OpenMP* 誕生から 20 年

並列プログラミングを C/C++ および Fortran プログラマーが利用しやすいものにし エクサスケール計算への道をソフトウェアに提供

Rob Farber TechEnablement グローバル・テクノロジー・コンサルタント

OpenMP* Architecture Review Board (ARB) (英語) は、1997 年 10 月に OpenMP* Fortran 仕様 1.0 を、そしてその約 1 年後に C/C++ 仕様を発表しました。当時、世界最速のスーパーコンピューター **ASCI Red** (英語) は、2 基のインテル® Pentium® Pro プロセッサー (200MHz) を搭載した計算ノードで構成されていました。(今では信じられないことですが、当時は最先端のスーパーコンピューティングで、シングルコアの 200MHz のインテル® Pentium® プロセッサーが最速と考えられていました。) 4600 万ドル (現在の約 6800 万ドルに相当) を投じて構築された ASCI Red は、TOP500 LINPACK ベンチマークで TFLOPS (1 秒間に 1 兆回の浮動小数点演算を実行) を達成した最初のスーパーコンピューターでした。また、その後の傾向の前兆とも言える、メガワットの電力を消費した最初のスーパーコンピューターでもあります。それと比べると、最近のデュアルソケットのインテル® Xeon® プロセッサー v4 ファミリーは、TFLOPS の計算性能と電力消費の両方において格安と言えます。

Center for Research in Extreme Scale Technologies (英語) ディレクターの Thomas Sterling 博士は、次のように述べています。「OpenMP* により、単純さと統一性を強調した単一システムのプログラミングと実行

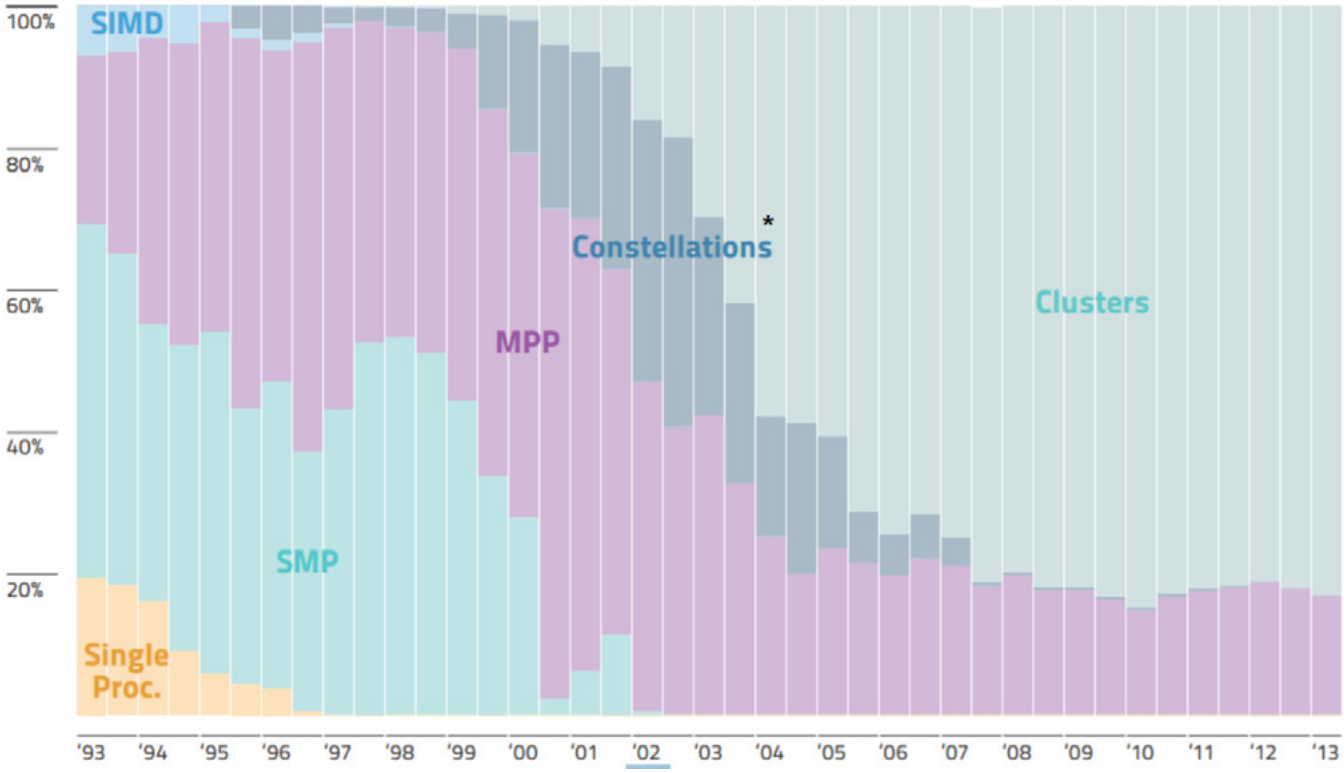
という見方がもたらされました。これは、システム・ソフトウェアの開発者には、非同時性、レイテンシー、制御のオーバーヘッドへの対応を求める一方、将来のハードウェア・システムの設計者には、エクサスケールの時代に向けてユーザーの生産性とパフォーマンスの移植性に取り組むように働きかけます。過去 20 年間の素晴らしい成果は、スケーラブル・コンピューティングの次の 20 年間につながるでしょう。」

