SSTC
High Performance and Productivity
スケーラブルシステムズ株式会社

Whitepaper

コンピュータプラットフォームでの並列処理の新たな展開と挑戦

スケーラブルシステムズ株式会社

コンピュータプラットフォームでの並列処理の新たな展開と挑戦

2006.07.15

はじめに

計算機に対する恒久的な性能向上の要求に対応するために計算機は常に進化を続けています。現在ではそのような性能向上に対する要求に応えるため、計算機システムには様々なレベルでの“並列処理”を可能とする技術が組み込まれています。これらの“並列処理”はユーザが意識することなく、その処理がなされていることもあり、普段意識することは少ないかもしれません。また、コンピューティングについては常に多くの話題があります。しかし、“並列処理”が大きなニュースとして取り上げられる機会が多いとは言えません。唯一、並列処理が大きなトピックスとして取り上げられ、そのテクノロジーについて常に大きな関心が向けられている分野があります。ハイパフォーマンスコンピューティング（HPC）分野と呼ばれる高性能コンピュータシステムを利用し、エンジニアリング設計やサイエンスにおけるシュミレーション、気象予測や地球環境シュミレーションなどの用途で利用され、問題をいかに短時間で解くかが常に求められている分野です。

この HPC 分野では、問題をより短時間で処理するために、様々な並列処理のテクノロジーが提案、開発されてきました。今後、エンタープライズ、デジタルオフィス、そしてデジタルホームで必要とされる処理性能の向上にはこれらの HPC 分野で求められる並列処理の一層の活用が必要になります。

コンピュータプラットフォームでの並列処理

なぜ、IT インフラのコアとなるコンピュータプラットフォームでは、並列処理がより重要になってくるのでしょうか？

コンピュータプラットフォームに対する継続的な処理性能の向上とリアルタイム性の追求、そして経済性とエネルギー効率への厳しい要求に対して、コンピュータプラットフォーム自身も大きく変わろうとしています。その中心となっているのが、マイクロプロセッサの 64 ビット化と“マルチコア化”です。これらの 64 ビット化されたマルチコアプロセッサを採用したサーバ、ノート/デスクトップが市場に投入されることにより、計算機を含む IT インフラは大きな変革の時を迎えています。このような IT インフラの変化に対応することが現在、求められているためです。

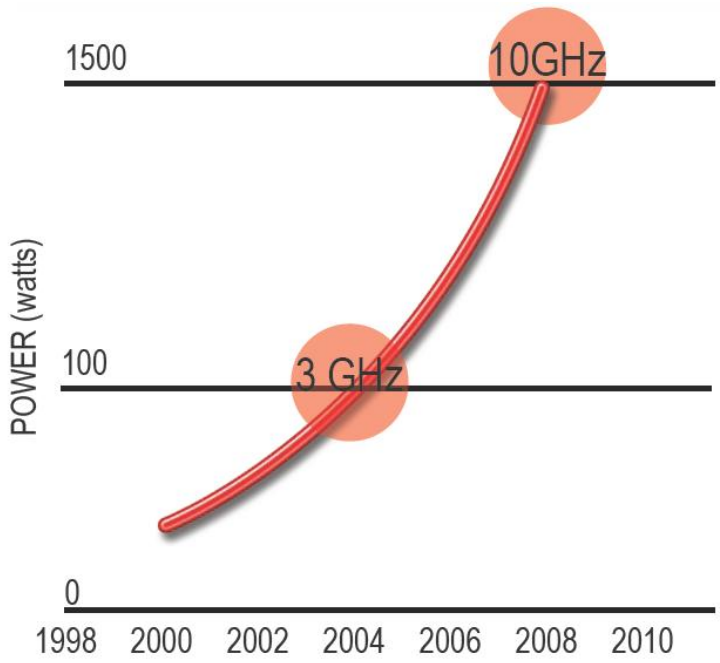
コンピュータプラットフォームの頭脳となるマイクロプロセッサは、求められる性能向上に対してトランジスタ数が“ムーアの法則”の予測に対して指数関数的に増加することでその性能向上を図り、同時により高いクロックスピードを目指したプロセッサの開発が行われてきました。マイクロプロセッサは、より高いクロックスピードを目指した開発と並行してそのアーキテクチャも進化しています。これまでのプロセッサの開発では、その性能を引き上げるために主に次の 2 点を重点とした開発が行われています。一つは、動作周波数（クロック・スピード）の向上であり、もう一つは、1 サイクルで実行できる命令数（IPC）の向上です。

動作周波数は命令実行のパイプライン化とそのパイプラインを細分化することでその向上が図られてきました。実際には、過去に得られたプロセッサ性能向上の 80%はその動作周波数の向上によって実現されたというレポートもあります。IPC の向上には命令レベルの並列性（ILP：Instruction Level Parallelism ）を高めるためにスーパースカラ命令による複数の命令の同時実行が大きく寄与しています。このスーパースカラ命令での複数命令の実行を可能とする様々な高速化技術（Out-of-Order 実行や命令の投機的実行）などによって、その並列実行性は非常に高いものとなっています。しかし、これらの多くの高速化技術での投入によってプロセッサは非常に複数化し、同時に、その動周波数の向上は消費電力と発熱という問題をより深刻化させています。このプロセッサのロジックの複雑さと消費電力と

発熱の問題は、プロセッサの動作周波数の向上を従来と同じペースで図ることをより困難にしています。

動作周波数とスーパースカラによる命令発行は、メモリ待ち時間の増大に対応するためにより大容量のキャッシュを実装することで、マイクロプロセッサの性能向上に大きく寄与してきました。歴史的には、動作周波数の増大は、半導体デバイスのスイッチング遅延の改善のための半導体の微細化とより深いパイプライン化によって、命令実行をより多くのクロックサイクルに分散させ、各サイクルでの処理を減らすことで実現されています。パイプライン化には、様々な弊害もあり、より深いパイプラインでは、制御フローの複雑化という問題とペナルティの問題があり、高度な分岐予測や投機的実行等といった非常に複雑な機構が必要になっています。

プロセッサの周波数は、ある意味、性能の尺度としての位置づけにあり、より高い動作周波数が市場の要求であったことも歴史的には事実ではありましたが、より高い周波数とより深いパイプラインの実装があるレベルを超えた時点、そのマイナス面、特に深いパイプラインの非効率さと電力消費と低いエネルギー効率が問題になってきています。

