルバー・レーティングを取得します。例えば、simpleOutFile=sedan_4m-96.out の場合、以下のコマンドを使用します:

```
resLine=$(grep "Solver rating" "$simpleOutFile") solverRating=$(echo
"$resLine" | awk '{print $4}') resLine=$(grep "Solver speed"
"$simpleOutFile") solverSpeed=$(echo "$resLine" | awk '{print $4}')
resLine=$(grep "Solver wall time per iteration" "$simpleOutFile")
solverTime=$(echo "$resLine" | awk '{print $7}')
```

7.6 Ansys Mechanical

7.6.1 ダウンロードの手順

Ansys の全製品は、https://download.ansys.com (英語) からダウンロードできます。ダウンロードは登録済み/ライセンスユーザーのみが利用できるため、最初にアカウントを登録して認証する必要が

あります。

ダウンロード・ウェブサイトでは、最新リリース版と旧リリース版をダウンロードできます。必要なバージョンとプラットフォームを選択し、フルパッケージをダウンロードするか、選択した製品のみをダウンロードするか選択ができます。”ISO イメージ” ではなく、”プライマリー・パッケージ” をダウンロードすることをお勧めします。”プライマリー・パッケージ” は、Windows プラットフォームでは zip ファイル、Linux では圧縮された tar ファイルになります。 ファイルをディレクトリーに展開し、そのディレクトリーに移動して、INSTALL スクリプトを実行します。

```
./INSTALL -install_dir /path/to/your/desired/Ansys/location/ -silent
```

7.6.2 実行手順

Mechanical を実行するには、ユーザーはライセンス情報を提供する必要があります。インストール中にライセンス情報を入力できます。インストール時にこの手順を実行すれば、CFX の実行中に追加情報を提供する必要はありません。

インストール時にライセンス情報が入力されていない場合、ユーザーは環境変数 ANSYSLMD_LICENSE_FILE を通じてライセンス情報を提供する必要があります。正しいライセンスサーバーを port@machine_name の形式で指定していることを確認してください。

Mechanical の場合、各バージョンには、特定のリリース番号に関連付けられた固有の番号が割り当てられています。例えば、2025R1 リリースでは 251 になります。

現在、標準ベンチマークを入手するには Ansys に問い合わせる必要があります。 例えば、標準ベンチマークまたはカスタマー・ワークロードを実行する場合、入力データが v25direct-3.data にあり、ユーザーが計算に 3 つのノードと各ノードに 32 の MPI ランクを割り当てた場合、次のコマンドラインを使用して Ansys Mechanical を実行できます:

```
<MechanicalRoot>/bin/ansys251 -b -perf on -dis -machines
nodeA:32:nodeB:32:nodeC:32 -nt 1 -cflush -mopt incore  -i
<WKLD_DIR>/v25direct-3.dat -o <OUT_DIR>/my96-3node-results.out >&
<OUT_DIR>/my96-3node-results.log
```

マシン仕様に関して、最初の MPI ランク（ランク#0）の負荷を軽減するため、最初のノード上の MPI ランクの数を減らすことができます。例えば、nodeA:10:nodeB:32:nodeC:32:nodeD:32 は、4 つのノードで合計 106 個の MPI ランクを使用します。

注: 一般的に、複数スレッドを使用することは推奨しません。そのため、上記のコマンドでは “-nt 1” というパラメーターを使用しています。

実行が正常に完了したら、以下の BASH スクリプトを使用して、主要なパフォーマンス・メトリックであるソルバー・レーティングを計算します:

```
resLine=$( grep "Elapsed time spent computing solution" <OUT_DIR>/my96-
3node-results.out )
solverTime=$( echo "$resLine" | awk '{print $7}' ) solverRating=$( echo
"scale=2; 86400 / $solverTime " | bc -l )
```

7.7 ヨーロピアン・オプションの二項価格評価モデル

7.7.1 ダウンロードの手順

インテル FSI（金融サービス業界）のサンプルは、すべて以下の場所にあります:
https://github.com/intel/Financial-Services-Workload-Samples/（英語）

7.7.2 ビルド手順

```
spack env create fsi-env
spack env activate fsi-env

# テストされたコンパイラーとライブラリー
spack install --add intel-oneapi-compilers@2025.3.0 intel-oneapi-
tbb@2022.3.0
spack load intel-oneapi-compilers intel-oneapi-tbb
git clone https://github.com/intel/Financial-Services-Workload-Samples
cd Financial-Services-Workload-Samples/BinomialOptions

# テストされた git リビジョン
git checkout 5591fe9
export __INTEL_POST_CFLAGS="-D__INTEL_COMPILER=1 -DVERBOSE=1 \
-qopt-report=max -Wno-vla-cxx-extension -fimf-precision=low \
-vecabi=cmdtarget -O3 -g -fiopenmp -std=c++20 -O3 -xCORE-AVX512 \
-qopt-zmm-usage=high -mrecip=all:0 -Xclang -target-feature \
-Xclang -fast-vector-fsqrt -Xclang -target-feature \
-Xclang -fast-scalar-fsqrt -ffinite-math-only -fno-honor-nans \
-ffp-contract=fast -ltbb -ltbbmalloc"

icpx binomial_cpu.cpp binomial_main.cpp -o binomial.avx512
```

7.7.3 実行手順

128 コア CPU（シングルソケット）で実行するには、以下を使用します:

```
cd MPtest
```

```
cp ../binomial.avx512 .; source ./clean.sh;
PR=128 source ./runbatch.sh binomial.avx512; wait
```

パフォーマンス・メトリック（経過時間（秒））を求めるには、以下を使用します:

```
PR=128 source ./getresults.sh res
```

7.8 ヨーロピアン・オプションの価格設定のためのブラック・ショールズ・モデル

7.8.1 ダウンロードの手順

インテル FSI（金融サービス業界）のサンプルは、すべて以下の場所にあります:

https://github.com/intel/Financial-Services-Workload-Samples/（英語）

7.8.2 ビルド手順

```
spack env create fsi-env
spack env activate fsi-env

# テストされたコンパイラーとライブラリー
spack install --add intel-oneapi-compilers@2025.3.0 intel-oneapi-
tbb@2022.3.0
spack load intel-oneapi-compilers intel-oneapi-tbb
git clone https://github.com/intel/Financial-Services-Workload-Samples
cd Financial-Services-Workload-Samples/BlackScholes

# テストされた git リビジョン
git checkout 5591fe9

# コンパイルエラーの修正
sed -i 's/#pragma vector/\/\/#pragma vector/g' BlackScholesDP.cpp
export __INTEL_POST_CFLAGS="-D__INTEL_COMPILER=1 -DVERBOSE=1 \
-qopt-report=max -Wno-vla-cxx-extension -fimf-precision=low \
-vecabi=cmdtarget -O3 -g -fiopenmp -std=c++20 -O3 -xCORE-AVX512 \
-qopt-zmm-usage=high -mrecip=all:0 -Xclang -target-feature \
-Xclang -fast-vector-fsqrt -Xclang -target-feature \
-Xclang -fast-scalar-fsqrt -ffinite-math-only -fno-honor-nans \
-ffp-contract=fast -ltbb -ltbbmalloc"

icpx BlackScholesDP.cpp -o BlackScholesDP.avx512
```

注: このベンチマークは、コア数が比較的少ないサーバー向けに 10 年以上前に作成されたものです。数百コアのマシンで実行するには、BlackScholes.cpp のオプション数をオリジナルの値 65536 から大幅に増やすことを推奨します。例えば、次のようにします:

```
const int OPT_N = 32*65536;
```

OPT_N の初期値を 65536 に設定すると、OpenMP のオーバーヘッドが有用な計算時間よりも長くなり、スレッドとコアへの割り当てが重要な要素となるため、通常、大規模なマシンではパフォーマンスに大きなばらつきが生じます。

7.8.3 実行手順

オリジナルバージョンのパフォーマンスを最大限に引き出すには、ソケットまたは NUMA ドメインごとに 1 つのインスタンスを実行します（ベンチマークを行う場合は、1 つのソケットまたは NUMA ドメイン上で 1 つのインスタンスを実行してください）。OMP_NUM_THREADS を適切に設定する必要があります。コアごとに 1 つのスレッドを実行することを推奨します。例えば、ソケットあたり 128 コアのマシンでは、以下のコマンドラインを使用します:

1 つのソケット（ソケット 0）で:

```
env OMP_NUM_THREADS=128 taskset -c 0-127 ./BlackScholesDP.avx512
```

ソケットあたり 42 コアの NUMA ノード（ノード 2）1 台の場合:

```
env OMP_NUM_THREADS=42 numactl --cpunodebind=2 --membind=2
./BlackScholesDP.avx512
```