OpenMP*: 将来を考慮した、開発者による取り組み

OpenMP* イニシアチブは、デベロッパー・コミュニティーの働きかけによって発足しました。当時、異なる共有メモリー型並列プラットフォーム間でコードを移行するための、信頼性のある標準規格に対する関心が高まっていました。

OpenMP* が登場する前は、Pthreads などのスレッド化モデルを明示的に使用するか、MPI などの分散フレームワークを使用して並列コードを作成する必要がありました。(最初の MPI 標準は 1994 年に完成しました。) OpenMP* プラグマを追加するだけで、共有メモリーモデルで並列処理を利用できるという便利さは画期的でした。しかし、シングルスレッド・プロセッサーのクラスターが主流だった当時のハイパフォーマンス・コンピューティングでは、スレッドベースのコンピューティング・モデルに対する関心は限られていました。一部のハードウェア・プラットフォームでは、プラグイン CPU を追加することで、ハードウェア・ベースのマルチスレッド・パフォーマンスを利用できましたが、一般にスレッドは、スケーラブルな並列パフォーマンスではなく、OS タイムスライスを使用して非同期動作をエミュレートするソフトウェア手法と考えられていました。当時、スレッドに関する議論の焦点は、メモリーを共有する軽量なスレッドではなく、フォーク / ジョインで生成されるプロセスなどの重いスレッドの使用でした。ノード内のハードウェア並列処理は、デュアルコアまたはクアッドコア・プロセッサー・システムに限定されていたため、OpenMP* によるスケーリングは取るに足らない問題でした。

このため、分散メモリー MPI コンピューティングが並列処理への唯一の道と考えられていた中、1997 年に発表された OpenMP* 仕様は、将来をよく考慮したものであると言えます。基本的に、ネットワークを介して多数のマシンを接続することは安価で簡単です。**デナード・スケーリング則** (英語) の世界では、アプリケーション・パフォーマンスの向上は、MPI ノードを追加するか、シリアル・ソフトウェアをより高速に実行できるより高いクロックレートのプロセッサーを搭載したマシンによって達成可能です。そのため、COTS (既製品の) ハードウェアを使用してクラスターを構築することでスピードアップが図られ、ハイパフォーマンス・コンピューティングでは一般にこの手法が採用されていました (**図 1**)。例えば、1998 年のオリジナルの Beowulf のハウツーでは、「Beowulf は、クラスター・コンピューターで並列の仮想コンピューターを形成しており、多数のワークステーションではなく単一のマシンのように動作します。」と説明しています。つまり、科学または商業分野においてマルチコア・プロセッサーに対する需要がほとんどなかったのです。マルチスレッド並列コンピューティングは、主流のプログラミング・モデルというよりも、興味深い HPC プロジェクトという程度でした。**図 1** に示すように、超並列 SIMD (Single Instruction, Multiple Data) の時代は短く、SIMD アーキテクチャー・ベースの CM-2 スーパーコンピューターと CM-5 MIMD (Multiple Instruction, Multiple Data) 超並列プロセッサー (MPP) スーパーコンピューターの製造元である Thinking Machines Corporation の消滅とともに幕を閉じました。この時代の SMP (ここでは共有メモリー・マルチプロセッサーを表す) の一例として、SGI* Challenge があります。