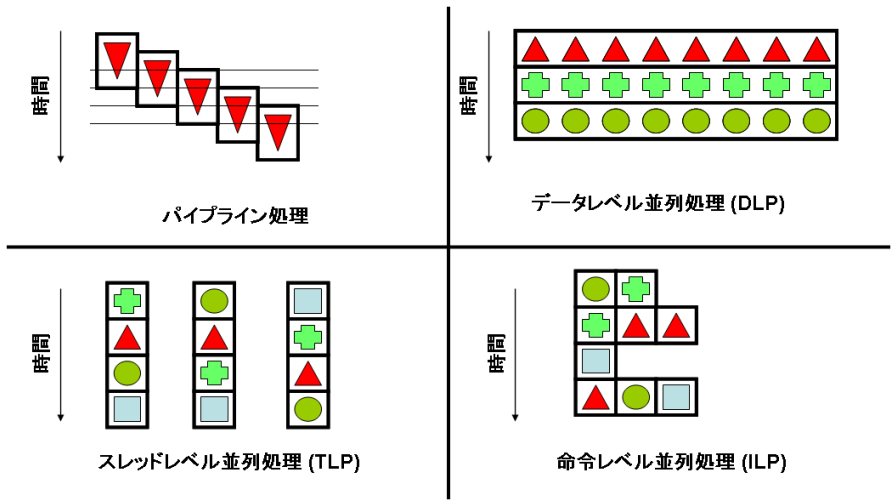


動作周波数とパイプライン化による性能向上の目標

マイクロプロセッサの動作周波数として、10GHz を目指す開発プランは、その必要とする電力消費量が現実的なレベルを越えて増大したため、その開発プラン自身の見直しが必要でした。

スーパースカラ命令の発行に関しても限界があります。パイプライン化と同じように、効率損失の問題はスーパースカラ命令の実行でも深刻な問題となっています。スーパースカラ命令では、実行時の同時並列処理の問題と共に、プログラムのアルゴリズムでの命令実行の並列実行を可能とするその依存性解析が重要となり、可能な限り命令実行スロットを埋めるために、動的に命令のリオーダーなども必要です。そのため、スーパースカラの並列性能の向上では、そのスロットの増加とそのスロット利用の効率化を図ることは、指数関数的な複雑性をもたらします。もう一つ、スーパースカラで問題になるのは、その技術が基本的にはプログラムにおける命令レベルの並列性（ILP）に依存することです。ILP が少ないコードでは、スーパースカラの効率は、如何に複雑な処理ロジックを組み込んだ実行ユニットでも、その実行ユニットをフルに稼動させることが困難になります。また、一般に ILP として利用されるプログラムの並列実行

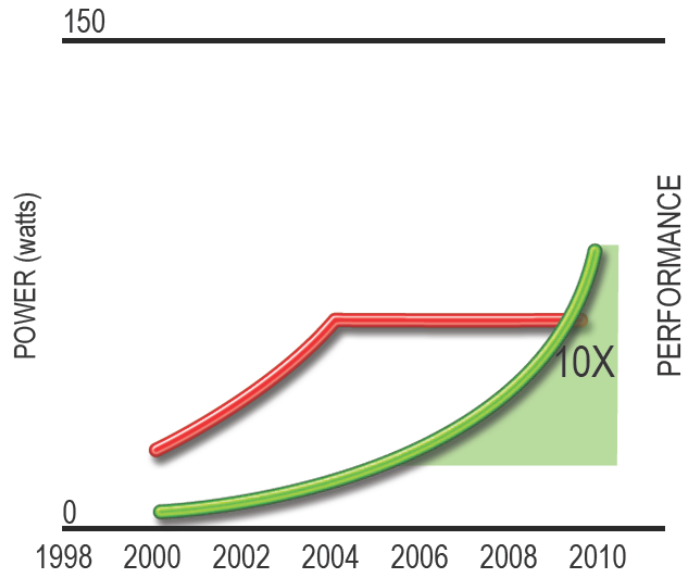
性は、実は、データレベルの並列性（DLP）であることが、様々な研究で明らかになっています。したがって、このようなDLP の利用をより効率的に利用して、高性能を実現することが必要となります。



マイクロプロセッサでの並列性の利用

マイクロプロセッサでは、その処理を時間的、空間的に並列に処理することで、高速化を図っています。このような時間的、空間的な並列処理を実現するために、様々なテクノロジがマイクロプロセッサには組み込まれています。

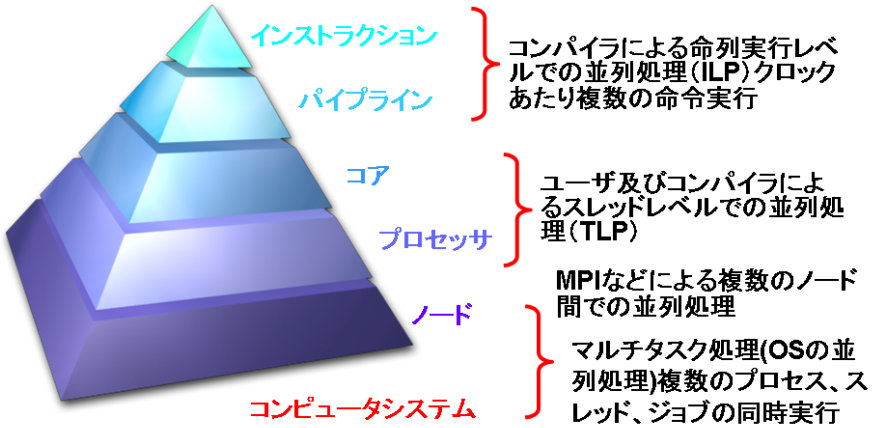
このように従来のパイプライン化やスーパースカラ命令といったテクノロジだけでは、市場が要求する性能向上の実現が困難になっても、マルチメディアやトランザクション処理、製造業での設計計算、先端の科学技術シュミレーションといった多くの分野で、プロセッサの性能はより高いものが求められています。また、システムのセキュリティ機能の強化や省電力のための電力管理といった機能を、プロセッサの性能を損なうことなく実装することが求められています。



性能と消費電力

現在のマイクロプロセッサの開発は、消費電力を増加させることなく、その性能を大幅に向上させることを目指しています。

そのためには、新しいレベルでの並列処理技術が必要になっています。そのための、並列処理技術として、ベクトル処理、マルチスレッド処理、マルチコアでのマルチプロセッシングなどが、現在のマイクロプロセッサでは採用されています。これらは、パイプライン化やスーパースカラとは、異なったレベルでの並列性を活用し、また、複雑な制御構造をプロセッサに加えることなく、ソフトウエアの支援によって、高い実行性能を実現することを目指しています。パイプライン化とスーパースカラでは、粒度の小さい命令レベルの並列性（ILP）を活用し、パイプライン化では時間的な並列性を、そして、スーパースカラでは、空間的な並列性を利用しています。一方、ベクトル処理では、データレベルの並列性を活用し、マルチスレッド処理では、スレッドレベルでの並列性（TLP）を利用し、マルチプロセッシングでは、プロセスレベルでの並列性を活用しています。