複数のコマンドを使用することで、複数のソケット（または NUMA ノード）上で複数のコピーを同時に実行できます。

出力形式は次のとおりです:

```
Double Precision Black-Scholes Formula, version 1.4
Build Time        = Jun 18 2025 10:10:18
Input Dataset     = 65536
Repetitions       = 32000
Chunk Size        = 256
Worker Threads    = 112

===============================================
Time Elapsed      = 0.3258 sec
Throughput        = 12.8758 GOptions/sec
===============================================
```

上記の‘Throughput’の行は、パフォーマンスを示しています。

7.9 GROMACS

7.9.1 ダウンロードの手順

https://manual.gromacs.org/ （英語）にアクセスすると、必要なバージョンをダウンロードできます。一般には、対象となる年の最新のパッチリリース（通常は当年度のもの）を使用するべきです。しかし、GROMACS には優れた Spack レシピがあるため、このレシピを使用することを推奨します。

7.9.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

```
spack env create my-gromacs-env
spack env activate my-gromacs-env

spack install gromacs %oneapi ^intel-oneapi-mpi ^intel-oneapi-mkl
```

オプションとして、例えば @2025.x を使用して、特定のバージョンの GROMACS、コンパイラー、またはライブラリーを取得できます。

7.9.3 実行手順

```
spack load gromacs%oneapi
```

GROMACS の実際の使用例をダウンロードしてください。詳細は README ファイルを参照してください。

```
wget
https://zenodo.org/records/11234002/files/stmv_gmx_v2.tar.gz?download=1 -
O stmv_gmx_v2.tar.gz
tar xfz stmv_gmx_v2.tar.gz stmv_gmx_v2/pme_nvt.tpr
```

単一ノード上で以下のように実行します:

```
MPI_RANKS=$(nproc)
export OMP_NUM_THREADS=1
mpirun -np $MPI_RANKS gmx_mpi mdrun -notunepme -dlb yes -v -resethway -
noconfout -nsteps 10000 -s stmv_gmx_v2/pme_nvt.tpr
```

注: `-notunepme -dlb -resethway -noconfout -nsteps` は、`mdrun` パラメーターです。

`mdrun` パラメーターの詳細については、
https://manual.gromacs.org/documentation/current/user-guide/mdrun-performance.html (英語) を参照してください。

通常、最適な結果を得るには、コアごとに 1 つの MPI ランクを 1 つの OpenMP スレッドで実行します (つまり、`export OMP_NUM_THREADS=1`)。ただし、ワークロードが大きい場合、ハイパースレッドの論理コアを両方とも利用するため、コアごとに 2 つの OpenMP スレッドを使用する方が適しています (つまり、`export OMP_NUM_THREADS=2`)。

GROMACS は、最終的なパフォーマンス・メトリックである "Performance" (ns/日) を標準エラー出力と `md.log` 出力ファイルに出力します。

7.10 LAMMPS

7.10.1 ダウンロードの手順

入力ワークロードを取得するには、公式 GitHub (英語) または LAMMPS ウェブサイト (英語) から LAMMPS の**最新安定版リリース (ソースコード・パッケージ)** をダウンロードしてください。これには、`src/INTEL/TEST` ディレクトリー内のすべてのデフォルト・ワークロードが含まれます。

7.10.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします (6.2 節を参照)

```
# Spack 環境を作成してアクティブ化:

spack env create lammps spack env activate lammps

# spec にアプリケーションを追加:
# 最新の安定版 (stable_22Jul2025) を使用 (20250722)

spack add lammps@20250722 cflags='-xCORE-AVX512 -O3 -qopt-zmm-usage=high -
ffp-model=fast -ffast-math -freciprocal-math -qopenmp-simd -qopenmp'
cxxflags='-xCORE-AVX512 -O3 -qopt-zmm-usage=high -ffp-model=fast -ffast-
math -freciprocal-math -qopenmp-simd -qopenmp' ldflags='-lmkl_intel_ilp64
-lmkl_sequential -lmkl_core' +asphere+class2+dpd-
basic+intel+kspace+manybody+misc+molecule+openmp+opt+replica+rigid fft=mkl
fftw_precision=single %oneapi ^intel-oneapi-mkl ^intel-oneapi-mpi ^intel-
oneapi-tbb
```

```
# 現在の仕様で全てのアプリケーションをインストール（バージョンが非推奨の場合は --deprecated を追加）

spack install
```

7.10.3 実行手順

ほとんどの場合、最適な解決策は各コアに MPI ランクを使用することです（つまり、“純粋な MPI”）。ただし、ワークロードによっては、ハイパースレッディングによってパフォーマンスがわずかに向上する場合があります（すべての物理コアで MPI、2 つの OMP スレッド）。そのため、両方のオプションを確認してください。

1 つのノードで実行する場合、LAMMPS パッケージの src/INTEL/TEST にある rundir に移動してください。このディレクトリーには、すべてのデフォルト・ワークロードと、すべてのワークロードを 1 つのノードで実行する run_benchmarks.sh スクリプトが含まれています。

```
cd <github から LAMMPS へのパス>/src/INTEL/TEST

# バイナリーの名前を修正するため、初回実行前に一度以下の操作を行います
sed -i 's/lmp_intel_cpu_intelmpi/lmp_oneapi/g' run_benchmarks.sh

ln -s $(spack location -i lammps)/bin/lmp <github から LAMMPS へのパス>/src/lmp_oneapi

./run_benchmarks.sh 2>&1 | tee run.lod
```

run_benchmarks.sh は、すべてのデフォルト・ワークロードを実行することに注意してください。これにより、各ワークロードに対して 2 つの分割処理が実行されます:

1. OpenMP スレッドなしで、コアごとに 1 つの MPI ランク
2. 各コアに 1 つの MPI ランクと 2 つの OpenMP スレッド（ハイパースレッドを使用）

rundir ディレクトリー内に、ワークロードと分割ごとにログファイル（例: <NODENAME>_lmp_oneapi_<DATE>）が生成されます。

パフォーマンス・メトリック:

出力された .log ファイルには、”Performance:” で始まる 2 行と、ファイルの末尾に合計実行時間が表示されます。

```
$ grep Performance airebo.384c1t.log

Performance: 0.063 ns/day, 382.846 hours/ns, 1.451 timesteps/s, 163.692 Matom-step/s ### この行を無視
```

```
Performance: 0.052 ns/day, 461.823 hours/ns, 1.203 timesteps/s, 135.699
Matom-step/s ### これは実際のパフォーマンス行です
```

最初の行の “Performance:” はウォーミングアップの手順であるため無視してください。すべてのワークロードにおける主要なメトリックはタイムステップ/秒です（数値が高いほど良い）。

7.11 モンテカルロ法によるヨーロピアン・オプションの価格設定

7.11.1 ダウンロードの手順

インテル FSI のサンプルはすべて以下の場所で入手できます:

https://github.com/intel/Financial-Services-Workload-Samples/（英語）

7.11.2 ビルド手順

```
spack env create fsi-env
spack env activate fsi-env

# テストされたコンパイラーとライブラリー
spack install --add intel-oneapi-compilers@2025.3.0 intel-oneapi-
tbb@2022.3.0 spack install --add intel-oneapi-mkl@2024.2.2
spack load intel-oneapi-compilers intel-oneapi-tbb intel-oneapi-mkl
git clone https://github.com/intel/Financial-Services-Workload-Samples
cd Financial-Services-Workload-Samples/MonteCarloEuropeanOptions

# テストされた git バージョン
git checkout 5591fe9
export __INTEL_POST_CFLAGS="-D__INTEL_COMPILER=1 \
-qopt-report=max -Wno-vla-cxx-extension -fimf-precision=low \
-vecabi=cmdtarget -O3 -g -fiopenmp -std=c++20 -O3 -xCORE-AVX512 \
-qopt-zmm-usage=high -mrecip=all:0 -Xclang -target-feature \
-Xclang -fast-vector-fsqrt -Xclang -target-feature \
-Xclang -fast-scalar-fsqrt -ffinite-math-only -fno-honor-nans \
-ffp-contract=fast -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core \
-ltbb -ltbbmalloc"

icpx MonteCarloInsideBlockingDP.cpp -o MonteCarloInsideBlockingDP.avx512
```

7.11.3 実行手順

このテストは、シングルスレッドのインスタンスをバッチ処理で実行するように設計されています。次のコ

マンドシーケンスを使用します:

```
cd MPTest                    # このディレクトリーにはすべての .sh ファイルが含まれています
cp ../MonteCarloInsideBlockingDP.avx512 .
source ./clean.sh

# プロセス数（PR）はコア数と等しくなければなりません。
PR=256 source ./runbatch.sh MonteCarloInsideBlockingDP.avx512; wait
```