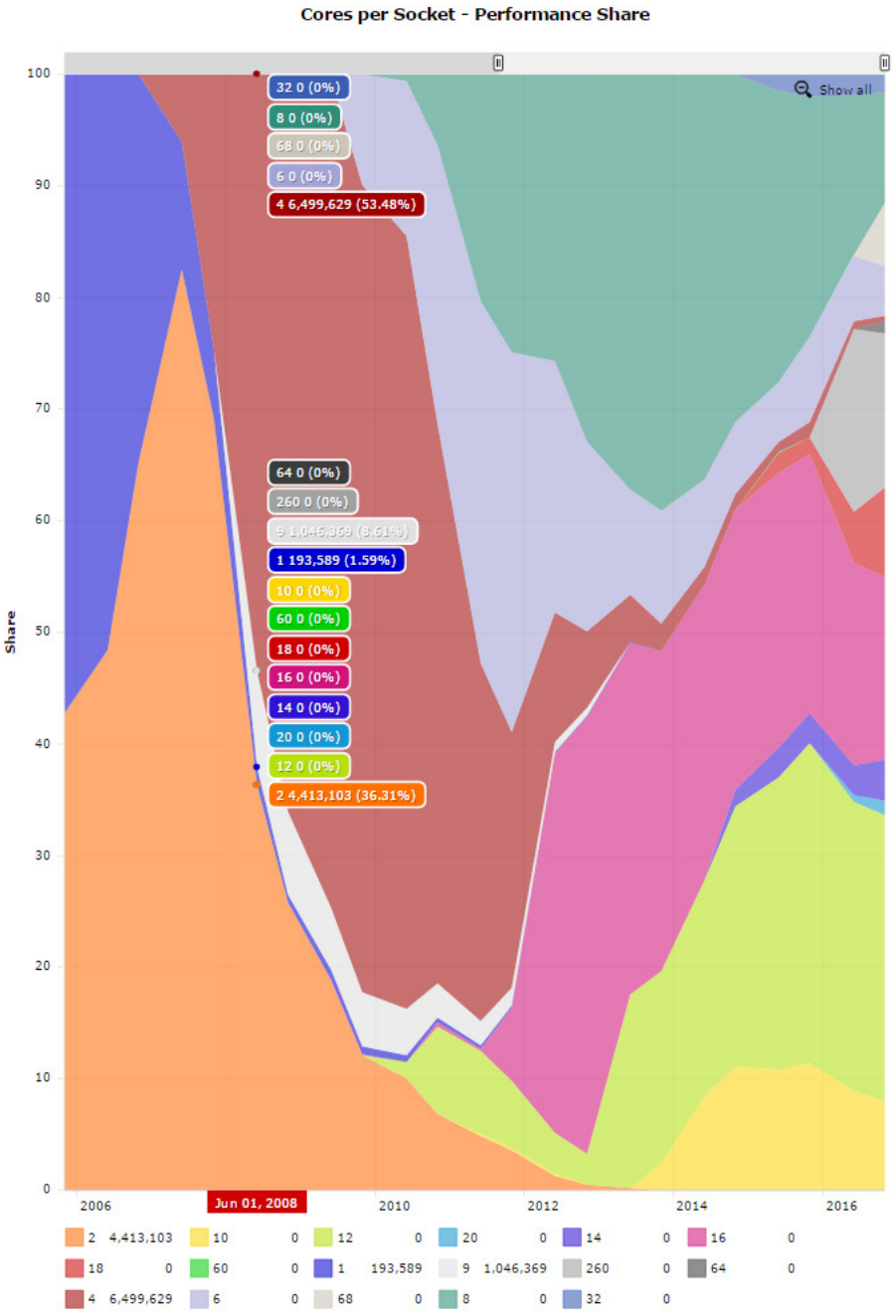


1 年ごとの TOP 500 に占めるシステム・アーキテクチャーの割合 (出典 : **top500.org** (英語))
(コンステレーション (Constellations) は、大規模 SMP システムのクラスターを指します。)

OpenMP* 時代の幕開け : デナード・スケーリング則の衰退とマルチコアへの関心の高まり

2005 年から 2007 年の間に、デナード・スケーリング則は衰退し、現代のマルチコア・プロセッサーが登場し始めました。クロックレートの上昇により大幅なパフォーマンスの向上が達成できなくなったため、さらなるパフォーマンスを生み出し、アップグレードを促すためにも、メーカーはプロセッサー・コアを増やし始めました。この方針転換により、次世代のハードウェアではクロックレートの向上により自動的にコードが高速化される、という期待はできなくなりました。その結果、アプリケーション・パフォーマンスを向上する手段として、スレッドベースのコンピューティングの本格的な調査が始まりました。とは言うものの、ほとんどのアプリケーションは、マルチプロセッサー上でコアごとに 1 つのシリアル MPI ランクを実行するように並列化されただけでした。