コンピュータシステムでの並列処理

コンピュータシステムは、コンポーネントの組み合わせによって構成されますが、これらのコンポーネント内での処理の多くは、様々な形で並列に実行されています。これらの並列処理は、様々なレベルで実行されていて、ユーザがそれを意識する場合も、意識しない場合もあります。

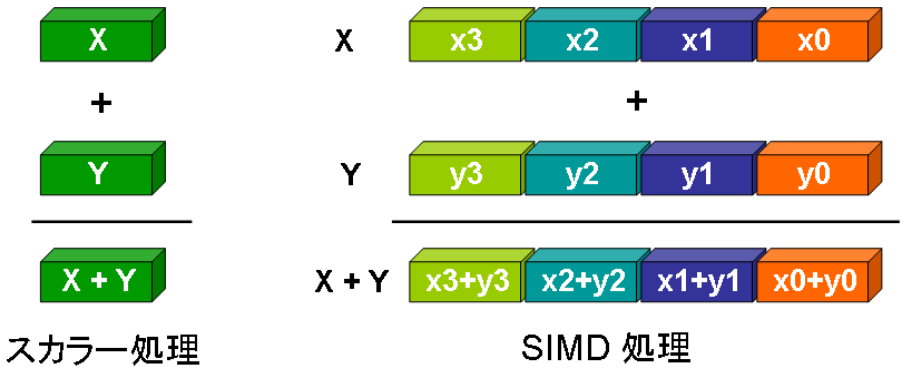
ベクトル処理

スーパースカラによる複数の命令の同時発行と実行に際して、Flynn のボトルネックと呼ばれるコンピュータアーキテクチャ上の問題点がその性能を大きく制限することになります。ベクトル処理は、この Flynn のボトルネックに対して、一つの命令でより多くの演算を処理し、命令のフェッチとその発行を減らし、ボトルネックの回避を図ることを可能とします。

ベクトル処理は、もちろんデータレベルの並列性があり、そのデータ処理の実行に際して、実行順序に関する依存性が存在しないことが前提になります。ベクトル処理では、データレベルの並列性が明確な行列演算や代数計算を多用する科学技術計算用に開発された CRAY のスーパーコンピュータなどのベクトル計算機で活用されていました。しかし、これらのベクトル計算機は、非常に高速なベクトル処理は可能でしたが、相対的にスカラー計算の性能が低く、アムダールの法則で示されるように非ベクトル化部分の性能が結局、その性能を制限することになり、マイクロプロセッサのスカラー性能の向上によって、ベクトル計算機は非常に限定された用途のみで利用される専用システムとなっています。

一方、マイクロプロセッサを搭載した PC は当然、その処理性能の向上をマイクロプロセッサでのパイプライン化とスーパースカラによって実現してきました。そのため、PC でのワークロードの処理は、基本的には、‘スカラー’処理として発展してきました。しかし、PC でのデジタルメディア処理の必要性がより強く求められることにより、PC が必要とする技術が急速に変化してきました。市場規模とマーケットのニーズ、そして、新しいマーケットへの展開によって、PC におけるデジタルメディアの処理の高速化に対する非常に強い要望に答えることが求められてきました。このようなデジタルメディア処理を高速に処理することが可能なアーキテクチャがマイクロプロセッサに求められることによって、ベクトル処理は、別の形で復活しその性能に関する改善が一層進むことになります。

マルチメディア処理で要求される計算処理は、汎用のスカラー処理よりも科学技術計算での計算処理により近い処理フローを持ちます。データをブロック化し、そのデータに対してフィルタリングや圧縮、復元などを処理する場合、それらのデータを同時並行して処理することが可能です。これは、科学技術計算で多用される配列要素に対する反復処理に近いものであり、非常に計算集約型の演算となります。また、マルチメディア処理では、科学技術計算では強く要求されないリアルタイム性が強く求められます。科学技術計算は基本的には、バッチ処理であり、リアルタイムの処理性能が求められることはあまりありませんが、マルチメディア処理では、実行時間に確実性が求められ、リアルタイム性能が重要になります。



スカラー処理と SIMD 処理

SIMD 処理では、ショートベクターでの並列性を利用し、データの各要素を並列に処理します。一つのオペレーションによって、複数の要素に対する処理を可能とすることで、処理の高速実行を可能とします。

これらのマルチメディア処理を高速に、かつリアルタイムで処理するには、ベクトル処理が有効であり、また、このようなベクトル処理をに適したベクトル・アーキテクチャが求められました。このベクトル・アーキテクチャは、初期のベクトル計算機でのベクトル処理のためのアーキテクチャよりも、多くの意味で優れたベクトル処理が可能となっています。

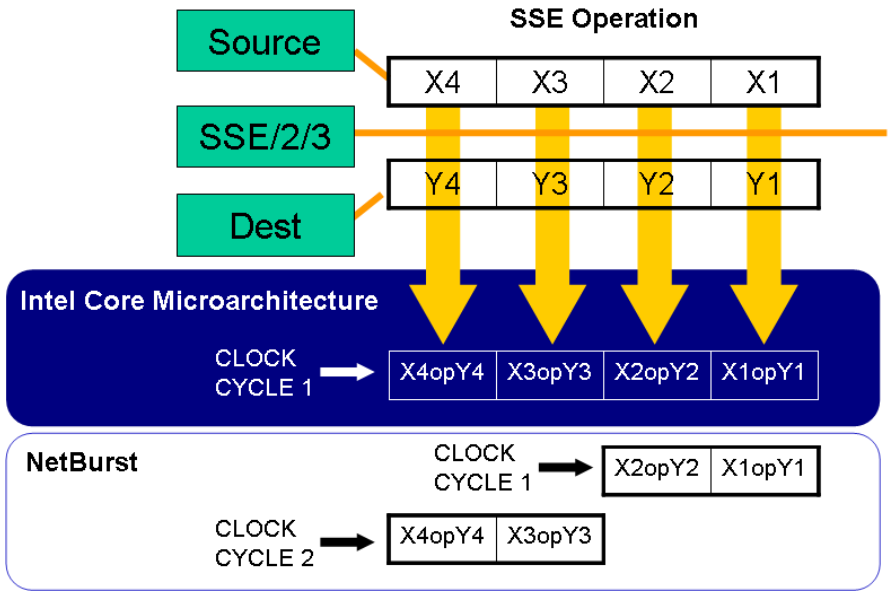
この新しいベクトル・アーキテクチャは、SIMD（Single-Instruction-Multiple-Data） と呼ばれるものです。ここでの SIMD アーキテクチャは、著名な Flynn のコンピュータアーキテクチャの分類で意図したものよりも、限定されたものとなっています。