最後のコマンドは、インスタンスの平均処理時間を出力します。これは、パフォーマンスの逆メトリックです。

```
PR=256 source ./getresults.sh res
```

7.12 モンテカルロ法によるアメリカンオプションの価格設定

7.12.1 ダウンロードの手順

インテル FSI のサンプルはすべて、https://github.com/intel/Financial-Services-Workload-Samples/（英語）で入手できます。

7.12.2 ビルド手順

```
spack env create fsi-env
spack env activate fsi-env

# テストされたコンパイラーとライブラリー
spack install --add intel-oneapi-compilers@2025.3.0 intel-oneapi-tbb@2022.3.0
spack install --add intel-oneapi-mkl@2024.2.2
spack load intel-oneapi-compilers intel-oneapi-tbb intel-oneapi-mkl
git clone https://github.com/intel/Financial-Services-Workload-Samples
cd Financial-Services-Workload-Samples/MonteCarloEuropeanOptions

# テストされた git リビジョン
git checkout 5591fe9
export __INTEL_POST_CFLAGS="-D__INTEL_COMPILER=1 \
-qopt-report=max -Wno-vla-cxx-extension -fimf-precision=low \
-vecabi=cmdtarget -O3 -g -fiopenmp -std=c++20 -O3 -xCORE-AVX512 \
-qopt-zmm-usage=high -mrecip=all:0 -Xclang -target-feature \
-Xclang -fast-vector-fsqrt -Xclang -target-feature \
-Xclang -fast-scalar-fsqrt -ffinite-math-only -fno-honor-nans \
-ffp-contract=fast -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core \
-ltbb -ltbbmalloc"

icpx main.cpp utilities.cpp -o amc-avx512
```

7.12.3 実行手順

このテストは、シングルスレッドのインスタンスをバッチ処理で実行するように設計されていました。次のコマンドシーケンスを使用します:

```
cd MPTest
cp ../amc-avx512 .
source ./clean.sh
# プロセス数（PR）はコア数と等しくなければなりません
PR=256 source ./runbatch.sh amc-avx512; wait
```

最後のコマンドは、インスタンスの平均処理時間を出力します。これは、パフォーマンスの逆メトリックです。

```
PR=256 source ./getresults.sh res
```

7.13 MPAS-A

MPAS-A（Model for Prediction Across Scales – Atmosphere）は、米国大気研究センター（NCAR）とロスアラモス国立研究所が共同で開発した、最先端の非静力学全地球大気モデルです。MPAS-A は気象および気候研究の両方に対応するように設計されており、高解像度の地球規模および地域規模のシミュレーション向けの独自の機能を提供します。

公式ウェブサイト: https://www.mmm.ucar.edu/models/mpas（英語）
公式 GitHub リポジトリー: https://github.com/MPAS-Dev/MPAS-Model（英語）

7.13.1 ダウンロードの手順

MPAS-A バージョン 7 のベンチマーク・ケースをダウンロードするには、次のリンクを参照してください: https://www2.mmm.ucar.edu/projects/mpas/benchmark/v7.0/（英語）

例えば、benchmark_60km データセットをダウンロードするには、https://www2.mmm.ucar.edu/projects/mpas/benchmark/v7.0/MPAS-A_benchmark_60km_v7.0.tar.gz を使用します。

7.13.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

6.2 節でシステムにインストールされた正確な Spack バージョンを確認するには、次のコマンドを実行します:

```
spack --version
# 出力: 1.0.0.dev0 (cd3da0e6b7f4bc805777f5fb334e991e45a12262)
```

Spack を使用して、AVX2 最適化フラグとインテル® oneAPI コンパイラーを指定し、MPAS-A バージョン 7.3 をインストールします:

```
spack install --add mpas-model@7.3 \
fflags="-O3 -xCORE-AVX2" \
cflags="-O3 -xCORE-AVX2" \
%oneapi \
^netcdf-c@4.9.2+mpi+parallel-netcdf \
^netcdf-fortran@4.6.1 \
^parallel-netcdf@1.12.3+fortran \
^parallelio@2.6.2+pnetcdf \
^intel-oneapi-mpi
```

MPI を使用した並列実行の場合は、グリッド・パーティショナー METIS ライブラリーもインストールします:

```
spack install metis%oneapi
```

7.13.3 実行手順

以下の手順に従って実行してください。これにより、256 の MPI ランクが起動されます。MPAS-A を実行する際は、**物理コアごとに 1 つの MPI ランク**を使用することを推奨します。

```
#!/bin/bash

# Spack 経由で MPAS-A と Metis をロード
spack load mpas-model%oneapi
spack load metis%oneapi

# Spackb が管理する MPAS-A のインストール・ディレクトリーに移動
spack cd -i mpas-model

# UCAR から MPAS-A 60km ベンチマークのデータセットをダウンロード
wget https://www2.mmm.ucar.edu/projects/mpas/benchmark/v7.0/MPAS-
A_benchmark_60km_v7.0.tar.gz

# ベンチマークのデータセットを抽出
tar -xzvf MPAS-A_benchmark_60km_v7.0.tar.gz

# ベンチマーク・ディレクトリーに移動
cd MPAS-A_benchmark_60km_v7.0

# 現在のディレクトリーに MPAS-A 実行ファイルへのシンボリック・リンクを作成
ln -sf ../bin/atmosphere_model .

# MPI メモリーエラーを防ぐため、スタックサイズの制限を解除
```

```
ulimit -s unlimited

# 合計 MPI プロセス数 (1 ノード、インテル® Xeon® 6980P でフルノード利用の場合 256)
export NUM_MPI_RANKS=256

# MPI ランクあたりの OpenMP スレッド数 (ハイブリッド MPI+OpenMP 構成)
export OMP_NUM_THREADS=1

# 共有メモリー転送を使用 (シングルノード最適化)
# マルチノード InfiniBand の場合: export I_MPI_FABRICS=shm:ofi
export I_MPI_FABRICS=shm

# CPU ピンニングドメインを自動的に構成
export I_MPI_PIN_DOMAIN=auto

# グループ MPI は、高い局所性を実現するため、隣接するコアでランク付けされる
export I_MPI_PIN_ORDER=bunch

# OpenMP スレッドを親 MPI プロセスの近くにバインド
export OMP_PROC_BIND=close

# OpenMP スレッドを物理コアに配置する (SMT スレッドは配置しない)
export OMP_PLACES=cores

# OpenMP スレッドスタックサイズを 512MB に増やす
export KMP_STACKSIZE=512M

# Metis を使用してドメイン分割を生成
# x1.163842.graph メッシュを $NUM_MPI_RANKS 個のサブドメインに分割
gpmetis x1.163842.graph.info $NUM_MPI_RANKS

# 設定済みの環境変数をすべて使用して、Hydra MPI で MPAS-A を起動
mpiexec.hydra -genvall -n $NUM_MPI_RANKS ./atmosphere_model
```

シミュレーションが完了したら、次のコマンドを実行して、タイムステップごとの平均経過時間（秒単位）を取得します：

```
string=$(sed -n '/time integration/p' log.atmosphere.0000.out) && read -ra
arr <<< "$string" && echo ${arr[7]}
```

7.14 NAMD

7.14.1 ダウンロードの手順

NAMD のソースコードはこちら（英語）からダウンロードできます。Spack レシピについては、バージョン 3.0.1（2024-10-14）プラットフォームのソースコードを選択してください。

出力結果を解析してパフォーマンス・メトリックを取得するには、こちらのスクリプトを入手してください。

7.14.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

以下の Spack レシピを使用してください（必要に応じて、下記の差分パッチを適用します）:

```
spack env create myenv
spack env activate myenv
spack add namd@3.0.1 cflags='-qopt-zmm-usage=high -qopenmp-simd -
freciprocal-math' cxxflags='-qopt-zmm-usage=high -qopenmp-simd -
freciprocal-math' ldflags='-fuse-ld=lld' +avxtiles fftw=mkl
interface=tcl %oneapi ^charmpp ~fortran cxxflags='-O3 -xCORE-AVX512 -qopt-
zmm-usage=high' ldflags='-fuse-ld=lld' +omp+smp backend=mpi build-
target='charm++' %oneapi ^intel-oneapi-mpi
spack install namd
```