図 2 は、TOP500 のパフォーマンス・シェアを示すグラフです。このグラフから分かるように、2007 年から 2008 年の間に、パフォーマンス分野ではマルチコア・プロセッサーが主流となりました。それ以降、コア数が増加傾向にあることは一目瞭然です。



2 TOP500 におけるコア数の増加によるパフォーマンスの向上 (出典 : **top500.org** (英語))

OpenMP* によるコードの現代化

コア数の増加は、OpenMP* と MPI プログラムにさらなる並列性による利点をもたらしました。しかし、OpenMP* を主要 API へと成長させた要因は、ベクトル並列処理と多数のコアを搭載したプロセッサーです。

多くのレガシー・アプリケーションは、並列化がパフォーマンスへの道とされた COTS ハードウェア向けに作成されていたため、プロセッサー・コアごとに 1 つの MPI ランクを利用していました。コードが (特に HPC コードでは) ベクトル化されていなかったわけではなく、COTS クラスターで使用されていたプロセッサーのベクトル幅が狭かったため、パフォーマンスが制限されていました。また、ベクトルユニットのプログラミングも困難でした。その結果、多くのプログラマーは、MPI のさらなる並列化によってもたらされるパフォーマンスの向上を頼りにし続け、各 MPI ランク内で実行されるベクトル化されたループの利点は、付加的な利点と考えられていました。

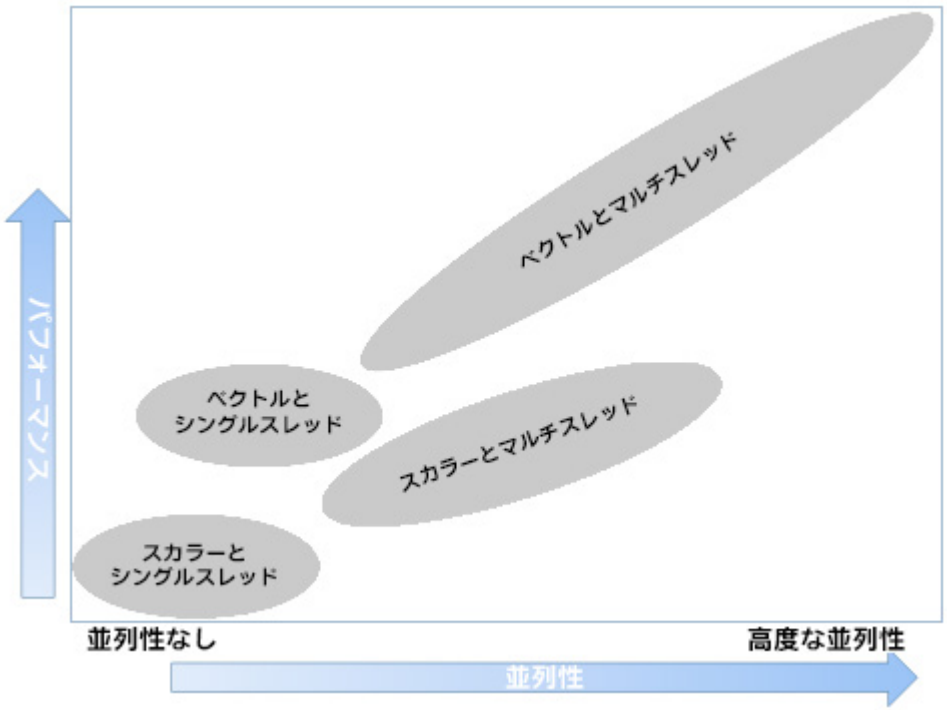
2006 年頃から始まった SIMD とデータ並列プログラミングの再興は、レガシーコードをハードウェア・スレッド並列処理に対応させることで、広範なアプリケーションと計算ドメインでパフォーマンスを大幅に向上できることを示しました。

ペタスケール、そしてエクサスケール・コンピューターへの道のりにおいて電力効率が大きな課題であることが分かると、この傾向はさらに加速しました。2007 年に登場した **Green500** (英語) リストにより、大規模コンピューティングにおける " どんなことをしてでもパフォーマンスを向上する " 時代は終わりを告げました。