現在のマイクロプロセッサは、この SIMD アーキテクチャによる実行ユニットをプロセッサに組み込んでいます。この SIMD ユニットは、既存の汎用レジスタを利用することで、スカラー処理とシームレスに結合されます。スカラー処理の加速を行い、マイクロアーキテクチャの改良によって、同時にベクトルエンジンとしての SIMD 演算ユニットを利用して効率化を図ることが可能となっています。これによって、初期のベクトル計算機が悩まされたアムダールの法則による

性能の問題を減らすことを可能としています。また、SIMD アーキテクチャでは、ベクトル計算機のようにパイプライン化された演算ユニットを利用するのではなく、空間的に並列に実行可能なユニットを利用することで、プロセッサを複雑化することなく実装されています。

SIMD ベクトルは、デジタル信号とマルチメディア処理を目的として開発されたため、16 ビットのデータを断片的に処理するような処理を行うことが必要です。SIMD ベクトルユニットでは、128 ビット幅でのデータ処理が可能であり、一つの命令で、16 ビットのデータに対して、8 つの演算が可能となっています。この SIMD ユニットは、16 ビット幅のデータだけでなく、32 ビット、64 ビットのデータを処理することも可能となっています。これによって、この SIMD ユニットは、通常の科学技術計算などの用途でも利用できるようになっています。科学技術計算では、ベクトル処理は非常に重要であり、高速のスカラーユニットとベクトルユニットが連携して処理を行うことが可能なアーキテクチャは、アプリケーションレベルの高速化を図る上で、最も求められるものとなっています。

一方、SIMD アーキテクチャの欠点は、データレベルの並列性をハードウエアに対して、ソフトウエアが明示的に指示する必要がある点です。しかし、現在のコンパイラは、データレベルの並列性とその処理の相互依存性の有無を判断し、SIMD の処理命令を行うコードを生成します。自動ベクトル化機能は、コンパイラの基本機能となっています。また、コンパイラは、コンパイル時に、ベクトル化処理を行ったかどうかのメッセージなどを出力するため、ベクトル化のためのアルゴリズムやコーディングの変更などを行うことで、更に高速実行を可能にします。



インテル Core マイクロアーキテクチャ - Advanced Digital Media Boost

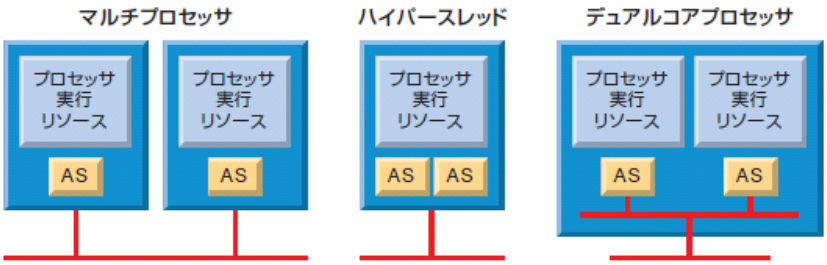
Intel Core マイクロアーキテクチャでは、従来のインテルマイクロアーキテクチャと比較しても実行ユニットの強化が大幅になされています。実行ユニットが強化され、より多くの命令が 1 クロックで実行可能となっています。

マルチコア･プロセッサによるマルチプロセッシング

マルチコアプロセッサの構成自身は非常にシンプルです。例えば、デュアルコアプロセッサはシステム内に 2 つのマイクロプロセッサが SMP（対象マルチプロセッサ）として実装された場合と同じように利用できます。将来的により多くのコアがマイクロプロセッサ上に実装されることになりますが、基本的にはそれらは複数のプロセッサがメモリを共有

して提供される SMP システムとして実装されます。

このようなマルチコアプロセッサでは、プロセッサ内でクロック当たりに処理できる命令は従来のシングルコアのプロセッサよりも大幅に増えます。デュアルコアプロセッサは、同時に 2 つのスレッドの処理を可能とします。コア数が増えれば、それだけ、多くのスレッドの同時実行が可能になります。マルチコア化によるマイクロプロセッサ自身の性能向上のペースは従来のクロックスピードで実現されていた性能向上を大きく上回る可能性があります。



AS (Architecture State) は、汎用レジスタや制御レジスタ、APIC (Advanced Programmable Interrupt Controller) レジスタなどプロセッサの状態を保持するものです。

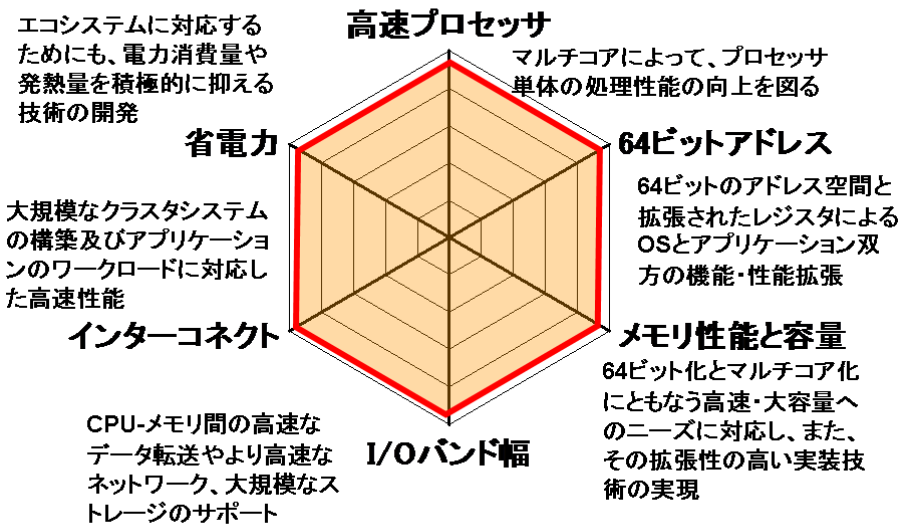
マルチコアプロセッサの実装

マルチコアプロセッサは 2 つ以上の“実行コア”を 1 つのプロセッサ内に実装しています。このマルチコアプロセスはシングルプロセッサのソケットに挿入することが可能であり、オペレーティングシステムはプロセッサ内の“実行コア”をそれぞれ独自のプロセッサと認識し、プロセッサの全リソースを同時に利用することを可能とします。デュアルコアプロセッサは 2 つの実行コアをもつマルチコアプロセッサとなります。