7.14.3 実行手順

ワークロードを実行するには、以下のスクリプトを使用してください。

```
spack load namd %oneapi

export I_MPI_FABRICS=shm
export I_MPI_PIN=off

DATE_STRING=`date +%s`
LOG_DIR="../RESULTS_"$DATE_STRING
mkdir $LOG_DIR

N_CORES=`grep processor /proc/cpuinfo | wc -l`

BINARY="namd3"

# ここで対応するワークロード名に変更します
WORKLOAD="stmv_npt_2fs"

# ノードごとに使用する MPI タスクの数
NPPN="32"
# ノードあたりの MPI タスク数に基づいて NAMD アフィニティーを設定
NAFFIN=`echo $N_CORES $NPPN | awk '{p=($1-$2)/$2; c=$1-1; f=p+1; print
"+ppn",p,"+commap ",0"-"c":"f,"+pemap 1-"c":"f"."p}'`

echo -n "Running $BINARY $WORKLOAD on $H_CORES with PPN=$NPPN..."
```

```
cmd="mpirun -ppn $NPPN -np $NPPN $BINARY $NAFFIN
$NAMD_BENCHMARK/$WORKLOAD/$WORKLOAD.namd"

echo -n "The command is $cmd"

echo $cmd >> $LOG_DIR/commands.log $cmd > $LOG_DIR/$WORKLOAD.log
$PYTHON ./ns_per_day.py $LOG_DIR/$WORKLOAD.log | awk
'$1=="Nanoseconds"{print $4}'

echo "Running $BINARY $WORKLOAD is done"
```

実行が完了したら、次のスクリプトを実行してパフォーマンス・メトリック（ns/日）を取得します: https://www.ks.uiuc.edu/~dhardy/scripts/ns_per_day.py（英語）

7.15 NEMO

NEMO 海洋モデル（Nucleus for European Modelling of the Ocean の略）は、海洋と気候システムとの相互作用をシミュレーションおよび研究するために設計された、最先端のオープンソース・モデリング・フレームワークです。欧州コンソーシアムによって開発された NEMO は、海洋学研究、運用予測、季節予測、気候研究などに幅広く利用されています。

公式ウェブサイト: https://www.nemo-ocean.eu（英語）
ユーザーガイド: https://sites.nemo-ocean.io/user-guide（英語）

7.15.1 ダウンロードの手順

このレシピでは、NEMO がまだ SPACK に含まれていないため、公開されている git リポジトリーから NEMO を手動でビルドする方法を説明します。NEMO を使用するには、システムに **XIOS**、**HDF5**、**NetCDF**（**C 言語**および **Fortran 言語**）がインストールされている必要があります。

以下は、インテル® oneAPI コンパイラーを使用して NEMO バージョン 4.2.2 をインストールおよび構成する手順ガイドです。

7.15.2 ビルド手順

最初に、システムに HDF5、NetCDF、XIOS、およびその他の必要なライブラリー（例:ZLIB）をインストールします:

```
#!/bin/bash

# I/O ライブラリーのターゲットパスを定義
export IO_LIBS_PATH=<ローカルシステム上のパス>
```

```
# 環境変数を設定
export CC=icx
export CXX=icpx
export FC=ifx
export F77=ifx
export F90=ifx
export MPICC=mpiicx
export MPIFC=mpiifx
export CFLAGS="-fPIC -O3 -xHost -ip -fno-alias -align"
export FFLAGS="-fPIC -O3 -xHost -ip -fno-alias -align"
export CXXFLAGS="-fPIC -O3 -xHost -ip -fno-alias -align"
export LDFLAGS="-L/usr/local/lib"
export CPPFLAGS="-I/usr/local/include"
export LD_LIBRARY_PATH=${IO_LIBS_PATH}/lib:$LD_LIBRARY_PATH

# ZLIB をインストール (既にインストールされている場合は不要)。
# 別のインストール・プリフィクスを使用する場合は、
# IO_LIBS_PATH を調整します
wget https://zlib.net/zlib-1.3.1.tar.gz
tar xf zlib-1.3.1.tar.gz
cd zlib-1.3.1
./configure --prefix=$IO_LIBS_PATH make
sudo make install cd ..

# HDF5 ライブラリーをインストール
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/hdf5-
1.14.6/src/hdf5-1.14.6.tar.gz
tar xf hdf5-1.14.6.tar.gz
cd hdf5-1.14.6
./configure --prefix=$IO_LIBS_PATH --enable-shared --enable-fortran --
enable-parallel --with-zlib=$IO_LIBS_PATH --with-pic
make clean
make -j4 && make install
cd ..

# NETCDF- C ライブラリーをインストール
wget https://github.com/Unidata/netcdf-c/archive/refs/tags/v4.9.2.tar.gz
tar xf v4.9.2.tar.gz
cd netcdf-c-4.9.2
export CFLAGS="-O2 -I${IO_LIBS_PATH}/include"
export CPPFLAGS="-O2 -I${IO_LIBS_PATH}/include"
export LDFLAGS="-L${IO_LIBS_PATH}/lib"
./configure --prefix=/usr/local --enable-netcdf-4 --enable-shared --
disable-dap --with-hdf5=$IO_LIBS_PATH
make clean
make -j4 && make install
cd ..
```

```
# NETCDF- Fortran ライブラリーをインストール
wget https://github.com/Unidata/netcdf-
fortran/archive/refs/tags/v4.6.1.tar.gz
tar xf v4.6.1.tar.gz
cd netcdf-fortran-4.6.1
export CPPFLAGS="-I${IO_LIBS_PATH}/include"
export CFLAGS="-I${IO_LIBS_PATH}/include"
export LDFLAGS="-L${IO_LIBS_PATH}/lib"
./configure CC=icx FC=ifx F77=ifx MPICC=mpiicx MPIFC=mpiifx \
  --prefix=$IO_LIBS_PATH \
  --enable-shared \
  LIBS="-lnetcdf -lhdf5_hl -lhdf5 -lz -lsz"
make clean
make -j4 && make install
cd ..
```

次に、XIOS ファイル I/O ライブラリーをインストールします:

```
# XIOS ライブラリーをクローン
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk XIOS
cd XIOS

# 以下の設定ファイルを makefile に追加
cat > arch/arch-ifx_local.fcm << EOF
%CCOMPILER           mpiicx
%FCOMPILER           mpiifx
%LINKER              mpiifx
%BASE_CFLAGS    -ansi -w -D_GLIBCXX_USE_CXX11_ABI=0 -std=c++11 -diag-
disable=10441
%PROD_CFLAGS    -O3 -fp-model source -no-fma -traceback -std=c++11 -
DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -traceback
%DEBUG_CFLAGS   -DBZ_DEBUG -g -traceback -fno-inline
%BASE_FFLAGS    -D__NONE__
%PROD_FFLAGS    -O3 -r8 -fp-model source -no-fma
%DEV_FFLAGS     -g -O2 -traceback
%DEBUG_FFLAGS   -g -traceback
%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++
%CPP            mpiicx -EP
%FPP            cpp -P
%MAKE           gmake EOF

# 以下の設定ファイルを makefile に追加:
cat > arch/arch-ifx_local.path << EOF
NETCDF_INCDIR="-I<your NETCDF directory>/include"
NETCDF_LIBDIR="-L<your NETCDF directory>/lib"
NETCDF_LIB="-lnetcdf -lnetcdff"
```

```
HDF5_INCDIR="-I<your HDF5 directory>/include"
HDF5_LIBDIR="-L<your HDF5 directory>/lib"
HDF5_LIB="-lhdf5_hl -lhdf5 -lhdf5 -lz -lcurl" EOF

ln -sf arch-ifx_local.path arch.path
ln -sf arch-ifx_local.env arch.env

# XIOS をビルド
./make_xios --full --prod --arch ifx_local -j8
```

最後に、NEMO をインストールします:

```
# NEMO コードをクローン
git clone --branch 4.2.2 https://forge.nemo-ocean.eu/nemo/nemo.git
nemo_4.2.2

cd nemo_4.2.2

# NEMO のすべてのコンパイラー・オプションは、/arch/arch-'my_arch'.fcm にあるファイルを使
用して制御されます。
# ここで、my_arch は、計算環境を参照するために使用する名称です。
# 以下の設定ファイルを makefile に追加:
cat > arch/arch-ifx_local.path << EOF
%NCDF_HOME            <your NETCDF directory>
%HDF5_HOME            <your HDF5 directory>
%XIOS_HOME            <your XIOS directory>
%OASIS_HOME           /not/defiled
%NCDF_INC             -I%NCDF_HOME/include
%NCDF_LIB             -L%NCDF_HOME/lib -lnetcdff -lnetcdf -L%HDF5_HOME/lib
-lhdf5_hl -lhdf5 -lhdf5
%XIOS_INC             -I%XIOS_HOME/inc
%XIOS_LIB             -L%XIOS_HOME/lib -lxios -lstdc++
%OASIS_INC            -I%OASIS_HOME/build/lib/mct -I
%OASIS_HOME/build/lib/psmile.MPI1
%OASIS_LIB            -L%OASIS_HOME/lib -lpsmile.MPI1 -lmct -lmpeu -lscrip
%CPP                  cpp
%FC                   mpiifx -c -cpp
%FCFLAGS              -i4 -r8 -O2 -fp-model precise -fno-alias -align
array64byte -xCORE-AVX512
%FFLAGS               %FCFLAGS
%LD                   mpif90 -f90=ifx %LDFLAGS
%FPPFLAGS             -P -traditional -std=c99
%AR                   ar
%ARFLAGS              rs
%MK                   gmake
%USER_INC             %XIOS_INC %NCDF_INC
%USER_LIB             %XIOS_LIB %NCDF_LIB
```

```
%CC                    mpiicx
%CFLAGS                -O0 EOF

# NEMO GYRE_PISCES テストケースをビルド
./makenemo -m linux_ifx_avx512 -r GYRE_PISCES -n MY_GYRE_PISCES clean
./makenemo -m linux_ifx_avx512 -r GYRE_PISCES -n MY_GYRE_PISCES -j 8


# NEMO BENCH テストケースをビルド
./makenemo -m linux_ifx_avx512 -a BENCH -n MY_BENCH clean
./makenemo -m linux_ifx_avx512 -a BENCH -n MY_BENCH
```