突如として、OpenMP* は、電力効率の良いコンピューティングとデータ並列性を利用する手段として位置付けられました。CPU は SIMD コードを効率良く実行できる汎用 MIMD デバイスであり、SIMD コードは、ハードウェア・ベクトル・ユニットに効率良くマップすることができます。そして、MIMD ベースのタスク並列処理は、単なるループ構造です。

パフォーマンスと電力効率の両方を向上するため、x86 ISA (命令セット・アーキテクチャー) に幅の広いベクトル命令が追加されました。ほかの ISA でも同様の取り組みが行われています。ハードウェア・ベクトル・ユニットは、シリコンチップ上で比較的小さな場所を占めますが、非常に電力効率の良い浮動小数点パフォーマンスを提供します。これにより、汎用プロセッサーの浮動小数点処理能力が劇的に向上し、コア数が多い (メニーコア) ベクトル並列プロセッサーの時代が幕を開けました。例えば、メニーコアのインテル® Xeon® プロセッサーやインテル® Xeon Phi™ プロセッサーなどです。

コアごとに 1 つの MPI ランクを利用するプログラミングは、SIMD、データ並列性、ベクトル・プログラミングによるパフォーマンスの利点を最大限に引き出すことができないため、非効率的なモデルであるとされ、コードの現代化に注目が集まりました。例えば、最新のハードウェアでは、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-512) 命令を効率良く利用することで、アプリケーション・パフォーマンスを倍精度コードで 8 倍、単精度コードで 16 倍向上できます。多くのプログラミング・プロジェクトは、OpenMP* と MPI 双方の利点を最大限に利用するため、OpenMP*/MPI を組み合わせたハイブリッド・モデルに移行済みか、移行中です。移行によってもたらされるパフォーマンスは、**図 3** に示すように、コア数とベクトル・パフォーマンスに依存します。最新のインテル® Xeon Phi™ プロセッサーでは、コアごとに 2 つのインテル® AVX-512 ベクトルユニットがあります。



3 最高のパフォーマンスは、図の右上のように、ハードウェアによるベクトル化と並列化の両方を利用した場合に得られる
(図の出典 : Morgan Kaufmann、imprint of Elsevier、無断での引用、転載を禁じます)

OpenMP*: 最先端の API

OpenMP* 標準は、SIMD プログラミングの重要性を認識し、2013 年 10 月に OpenMP* 4.0 標準で SIMD 節を実装しました。OpenMP* 4.0 仕様では、コプロセッサーと GPU などのアクセラレーターへのオフロードモードのプログラミングに対応するため、その他の節も追加されました。OpenMP* は、ハードウェアの変化にも対応しつつ、成長を続けています。

エクサスケール時代向けの OpenMP*

エクサスケール時代では、電力消費が最も重要になります。エクサスケール・コンピューティング・アーキテクチャーでは、電力効率の良いシリアルコアを並列ハードウェアとリンクする、**アムダールの法則**のハードウェア・インスタンス化とも言うべき傾向にあります。NERSC は、最新の Cori スーパーコンピューターは、コードをチューニングしなければ、プログラムが低速で実行するため、初心者が最先端のスーパーコンピューターで実行するようなものだとしています。エクサスケール・スーパーコンピューター向けの電力効率の良いシリアルコアは、シリアルコードの実行により多くの時間を要するため、これは不可避です。この傾向は、業績と利益にとって電力効率が極めて重要なデータセンターにも波及する可能性が高く、さらに、5G ではデータ量が最大 1,000 倍も増加することが予想されています (出典 : Forbes)。

幸いなことに、OpenMP* は、異なる共有メモリー型並列プラットフォーム間でコードを移行するため信頼性のある方法を提供するという最初の目標を達成しつつ、優れたパフォーマンスをもたらすことが実証されています。パフォーマンスと移植性—なんて素晴らしい組み合わせでしょう。

Rob Farber は、HPC 分野について幅広い知識を持った著者であり、グローバル・テクノロジー・コンサルタントです。可搬性が高く、パフォーマンスに優れた並列プログラミングの積極的な支持者です。Rob へのお問い合わせは、**info@techenablement.com** (英語) までお送りください。

アプリケーションを並列に実行するための OpenMP* 入門 (英語) ›