「インテル® コンパイラ OpenMP* 入門

デュアルコア/マルチコア対応アプリケーション開発①」より

しかし、マルチコア・プロセッサでは、複数のスレッドを同時に実行できなければ処理性能の向上を図ることが出来ません。このためには、システム全体とアプリケーションがマルチスレッドを意識し、その最適化を図る必要があります。このようなシステム最適化は、マルチコア化とそのマルチコア化に伴うプラットフォームの技術革新による性能向上、機能拡張、柔軟性、運用管理の強化とそれらのプラットフォーム基盤の潜在能力を引き出すソリューション構築のサポートが不可欠となります。



要求されるのはシステムの‘バランス’

マルチコアでのマルチスレッド処理では、メモリや I/O を共有した独立したプロセッサコアでの処理が行われています。このため、プロセッサの処理能力はシングルコアと比較して、大幅に向上しますが、メモリや I/O の共有のため、リソースに関する競合の可能性が高くなります。そのため、従来のシステム以上にメモリや I/O には高い性能が要求されます。64 ビット化やマルチコア化にともなう高速・大容量メモリのニーズに対応するために次世代の標準 I/O やメモリシステムが求められています。

マルチスレッド処理

デスクトップやラップトップコンピュータから大規模なデータセンターのシステムまでほとんどのコンピュータは同時に複数のタスクを処理しています。オペレーティングシステムも、これらの複数タスクの効率的な処理を行うために、マルチコア、マルチスレッドへの対応が求められています。

多くのオペレーティングシステムでは、複数のタスクをマルチプロセス（マルチタスキング）として処理しています。しかし、現在のオペレーティングシステムが行うマルチタスキングは、シングルコアのプロセッサ上では実際には１つのスレッド（スレッドとは、オペレーティングシステムが処理を行う基本単位）だけを実行するために、他のスレッドの実行を停止させています。したがって、複数のタスクを同時に実行しているとしても、プロセッサ上では 1 つのスレッドだけが実行されています。マルチタスクは各アプリケーションの実行時間を短縮するものではなく、実際には複数のタスクをオペレーティングシステムがその実行を切り替えながら実行するためにオーバーヘッドが生じる可能性もあります。マルチコアプロセッサでは同時に複数のタスクを実行コアで処理し、リソースを各スレッドが占有することが可能です。そのため、複数のプロセスを同時に実行出来るマルチコアプロセッサは、オペレーティングシステムの処理能力の大幅な向上を可能とします。しかし、マルチコアの利点をアプリケーションが引き出すためには、マルチスレッド対応（スレッドをマルチコア上で並列実行）が必要になります。

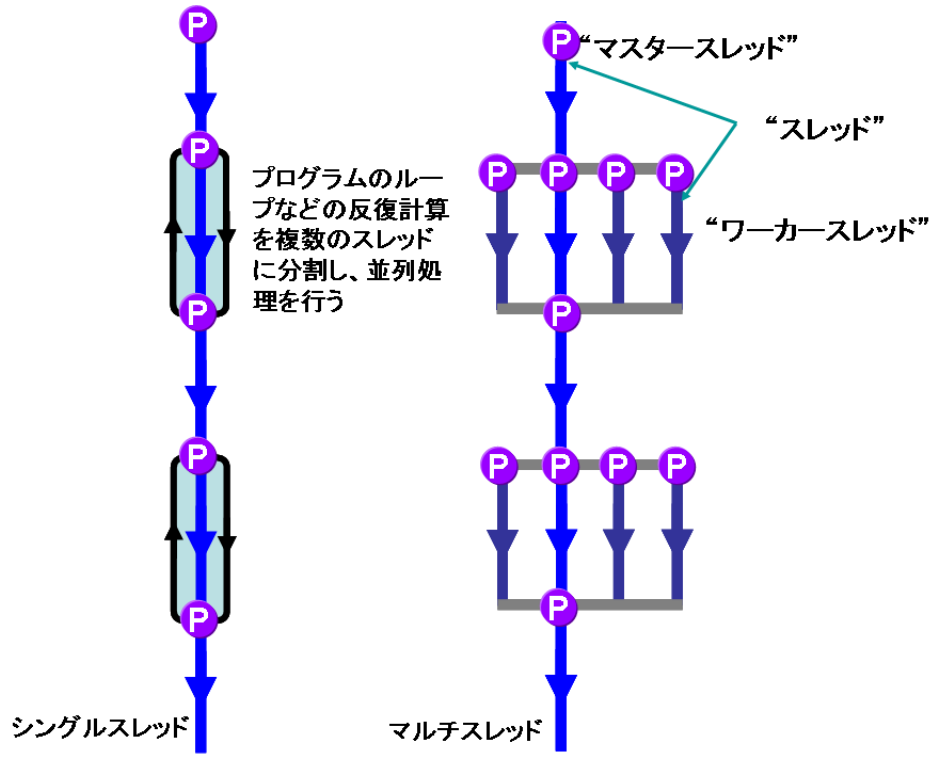
利点	機能
アプリケーションの性能向上	既にマルチスレッドに対応した多くのアプリケーションの実行性能の向上を図ることができます。
エコシステム	電力消費と発熱量を増やすことなくシステム性能の向上を図ることが可能となります。
ターンラウンド	マルチタスクによって、アプリケーション実行の効率化を図り、各アプリケーションのターンラウンドの改善を図ることが出来ます。
インターフェイスの多様化	デスクトップ、ラップトップなどでもより高い機能の実装を可能とし、インターフェイスの多様化への対応を可能とします
将来への投資	従来の計算機システムの限界を打破し、コンピュータの革新をもらたします。

マルチコア上でのマルチスレッド処理の利点

マルチコアプロセッサはコンピューティングにおける大きな革新をもらたします。マルチコアプロセッサによって、“並列処理”が今後、より重大になってきます。コンピューティングにおいて“並列処理”は特別なものではありません。複数のタスクの同時処理においては、マルチコアプロセッサは大きな利点があります。このようなマルチタスク処理においては、従来のシングルコアで実行していたアプリケーションをマルチコアプロセッサ上でのそのまま実行することは可能です。しかし、マルチコアプロセッサ上でのアプリケーション個々の実行性能の向上を図るためには、アプリケーションのマルチスレッド化が必要です。

マルチコア上でのアプリケーションの性能向上を図る並列処理で求められるマルチスレッドプログラミングが最も一般的に活用されているのがハイパフォーマンスコンピューティング（HPC）分野です。HPC 用途で利用されるコンピュータシステムはエンジニアリング設計やサイエンスにおけるシュミレーション、気象予測や地球環境シュミレーションなどの用途で利用され、問題をいかに短時間で解くことを常に求められています。HPC とホームコンピューティングは全く違う世界であると一般には考えられています。例えば、数百から数千のプロセッサをクラスタシステムとして構成した HPC システムをホームコンピューティングで利用するようなことは考えられません。しかし、HPC システムでは常に並列処理ということを考えたシステムの開発構築がなされています。