NEMO のビルドプロセスに関する詳細は、NEMO 公式ユーザーガイドを参照してください。

7.15.3 実行手順

OpenMP を使用せずに、**物理コアごとに 1 つの MPI ランク**で NEMO を実行することを推奨します。

NEMO BENCH ORCA1 テストケースを実行するには、NEMO のインストール・ディレクトリーに移動し、以下の手順を実行します:

```
#!/bin/bash

# NEMO ベンチマークの実験ディレクトリーに移動
cd nemo-4.2/tests/MY_BENCH/EXP00
# ORCA1 のような設定ネームリストをアクティブな namelist_cfg としてリンク
ln -sf namelist_cfg_orca1_like namelist_cfg

# メモリー割り当てのエラーを防ぐため、スタックサイズ制限を解除
ulimit -s unlimited

# MPI ランクの総数を設定
export NUM_MPI_RANKS=256

# MPI 通信には共有メモリー・ファブリックを使用 (シングルノード最適化)
export I_MPI_FABRICS=shm

# 最適な CPU ピンニングドメインを自動的に決定
export I_MPI_PIN_DOMAIN=auto

# グループ MPI は、高い局所性を実現するため、隣接するコアで一緒にランク付けします
export I_MPI_PIN_ORDER=bunch

# MPI を使用して NEMO シミュレーションを起動
mpiexec.hydra -genvall -n $NUM_MPI_RANKS ./nemo
```

nn_GYRE=15（GYRE 解像度 [1/度]）で NEMO GYRE_PISCES テストケースを実行するには、NEMO のインストール・ディレクトリーに移動し、以下の手順を実行します:

```
#!/bin/bash

# NEMO ベンチマークの実験ディレクトリーに移動
cd nemo-4.2/tests/MY_GYRE_PISCES/EXP00

# namelist_cfg ファイルで nn_GYRE=15（グリッド解像度）と jpkglo=101（垂直レベル数）を設定

# メモリー割り当てのエラーを防ぐため、スタックサイズ制限を解除
ulimit -s unlimited

# MPI ランクの総数を設定
export NUM_MPI_RANKS=256

# MPI 通信には共有メモリー・ファブリックを使用（シングルノード最適化）
export I_MPI_FABRICS=shm

# 最適な CPU ピンニングドメインを自動的に決定
export I_MPI_PIN_DOMAIN=auto

# グループ MPI は、高い局所性を実現するため、隣接するコアで一緒にランク付け
export I_MPI_PIN_ORDER=bunch

# MPI を使用して NEMO シミュレーションを起動
mpiexec.hydra -genvall -n $NUM_MPI_RANKS ./nemo
```

7.15.3.1 パフォーマンス・メトリックの抽出

ベンチマークはタイミングデータを timing.output ファイルに出力します。ベンチマーク・テストの完了後、実行ディレクトリー内で以下のコマンドを実行して、パフォーマンス・メトリック（タイムステップあたりの平均経過時間）を計算します:

```
grep ' timing step ' $1 | awk '{print $5}' | awk -f stats.awk
---
    items:      1000
      max:         0.505032
      min:         0.255904
      sum:       263.053349
     mean:         0.263053
 mean/max:         0.520864
```

stats.awk ファイルには以下が含まれています:

```
BEGIN{ a = 0.0 ; i = 0 ; max = -999999999  ; min = 9999999999 }
{
```

```
    i ++
    a += $1
    if ( $1 > max ) max = $1
    if ( $1 < min ) min = $1
}
END{ printf("---
\n%10s  %8d\n%10s  %15f\n%10s  %15f\n%10s  %15f\n%10s  %15f\n%10s  %15f\n"
,"it
ems:",i,"max:",max,"min:",min,"sum:",a,"mean:",a/(i*1.0),"mean/max:",(a/(i
*1.0 ))/max) }
```

7.16 NWChem

7.16.1 ダウンロードの手順

git clone https://github.com/nwchemgit/nwchem.git

7.16.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

以下の spack.yaml ファイルを作成します:

```
# spack.yaml の内容
spack:
   packages:
       all:
           providers:
               blas: [intel-oneapi-mkl]
               lapack: [intel-oneapi-mkl]
               scalapack: [intel-oneapi-mkl]
   specs:
       - nwchem+openmp fflags="-xCORE-AVX512 -qopt-zmm-usage=high"
armci=mpi-pr %oneapi ^intel-oneapi-mkl+cluster+ilp64 mpi_family=mpich
^intel-oneapi-mpi
   view: true
   concretizer:
       unify: true
```

環境を作成してインストールするには、以下のコマンド実行します:

```
spack env create nwchem-env
spack env activate nwchem-env
```

```
# (上記の) spack.yaml を $SPACK_ENV/spack.yaml に配置します
cp -pr spack.yaml $SPACK_ENV/spack.yaml
spack install
```

7.16.3 実行手順

次のスクリプトを実行します:

```
#!/bin/bash

# spack 環境をロード
spack env activate nwchem-env

# NWChem ビルドをロード
spack load nwchem armci=mpi-pr +openmp fflags="-xCORE-AVX512 -qopt-zmm-
usage=high" %oneapi ^intel-oneapi-mkl+cluster+ilp64 mpi_family=mpich
^intel-oneapi-mpi

# ワークロード C240 Buckyball の入力ファイルをダウンロード
wget https://nwchemgit.github.io/c240_631gs.nw

# OMP オプション
export OMP_NUM_THREADS=1

# 2 つのソケットとソケットあたり 96 コアを持つノードで NWChem のコマンドを実行
mpirun -n 192 -bootstrap ssh nwchem c240_631gs.nw |& tee run.log

# パフォーマンス・メトリックを出力
cat run.log | grep "Total times"
```

7.17 OpenFOAM

7.17.1 ダウンロードの手順

https://openfoam.com/ (OpenCFD Ltd.) からダウンロードしてください。

7.17.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします (6.2 節を参照)

Spack GitHub (英語) リポジトリーから常に最新の git を使用します。CGAL の依存関係にコンパイル上の問題があるため、"spack install" コマンドを実行する前に修正が必要です。CGAL 6.x バージョンのみが、oneAPI コンパイラーと問題なく連携します。環境にインストールするには、以下を使用し

ます:

```
# オプション: spack インストール・コマンドの "–verbose" オプションで
# より詳細なコンパイル出力を得るには、"args = ["-silent"]" を "args = []" に
# に置き換えてください。編集するには、spack edit openfoam を使用します
#
spack env create openfoam spack env activate openfoam
spack install --show-log-on-error --add openfoam%oneapi ^intel-oneapi-mpi
cxxflags="-O3 -mtune=native" ^cgal@6
```

7.17.3 実行手順

occDrivAerStaticMesh 65M の実行手順は以下のとおりです。

```
git clone https://develop.openfoam.com/committees/hpc.git
cd hpc/incompressible/simpleFoam/occDrivAerStaticMesh
wget
https://zenodo.org/records/15012221/files/polyMesh_65M.tar.gz?download=1 -
O polyMesh_65M.tar.gz
tar zxvf polyMesh_65M.tar.gz
mv polyMesh constant/

# "decompositionMethod hierarchical;" を "decompositionMethod scotch;" に変
更します
sed -i "/decompositionMethod/c\decompositionMethod scotch;"
system/include/caseDefinition
# "nCores 512;" を ”nCores N;" に変更します。ここで、N は使用する MPI ラ ンクの数です
sed -i "/nCores/c\nCores N;" system/include/caseDefinition
# "endTime 4000;" を "endTime 200;" に変更します
sed -i "/endTime/c\endTime 200;" system/controlDict.noWrite

spack load intel-oneapi-compilers
spack load openfoam

export I_MPI_PIN_PROCESSOR_LIST=allcores:map=spread

./Allrun
```