問題を短時間で解くためには並列処理が必要であり、多くの技術が開発され、利用されています。これらの技術を活用することで、マルチプロセッサ上でのマルチスレッド化を促進することが可能になります。エンタープライズ、デジタルオフィス、そしてデジタルホームで必要とされる処理性能の向上も、HPC が求める計算能力の向上も本質的なところは同じであり、HPC の技術をより有効にメインストリームのマーケットに活用することが求められます。並列処理技術をメインストリームコンピューティングで容易に利用することで、マルチコアプロセッサの利点をさらに発揮できるようにすることが可能となります。



マルチスレッドによるアプリケーションの実行モデル

マルチスレッド化にはアプリケーションプログラムの再コンパイル化や並列化のためのコードの書き換えが必要になります。マルチスレッド化されたアプリケーションでは複数のスレッドをマルチコア上で並列処理することで、より短時間で処理を行うことが可能となります。

開発環境

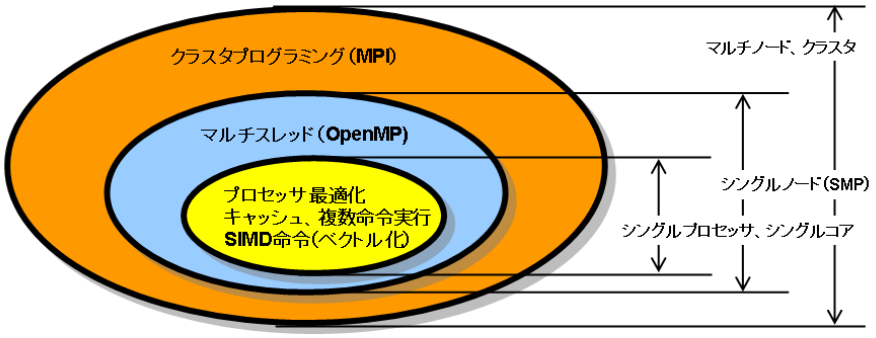
一般に、ソフトウエア資産は、ハードウエア資産よりも長期に継続して利用され、ハードウエアの更新に際しても、新しいハードウエアで継続して利用されます。言い換えれば、新しいハードウエア用にソフトウエアを完全に書き直して理由するようなことは一般的ではありません。そのため、ソフトウエアについては、その移植性やプログラミングモデルの選択は非常に重要になります。一方、ソフトウエアの開発コストは、相対的に増加しており、ハードウエアが複雑化することで、ソフトウエアの設計、開発の生産性の向上は大きな課題となっています。

ベクトル処理では、プログラマ又はコンパイラがデータレベルの並列性を見つけ、そのデータレベルの並列性を利用する SIMD 命令を発行するためのコードを生成する必要があります。このようなデータ並列を見つけることは、ある意味プログラマにとって、あまり関心を払いたくない点であり、コンパイラが自動でそれらを認識し、最適なコードを生成してくれることを期待しています。最新のコンパイラ・システムは、Fortran はもちろん、C/C++プログラムに対しても、高度なベクトル化を行います。また、コンパイラはプログラム構造の解析を行い、ベクトル化を阻害する要因の分析を行い、その内容をメッセージとして、出力することも可能です。これらのメッセージを参考に、潜在的な SIMD 処理の可能性を検討することによって、高速化を実現することが可能となります。スカラーのアルゴリズムをベクトル化し SIMD

処理を行うことは、現在ではマイクロプロセッサで最も性能向上を図ることを可能とするオプションの一つとなっています。(コンパイラによるベクトル化については、参考資料をご覧ください)

マルチスレッドを利用して並列処理を行うには、マルチスレッド・プログラミングが必要になります。マルチスレッドによる並列処理には、プログラムの再コンパイルや並列化のためのプログラムの変更が必要になります。このマルチスレッド・プログラミングについては、既に、高度にマルチスレッド化されたライブラリを利用したり、プログラムのマルチスレッド化を自動で行うコンパイル・ツールを利用することで、容易に並列化を行うことも可能です。また、Win32 や Unix/Linux Pthreads といったマルチスレッドでのプログラミングのための API を利用することも可能です。さらに、マルチスレッド・プログラミングのための強力な API である、OpenMP[1]もインテルコンパイラやマイクロソフトの Visual C++で利用可能です。

OpenMP は、業界標準規格であり、多くの Unix、Linux 及び Windows で利用可能です。Fortran、C/C++の言語規格に準拠しているため、OpenMP を利用することで、プログラムの移植性や互換性を一切、失うことなく並列処理が可能となります。OpenMP を利用したマルチスレッド・プログラミングでは、プログラムに対して、OpenMP API で規定された指示行を挿入し、並列実行のためのプログラム構造を指定します。OpenMP 指示行を指定したプログラムは、コンパイル時に並列化を指示するオプションを指定してコンパイルすることで、コンパイラは並列化コードを作成します。




```
do izone = 1, nzone        ノード内、ノード間並列化
  ……                         MPIやCluster OpenMPなどの利用
  do j = 1 , jmax          ノード内でのマルチスレッド並列化
    ……                         OpenMPやスレッドプログラミング
    do i = 1, imax         プロセッサリソースの並列利用
      ……                       ベクトル化
      ……                       スーパースカラ実行
                               パイプライン処理
    end do                     キャッシュ最適化など
```

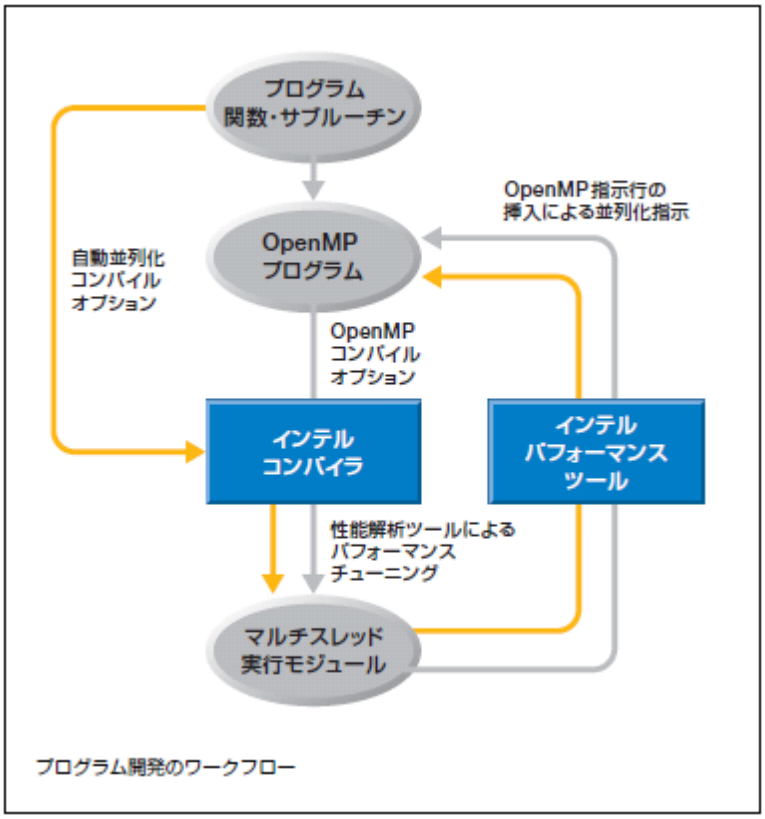
プログラムモデル

プログラミングは、ハードウエアの階層に応じて、用途、目的、プログラムの特性、アルゴリズムなどに応じた最適なものを選択することが出来ます。プログラミングでは、このプログラミング階層間で、他の階層の並列処理を阻害しないように注意する必要があります。

並列化されたプログラムは、複数スレッドが指定されたタスクを同時に実行し、より短時間でプログラムの実行を終

[1] OpenMP の詳細については、OpenMP のホームページ www.openmp.org に詳細な情報があります。

了することが可能となります。



インテルコンパイラでの開発フロー例

OpenMP のような高い互換性と移植性を備えたAPIでの並列化でも、並列化の適用には、多くの試行錯誤とデバッグ作業が必要になる場合もあります。優れた開発環境とサポートによって、これらの並列化作業は良い容易に実行することが可能となります。

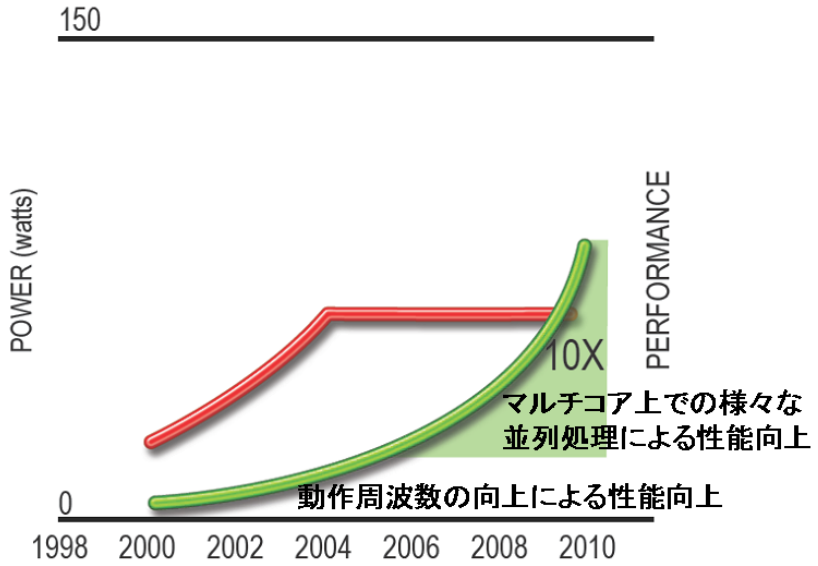
「インテル® コンパイラ OpenMP* 入門

デュアルコア/マルチコア対応アプリケーション開発①」より

おわりに

パイプライン化やスーパースカラ命令といった従来のテクノロジーだけでは、市場が要求する機能と性能を実現することが困難になっており、マイクロプロセッサには、命令レベルの並列実行での性能向上の限界を打破するために、より深いパイプライン化とスーパースカラ命令の強化という技術革新に加えて、ベクトル化やマルチスレッド・プログラミングといった新しいレベルの並列性の利用を可能とする様々な技術が投入されています。

マイクロプロセッサの技術革新と共に、ソフトウエアもコンパイラや開発ツールの機能が向上し、アプリケーションに対する並列処理の適用に関して、その生産性の向上も図られています。



動作周波数の向上から並列処理での性能向上へ

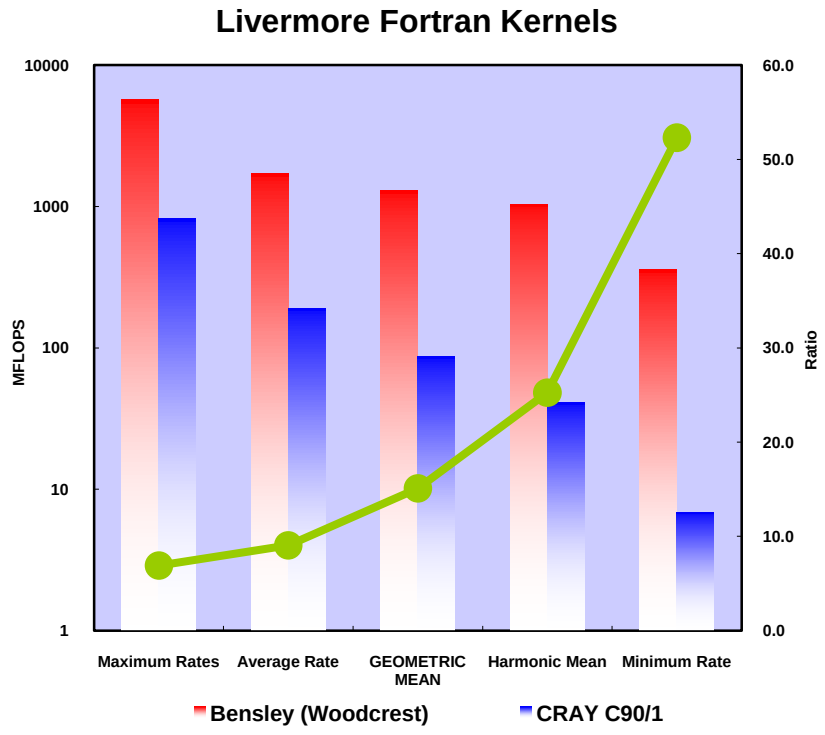
デュアルコアやマルチコアが一般化するに際して、その能力を最大限に発揮するための技術として、今後、ベクトル化やマルチスレッド・プログラミングなどが重要になることでしょう。

ベクトル処理やマルチスレッド処理を可能とするマルチコアプロセッサがマイクロプロセッサの主流になりつつある現在、それらの技術について理解し、様々なレベルの並列処理の活用によって、アプリケーションの性能は従来以上に高速化を図ることが可能となっており、このような高速化はコンピュータプラットフォームを劇的に変化させていきます。