パフォーマンス・メトリックはクロックタイム、つまり 1 分あたりの反復回数です。

```
# 上記を実行すると、通常は時間比較に使用される ClockTime が得られます。
# また、1 分あたりの反復回数も計算されます。

clocktime=$(grep ClockTime logFiles/50_simpleFoam* | tail -1 |
awk '{print $(NF-1)}')
niter=$(grep ClockTime logFiles/50_simpleFoam* | wc -l)
niterpermin=$(echo "scale=2;${niter}/(${clocktime}/60)" | bc -q)
```

```
printf "OpenFOAM ClockTime  = %d seconds\n" $clocktime
printf "OpenFOAM iterations/minutes = %.2f\n" $niterpermin
```

7.18 QuantLib

7.18.1 ダウンロードの手順

定量金融向けの無料のオープンソース・ライブラリーである QuantLib は、以下のサイトで入手できます:

https://github.com/lballabio/quantlib （英語）

7.18.2 ビルド手順

```
spack env activate quantlib-env
# tested compiler and libraries
spack install --add intel-oneapi-compilers@2025.3.0
spack install --add boost@1.88
spack load intel-oneapi-compilers boost git clone
https://github.com/lballabio/quantlib mkdir quantlib/build
cd quantlib/build
cmake .. -G "Unix Makefiles" -D CMAKE_BUILD_TYPE=Release \
- DQL_ENABLE_PARALLEL_UNIT_TEST_RUNNER=1 -DCMAKE_CXX_COMPILER=icpx \
- DCMAKE_CXX_FLAGS="-O3 -ffast-math -D__FAST_MATH__=1 \
-fimf-usesvml=true:exp,pow,expl,erf -fimf-domain-exclusion=31 \
-fimf-precision=med -fnomath-errno -fno-finite-math-only -fhonor-nans \
-Wno-unused-command-lineargument -fno-associative-math \
-march=graniterapids"

make -j
```

7.18.3 実行手順

このテストは元々、シングルスレッドのインスタンスをバッチ処理で実行するように設計されていました。次のコマンドシーケンスを使用します:

```
cd test-suite

# --size で適切なサイズを選択し、--nProc で同時実行数を選択します。
./quantlib-benchmark --size=1 --nProc=16
```

最後のコマンドは、実行された概要とパフォーマンス・メトリックを出力します。

```
Benchmark Size        = Custom (1)
Number of processes   = 16
System Throughput     = 11.248 tasks/s
Benchmark Runtime     = 7.73473s
```

7.19 RELION

7.19.1 ダウンロードの手順

Spack レシピは現在有効ではないため、次のセクションで説明する git コマンドを使用してダウンロードおよびビルドしてください。

インテル® oneAPI コンパイラー、MKL、TBB、IPP、および MPI の最新バージョンが必要です。

RELION には、以下の Linux パッケージのインストールが必要です: `CMake`、`libtiff-dev*`、`libpng-dev*`、`libjpeg-dev*`、および `libxft-dev` (または `libXft-devel`)。

7.19.2 ビルド手順

oneAPI コンパイラーを使用したビルド手順:

```
git clone https://github.com/3dem/relion.git relion
cd relion; mkdir build; cd build
cmake \
 -DCMAKE_C_COMPILER=icx \
 -DCMAKE_CXX_COMPILER=icpx \
 -DMPI_C_COMPILER=mpiicx \
 -DMPI_CXX_COMPILER=mpiicpx \
 -DCUDA=OFF \
-DALTCPU=ON \
 -DMKLFFT=ON \
 -DGUI=OFF \
 -DFETCH_WEIGHTS=OFF \
-DCMAKE_BUILD_TYPE=Release \
-DCMAKE_C_FLAGS="-g -O3 -qopenmp-simd -xCORE-AVX512 -qopt-zmm-usage=high" \
 -DCMAKE_CXX_FLAGS="-DTBB_SUPPRESS_DEPRECATED_MESSAGES -
DTIFF_DISABLE_DEPRECATED -g -O3 -qopenmp-simd -xCORE-AVX512 -qopt-zmm-
usage=high" \
-DCMAKE_EXE_LINKER_FLAGS="-static-intel -static-libgcc -qopenmp-
link=static -Wno-unused-command-line-argument"  ..
make -j
```

TIFF、PNG、JPEG ライブラリーの場所を指定し、ファイルを含める必要がある場合もあります (例): `-DTIFF_LIBRARY=<location>`、`-DTIFF_LIBRARIES=<location>`、-

DTIFF_INCLUDE_DIR=<location>、および -DTIFF_INCLUDE_DIRS=<location>。

7.19.3 実行手順

RELION は MPI ベースのクライアント・サーバーモデルを採用しています。プライマリー MPI ランク（MPI ランク 0）は、ワーク項目を制御し、多数の計算クライアントに配布します。したがって、**奇数個の MPI ランクで実行する必要があります。**

使用するランク数と、ランクあたりのスレッド数は、ソケットあたりの実際のコア数によって決まります。インテル® ハイパースレッディング・テクノロジーを有効にした場合（約 20% の速度向上）、以下の設定を推奨します:

- システムの**ソケットあたり**のコア数が 64 以下の場合、ソケットあたり 8 つの MPI ランクを使用し、ソケットあたり約 96 のコア数では 12 の MPI ランクを使用し、1 ソケットあたり約 128 のコア数の場合は 16 の MPI ランクを使用します。

- MPI ランクあたり 14～32 スレッドを目安とし、（MPI ランク）x（MPI ランクあたりのスレッド数）の積が、システム内のハイパースレッドの総数以下になるように設定します。

実行時間が非常に長い、あるいはメモリー使用量が多い大規模な問題では、RELION を複数のノード（システム）で実行することを推奨します。理想的には、同じ規模のノード（システム）を使用すると良いでしょう。 この場合、上記で算出した MPI ランクの数に、使用するノード数を掛け合わせ、以下のスクリプトを適切なノード数と MPI ランクの総数に合わせて修正してください。

たとえば、次のスクリプトは、ソケットあたり 86 コア（86×2×2＝344 ハイパースレッド）のデュアルソケット・サーバー上で、**Plasmodium Ribosome 3D 分類**を 25 回反復実行します。

```
wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion_benchmark.tar.gz
tar zxf relion_benchmark.tar.gz
cd relion_benchmark
cat << EOF > run_3d.sh
#!/bin/bash


NUM_NODES=1     # シングルマシン
TOTAL_HYPERTHREADS_PER_NODE=344    # この 86 コア製品の場合 – 86*2*2
NUM_COMPUTE=24    # これは、ソケットあたり 86 コア、つまり 12 * 2 のデュアルソケット・サーバーの例です。
TOTAL_NUM_MPI_RANKS=$((NUM_COMPUTE+1))
NUM_THREADS=$((TOTAL_HYPERTHREADS_PER_NODE/NUM_COMPUTE))
NUM_ITERATIONS=25
POOLSIZE=4
RANKS_PER_HOST=$((TOTAL_NUM_MPI_RANKS/NUM_NODES))
APPEXE=/path/to/relion/build/bin/relion_refine_mpi
```

```
export I_MPI_PIN_DOMAIN=numa
export OMP_NUM_THREADS=$NUM_THREADS


STARTTIME=$(/bin/date +%s)
mpirun -n $TOTAL_NUM_MPI_RANKS -perhost $RANKS_PER_HOST $APPEXE --i
Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --
ini_high 60 --dont_combine_weights_via_disc --ctf --tau2_fudge 4 --
particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1
--healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale
--pad 2 --random_seed 0 --o ./3D --pool $POOLSIZE --j $NUM_THREADS --iter
$NUM_ITERATIONS --cpu
ENDTIME=$(/bin/date +%s)
secs=$((ENDTIME - STARTTIME))


rm -f 3D_it*


printf '3D Classification WALLTIME %d seconds (%d MPI %d iter)\n' $secs
$((TOTAL_NUM_MPI_RANKS-1)) $NUM_ITERATIONS
EOF
chmod 755 run_3d.sh
./run_3d.sh
```