参考資料　1）初期ベクトル計算機と最新マイクロプロセッサの性能比較について

初期のベクトル計算機の性能評価のための著名なベンチマークに「Livermore Fortan カーネル（LFK）」があります。このベンチマークは、米国の Lawrence Livermore National Laboratory（LLNL）が作成したベンチマークコード[2]であり、浮動小数点演算の実際のサンプルとして、LLNL が実際に使用しているシュミレーションプログラムの計算コア部分を抜粋して作成されています。また、幾つかのカーネルは、ベクトルの内積演算、行列積、再帰計算などを含んでいます。

LFK には、ベクトル化可能なループとベクトル化できないループが混在しているため、コンピュータシステムのベクトル処理能力とコードをベクトル化するソフトウエアツールの能力を評価することも目的としていました。LFK は、ベクトル計算機の評価ツールとしては、よく利用されてきましたが　1）浮動小数点演算データのベクトル演算の評価を目的としている　2）Fortran 言語だけの評価　3）カーネルが LLNL の特殊なアプリケーションからの抜粋である　のような理由から CPU コアの評価目的では、全く利用されてきませんでした。しかし、最近は、マイクロプロセッサに SIMD ユニットが組み込まれ、デジタルメディアでの処理でも、この SIMD ユニットでのベクトル化処理性能が非常に重要になっていることからも、ベクトル化機能やベクトル演算ユニットの性能評価などの目的で利用できるのではないかとして、再評価されています。



LFK ベンチマーク性能

CRAY　C90　は、クレイリサーチ社が開発したベクトル計算機で、動作クロックは、250MHz　で、ピーク性能は 1GFLOPS　のシステムです。Xeon　5100　(Woodcrest)は、インテルの最新のマイクロプロセッサで、ここでは、2.67GHz

[2] 参考文献　McMahon, F., “The Livermore FORTRAN Kernels：A computer test of the numerical performance range, ”Technical Report, Lawrence Livermore National Laboratory, Dec. 1986.

のクロックでの評価を行っています。

この結果は、1990 年代の代表的なベクトルスーパーコンピュータである CRAY C90 での LFK の実行結果と最新のインテルマイクロプロセッサ Xeon 5100 番台（Woodcrest）の実行性能を LFK の 72 種類のカーネルループの実行結果の算術、幾何、調和平均で比較したものです。LLNL は、算術平均はベクトル化されたアプリケーションの性能を、幾何平均は最適化されたワークロードでの性能を、そして、調和平均が最適化されていないワークロードの性能を示すと分析しています算術平均は、両方のピーク性能の比率に近いですが、。最新のマイクロプロセッサの性能は、幾何平均で CRAY C90 の 25 倍の性能を示しています。この結果に示すように、初期のベクトル計算機とは違って、最新のマイクロプロセッサは、ベクトル演算とスカラー演算をシームレスに実行することを可能とし、アムダールの法則の制限を受けることなくベクトル処理が可能となっています。

参考資料　2）インテルコンパイラでのベクトル化

インテルの Fortran/C++コンパイラは、インテルプロセッサの SIMD 命令をサポートする MMX や SSE/SSE2/SSE3 を利用したベクトル化を行います。このベクトル化の適用は、コンパイラが自動で行い、コンパイルオプションの指定だけで実行可能です。

コンパイラによる逐次実行コードのベクトル化に関しては、初期のベクトル計算機の時代から研究が続けられており、これらのベクトル化技術の蓄積により、現在のコンパイラシステムは、高度なベクトル化機能を持ちます。これらのコンパイラの自動ベクトル化機能と現在のマイクロプロセッサの多くが実装している SIMD 命令の実行ユニットとそのための強力な命令セットは、マルチメディア処理の高速化を目指して開発された初期の目的に加えて、科学技術計算や画像処理といった分野でも活用されています。コンパイラの自動ベクトル化機能を利用することで、ユーザは簡単に少ない工数でよりプログラムの高速実行を可能とします。また、プログラムの自動ベクトル化は、プログラムの移植性を犠牲にすることなく、プログラムの最適化を可能とするものであり、プログラムの生産性の向上に大きく寄与します。

このコンパイラによる自動並列化は、Linux でも Windows でも、そして、MacOS でも利用可能です。例えば、Linux 上では、-[a]x{KNBPT} のオプションを指定することで、ベクトル化を含む、プロセッサが持つ命令セットに最適化されたコード生成が行われます。同時に、-vec-report{0123} によって、コンパイラが出力するベクトル化に関する診断メッセージとベクトル化適用の可否の情報を制御可能です。

以下のようなプログラムに対して、自動並列化を適用します。

```
 1  #define N 32
 2  float a[N],b[N],c[N],d[N];
 3
 4  int doit() {
 5          int i;
 6          for (i=0; i < N/2; i++) a[i] = i;
 7          for (i=1; i < N; i++) {
 8                  b[i] = b[i-1] + 2; /* data denedence cycle */
 9                  c[i] = 1;
10          }
11          d[0] = 10;
12          d[1] = 10;
13          d[2] = 10;
14          d[3] = 10;
15          return (0);
16  }
```

この場合、自動ベクトル化を適用した場合、以下のようなメッセージが出力されます。

```
$ icc -c -O3 -vec-report1 main.c -xP
main.c(6) : (col. 2) remark: LOOP WAS VECTORIZED.
main.c(7) : (col. 2) remark: PARTIAL LOOP WAS VECTORIZED.
main.c(11) : (col. 2) remark: BLOCK WAS VECTORIZED.
```

このメッセージは、プログラムの 6 行目のループは自動ベクトル化され、7 行目からのループは、ループを分割し、一部をベクトル化したことを示しています。コンパイラのベクトル化は、ループ形式以外でも、実行ブロック（上

記の例では、11 行目）に対して適用されます。

```
$ icc -c -O3 -vec-report3  main.c -xP
main.c(6) : (col. 2) remark: LOOP WAS VECTORIZED.
main.c(7) : (col. 2) remark: vector dependence: proven FLOW dependence between b line 8, and b line 8.
main.c(7) : (col. 2) remark: loop was not vectorized: existence of vector dependence.
main.c(7) : (col. 2) remark: PARTIAL LOOP WAS VECTORIZED.
main.c(11) : (col. 2) remark: BLOCK WAS VECTORIZED.
```

-vec-report2 と –vec-report3 の指定時には、全ループのベクトル化に関する診断メッセージを出力します。メッセージの内容を確認することで、プログラムに対する手動でのベクトル化の検討（アルゴリズムの変更やコードの書き換え）も可能となります。