上記のスクリプトは、最終的なパフォーマンス・メトリックである実行時間を秒単位で出力します。

以下は、**マラリア原虫リボソーム 2D 分類**を 25 回反復する手順です。

```
wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion_benchmark.tar.gz
tar zxf relion_benchmark.tar.gz
cd relion_benchmark
## 3D 分類用に既にダウンロード済みの場合、上記はスキップします。
cat << EOF > run_2d.sh
#!/bin/bash


NUM_NODES=1     # シングルマシン
TOTAL_HYPERTHREADS_PER_NODE=344    # この 86 コア製品の場合 – 86*2*2
NUM_COMPUTE=24    # これは、ソケットあたり 86 コア、つまり 12 * 2 のデュアルソケット・サーバーの例です。
TOTAL_NUM_MPI_RANKS=$((NUM_COMPUTE+1))
NUM_THREADS=$((TOTAL_HYPERTHREADS_PER_NODE/NUM_COMPUTE))
NUM_ITERATIONS=25
POOLSIZE=4
RANKS_PER_HOST=$((TOTAL_NUM_MPI_RANKS/NUM_NODES))
APPEXE=/path/to/relion/build/bin/relion_refine_mpi
```

```
export I_MPI_PIN_DOMAIN=numa
export OMP_NUM_THREADS=$NUM_THREADS


STARTTIME=$(/bin/date +%s)
mpirun -n $TOTAL_NUM_MPI_RANKS -perhost $RANKS_PER_HOST $APPEXE --i
Particles/shiny_2sets.star --dont_combine_weights_via_disc --ctf --
tau2_fudge 2 --particle_diameter 360 --K 200 --zero_mask --oversampling 1
--psi_step 6 --offset_range 5 --offset_step 2 --norm --scale --random_seed
0 --pad 2 --o ./2D --pool $POOLSIZE --j $NUM_THREADS --iter
$NUM_ITERATIONS --cpu
ENDTIME=$(/bin/date +%s)
secs=$((ENDTIME - STARTTIME))


rm -f 2D_it*


printf '2D Classification WALLTIME %d seconds (%d MPI %d iter)\n' $secs
$((TOTAL_NUM_MPI_RANKS-1)) $NUM_ITERATIONS
EOF
chmod 755 run_2d.sh
./run_2d.sh
```

上記のスクリプトは、最終的なパフォーマンス・メトリックである実行時間を秒単位で出力します。

7.20 SIMULIA PowerFLOW

7.20.1 ダウンロードの手順

ダッソーシステムズのすべてのソフトウェアは、https://software.3ds.com（英語）から入手できます。ダウンロードは、登録済みまたはライセンス済みのアカウントを持つユーザーに限定されているため、まずアカウントを作成し、認証プロセスを完了する必要があります。

ダウンロード・ポータルでは、最新のリリース版だけでなく、旧バージョンも提供しています。目的のバージョンとオペレーティングシステムを選択した後、インストール・パッケージ全体、または特定のコンポーネントのみをダウンロードできます。

ISO イメージではなく、“プライマリー・パッケージ” を使用することを推奨します。プライマリー・パッケージは、Windows 向けには ZIP アーカイブ、Linux 向けには圧縮 tar アーカイブで提供されます。アーカイブをフォルダーに展開し、そのフォルダーに移動して、以下に示す EXA_DIST 環境変数を設定し、インストールを実行して高速並列ストレージのセットアップを開始します。

```
EXA_DIST=/path/to/user/powerflow/6-2020-R5
```

一般的な推奨事項は、各製品リリースをそれぞれバージョン固有のサブディレクトリーに配置してインストールすることです。

7.20.2 実行手順

bin ディレクトリーと lib ディレクトリーへのパスを以下のように定義します:

```
PATH=${EXA_DIST}/bin:${EXA_DIST}/lib:$PATH
```

同様に、ライセンス・オプションとライセンスサーバーのパスを指定します:

```
RLM_LICENSE=port@machine-name
EXACORP_LICENSE=port@machine-name
RLM_LICENSE_FILE= port@machine-name
license_server_type=FLEXNET
```

計算処理が物理 CPU コアのみで実行されるようにし、ハイパースレッディング（HT）論理スレッドの使用を無効するには、次のオプションを有効にします:

```
OMP_DYNAMIC=off
```

インテル® MPI ディストリビューションは、環境変数を使用して明示的に上書きすることができ、以下の方法で MPI ランタイムの構成を正確に制御できます:

```
EXA_ALT_IMPI_ROOT
```

効率的なリソース利用を確保するため、システムレベルの処理向けに 1 つの CPU を確保しています。プライマリー（マスター）ノードでは、マスタースレッド専用のコアが 1 つ追加で割り当てられます。この制限がないと、マスタースレッドは**過剰な**計算エンティティーとなり、スレッドの移行、レイテンシーの増加、オーバーサブスクリプション（つまり、ノード上で物理的に利用可能な計算コア数より多くのコアを要求すること）といった望ましくない動作につながります。仮想コアに伴うパフォーマンス低下を避けるため、物理ハードウェア・コアのみを考慮します。ハイパースレッディングはコア数から明示的に除外されます。

ユーザーは入力モデルのワークロード・ファイル（*.cdi）を提供する必要があり、場合によっては *.lgi ファイルが必要となるかもしれません。

シミュレーションを実行するコマンドラインは次のとおりです:

```
$EXA_DIST/bin/exarun --simulate --nprocs $RUNCORES --mpirsh "ssh" --
mpifile hostfile --impi --mpioption "-prot -IBV -aff=automatic:bandwidth"
--num_timestamps 100000 --delay_vel_warnings 500000 $CASE
```

説明:

`--nprocs $RUNCORES` はすべてのシミュレーション・コアの数です、

`--impi` は `IMPI` バージョンです、

`--mpioption "-prot -IBV -aff=automatic:bandwidth"` は、InfiniBand IBV を指定します、

`--$CASE` はワークロードの入力モデルです。

シミュレーションがエラーや実行時の障害なく正常に完了すると、出力にはシミュレーション時間と合計実行時間（秒単位）が含まれます。

7.21 VASP

7.21.1 ダウンロードの手順

ユーザーはベンチマーク・ライセンスを取得し、最新のリリースバージョン（例: 6.4.3 節）のいずれかをダウンロードする必要があります。

ベンチマークは、https://github.com/NREL/ESIFHPC3/tree/master/VASP（英語）のサブディレクトリー `bench1/input` および `bench2/input` からダウンロードできます。

ベンチマークのすべてのバリアント（`-gga -gw -hse`）では、`INCAR-*` ファイルを `INCAR` に名前を変更する必要があります。例:

```
cp INCAR-gga INCAR
```

7.21.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

VASP のレシピが Spack の GitHub で入手できる場合は、Spack のレシピを使用してください。Spack のレシピが利用可能になるまでは、下記に説明する “代替レシピ” を使用できます。

`~/.spack/` に以下の内容の `packages.yaml` ファイルを作成します:

```
packages:
   intel-oneapi-mkl:
       externals:
       - spec: intel-oneapi-mkl@2025.1 +cluster +ilp64 mpi_family=mpich
           prefix: /opt/intel/oneapi/
       buildable: false
  intel-oneapi-mpi:
     externals:
     - spec: intel-oneapi-mpi@2021.15
         prefix: /opt/intel/oneapi/
     buildable: false
```

これで、Spack を使って VASP をインストールできます:

```
spack env create vasp
spack env activate vasp

spack add vasp@6.4.2 +openmp  %oneapi ^intel-oneapi-mkl ^intel-oneapi-mpi
spack install
```

Spack を使用しない代替レシピ（Spack レシピが入手できない場合のみ）

oneAPI 環境をソース:

```
source /opt/intel/oneapi/setvars.sh
```

VASP のソース・ディレクトリーに移動し、修正が必要なテンプレート `arch/makefile.include` をコピーします:

```
cd vasp-directory/
cp arch/makefile.include.intel_ompi_mkl_omp makefile.include
```

`makefile.include` の以下のパラメーターを変更します:

```
# コンパイラーとリンカーのフラグ
FC          = mpiifx -mprefer-vector-width=512 -xCORE-AVX512 -fiopenmp
FCL         = mpiifx -mprefer-vector-width=512 -xCORE-AVX512 -qmkl -
fiopenmp
# 最適化
OFLAG       = -O3 -g -traceback
# MKL フラグ
MKL_PATH    = $(MKLROOT)/lib/intel64
BLAS        =
LAPACK      =
BLACS       = -lmkl_blacs_intelmpi_lp64
SCALAPACK   = -lmkl_scalapack_lp64 $(BLACS)
```

```
OBJECTS    = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o
OBJECTS_O1 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_O2 += fft3dlib.o

INCS       =-I$(MKLROOT)/include/fftw
LLIBS      = $(SCALAPACK) $(LAPACK) $(BLAS)
OBJECTS_LIB = linpack_double.o

# パーサー・ライブラリーの場合
CXX_PARS    = icpx LLIBS       = -lstdc++

# これらの行をコメントアウト
# SCALAPACK_ROOT = ...
```

VASP の ”std” バ―ションをビルド

```
make DEPS=1 -j std
```

VASP ビルドをクリーンアップするには、以下を使用:

```
make veryclean
```

7.21.3 実行手順

VASP のビルド方法に応じて、OneAPI 環境をロードするか、Spack 環境をアクティブ化します:

```
spack env activate vasp         # Spack でインストールした場合
spack load vasp %oneapi         # Spack なしでビルドした場合
```

VASP 入力ファイルのあるディレクトリーに移動します:

```
INCAR
KPOINTS
POSCAR
POTCAR
```

環境変数を設定します:

```
export OMP_PROC_BIND=close
export OMP_PLACES=threads
export OMP_PROC_BIND=true
export I_MPI_FABRICS=shm:ofi
export I_MPI_DEBUG=5
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PIN_ORDER=bunch
export I_MPI_PIN_CELL=unit
```

```
# OMP スレッド数と MPI ランク数を設定します。以下に例を示します。
# 必要に応じてランク数とスレッド数を調整してください。

export OMP_NUM_THREADS=4
mpirun -n 8 /path/to/vasp_std
```

出力メトリックを取得するには、以下を使用します:

```
# 出力ファイル OUTCAR を解析して DFT ループ実行時間を取得
#
time=$(grep 'LOOP+' OUTCAR | awk '{print $7}')
```

最高のパフォーマンスは通常、ハイパースレッディングによって実現されます。例として、以下の MPI x OMP 分解を使用します:

- 128 コアの 6900 シリーズマシンでは、MPIxOMP=24x20 となります。
- 96 コアの 6900 シリーズマシンでは、MPIxOMP=24x16 となります。

7.22 WRF

気象研究予測モデル（WRF）は、米国で主に NCAR と NOAA によって開発された、高度なオープンソースの数値気象予測（NWP）システムであり、大気研究と運用予測の両方に使用されます。WRF は非常に柔軟性が高く、数メートルから数千キロメートルまでのスケールで気象現象をシミュレートできるため、幅広い気象用途に適しています。

公式ウェブサイト: https://www.mmm.ucar.edu/models/wrf （英語）
GitHub リポジトリー: https://github.com/wrf-model/WRF （英語）

7.22.1 ダウンロードの手順

WRF 4.4 ベンチマーク・ケースをダウンロードするには、次のリンクを参照してください:
https://www2.mmm.ucar.edu/wrf/users/benchmark/ （英語）

例えば、WRF v4.4 以降用の CONUS-2.5km データセットは、次のリンクからダウンロードできます:
https://www2.mmm.ucar.edu/wrf/users/benchmark/v44/v4.4_bench_conus2.5km.tar.gzm.tar.gz （英語）

7.22.2 Spack のビルド手順

要件: 共通の SPACK 環境をセットアップします（6.2 節を参照）

Spack を使用して、AVX-512 最適化フラグとインテル® oneAPI コンパイラーを使用する WRF バージョン 4.5.2 をインストールします:

```
spack install --add wrf@4.5.2 fflags="-xCORE-AVX512 -qopt-zmm-usage=high"
cflags="-xCORE-AVX512 -qopt-zmm-usage=high" build_type=dm+sm %oneapi
^intel-oneapi-mpi
```

7.22.3 実行手順

```
#!/bin/bash

# -------------------------------
# WRF をロードしてベンチマークを準備
# -------------------------------

# Spack パッケージ・マネージャーを使用して WRF モジュールをロード
spack load wrf%oneapi

# Spack が管理する WRF インストール・ディレクトリーに移動
spack cd -i wrf

# UCARv から公式 vCONUS-2.5kmv ベンチマーク・データセットをダウンロード
wget
https://www2.mmm.ucar.edu/wrf/users/benchmark/v44/v4.4_bench_conus2.5km.ta
r.gz
```

```
# ダウンロードしたベンチマーク・データセットのアーカイブを展開
tar xzvf v4.4_bench_conus2.5km.tar.gz


# 実データのシミュレーションを行うため、WRFb テストケース・ディレクトリーに移動
cd test/em_real

# 競合を避けるため、既存の namelist.input をすべて削除
rm namelist.input

# 現在のディレクトリーにベンチマーク入力ファイルへのシンボリック・リンクを作成
ln -sf ../../v4.4_bench_conus2.5km/* .

# ベンチマーク実行のメインの namelist.input として、再起動用ネームリストをリンク
ln -sf ../../v4.4_bench_conus2.5km/namelist.input.restart namelist.input

# 境界条件ファイルをリンク
ln -sf ../../v4.4_bench_conus2.5km/wrfbdy_d01 .

# WRF 再起動ファイルをリンクし、想定される入力ファイル名と一致するように名前を変更
ln -sf ../../v4.4_bench_conus2.5km/wrfrst_d01_2019-11-26_23:00:00.ifort
wrfrst_d01_2019-11-26_23:00:00

# ------------------------------
# WRF を実行する環境設定
# ------------------------------

# メモリー割り当てエラーを防ぐため、スタックサイズの制限を解除
ulimit -s unlimited

# 通信を共有メモリーのみに制限する（シングルノードでの実行向けに最適化）。InfiniBand 上でマル
チノードを実行する場合は、I_MPI_FABRICS=shm:ofi を使用できます。
export I_MPI_FABRICS=shm

# 最適な CPU ピンニングドメインを自動的に決定
export I_MPI_PIN_DOMAIN=auto

# グループ MPI は、高い局所性を実現するため、隣接するコアで一緒にランク付け
export I_MPI_PIN_ORDER=bunch

# MPI プロセスの総数
export NUM_MPI_RANKS=128

# MPI ランクあたりの OpenMP スレッド数（ハイブリッド MPI+OpenMP 構成）
export OMP_NUM_THREADS=2

# OpenMP スレッドを親 MPI プロセスの近くにバインド
```

```
export OMP_PROC_BIND=close

# OpenMP スレッドを物理コアに配置する（SMT スレッドは配置しない）
export OMP_PLACES=cores

#  OpenMP スレッドスタックサイズを 512MB に増やす
export KMP_STACKSIZE=512M

# WRF の内部タイル分解を設定
export WRF_NUM_TILES=32

# すべての環境変数を使用して Hydra MPI で WRF を起動
mpiexec.hydra -genvall -n $NUM_MPI_RANKS ./wrf.exe
```

最適な MPI x OpenMP の分解とタイル数（WRF_NUM_TILES）は、WRF テストケース、CPU アーキテクチャー、相互接続、およびノード数の構成によって異なります。例えば、CONUS-2.5km データセットは、128 の MPI ランク、2 つの OpenMP スレッド、および WRF_NUM_TILES=32 を使用した場合、シングルノードのインテル® Xeon® 6980P プロセッサー（128 コア/ソケット）で最高のパフォーマンスを発揮します。具体的な分解の詳細については、パフォーマンス結果とともに公開されている構成の詳細を参照してください。

7.22.3.1 パフォーマンス・メトリックの抽出

ベンチマークは、rsl.out.0000 または rsl.error.0000 ログファイルにタイミングデータを出力します。パフォーマンス・メトリック（タイムステップあたりの平均経過時間）は、テスト実行完了後に実行ディレクトリー内で以下のコマンドを実行して計算できます:

```
grep 'Timing for main' rsl.out.0000 | tail -239 | awk '{print $9}' |
awk -f stats.awk
---
    items:       239
      max:         4.956210
      min:         0.943830
      sum:       247.612110
     mean:         1.036034
 mean/max:         0.209038
```

上記は、15 秒の時間ステップで CONUS 2.5km を 1 時間のシミュレーションを行った平均時間（秒単位）を測定する代表的な出力例も示しています。タイムステップ数は、シミュレーションの総時間をタイムステップ数で割って算出されることに注意してください。

この場合、シミュレーションの総時間は 1 時間で、時間ステップは 15 秒であるため、3,600/15 = 240 ステップになります。このコマンドでは、平均計算の最初のタイムステップを無視します。これは、最初のタイムステップは通常、セットアップに時間がかかり平均計算に時間を要するためです。

stats.awk ファイルには以下の内容が含まれています:

```
BEGIN{ a = 0.0 ; i = 0 ; max = -999999999  ; min = 9999999999 }
{
    i ++
    a += $1
    if ( $1 > max ) max = $1
    if ( $1 < min ) min = $1
}

END{ printf("---
\n%10s %8d\n%10s %15f\n%10s %15f\n%10s %15f\n%10s %15f\n%10s %15f\n","it
ems:",i,"max:",max,"min:",min,"sum:",a,"mean:",a/(i*1.0),"mean/max:",(a/(i
*1.0 ))/max) }
```