ホワイトペーパー

NVIDIA の次世代型 CUDA™ コンピュート・アーキテクチャ

Kepler™ GK110

史上最速・最高効率の *HPC* アーキテクチャ

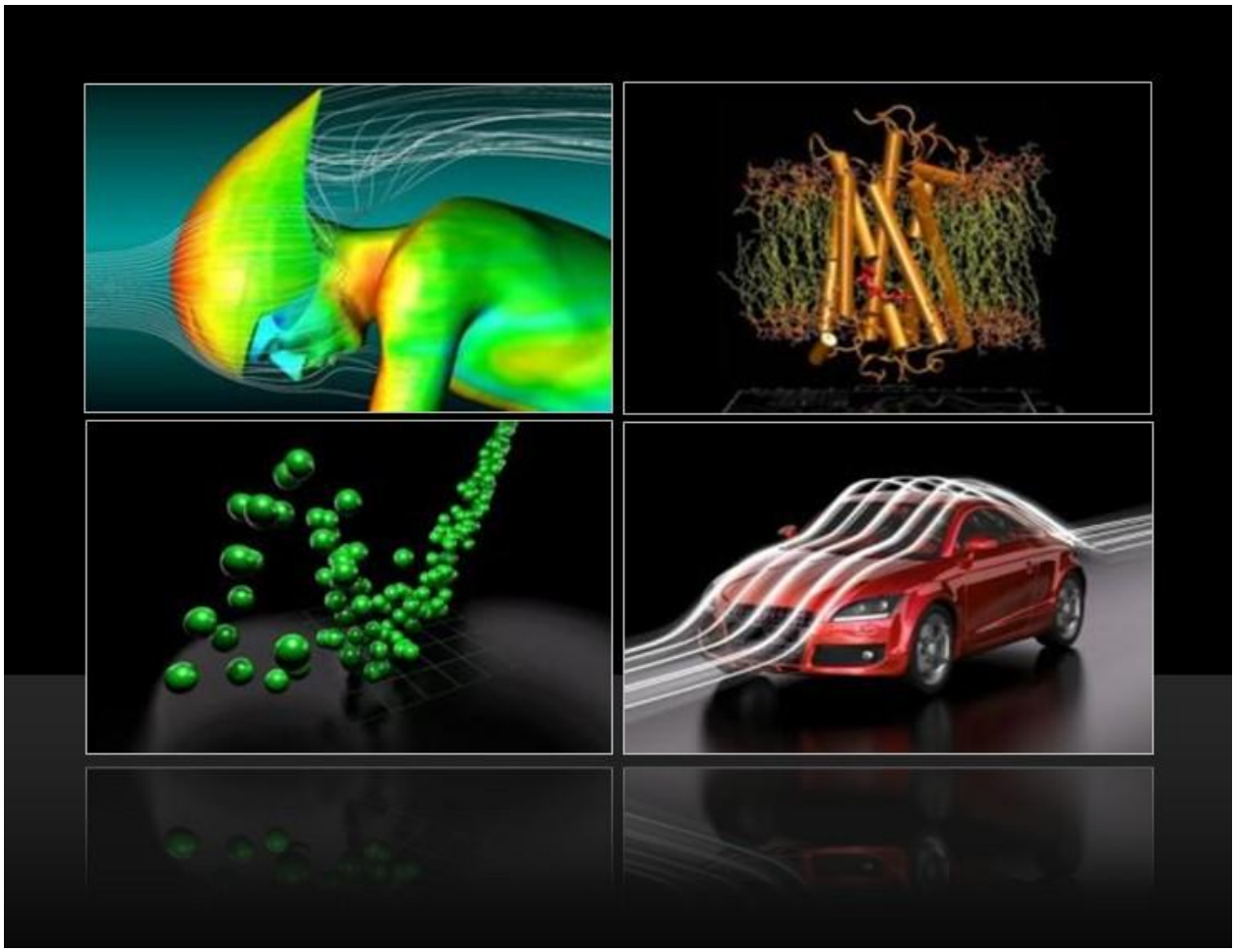
V1.0

目次

Kepler GK110 – 次世代型 GPU コンピューティング・アーキテクチャ

科学や医薬、工学、金融などさまざまな分野で、高性能な並列処理への要求が高まり続けていることをうけ、NVIDIA は技術革新を続け、卓越した GPU コンピューティング・アーキテクチャを世の中に送り出すことによって、それらのニーズにこたえています。既に NVIDIA の現行 Fermi GPU も、地質・地震データ処理や生化学関連のシミュレーション、気象・気候のモデリング、信号処理、金融工学、コンピューター支援エンジニアリング（CAE）、数値流体力学、データ解析などの分野で、ハイパフォーマンス・コンピューティング（HPC）の適用範囲を再定義し、その性能向上に貢献してきました。NVIDIA の新たなアーキテクチャ Kepler GK110 は、並列処理の水準をさらに大きく引きあげ、世界的に最も困難だとされているコンピューティング課題の解決を支援します。

これまでの GPU に比べて処理能力が格段に高いこと、また GPU による並列処理の実行を最適化し、実行効率を向上させる新しい方法が用意されていることから、Kepler GK110 では、並列プログラムの作成が容易になり、ハイパフォーマンス・コンピューティング技術に更なる革新をもたらすことができます。

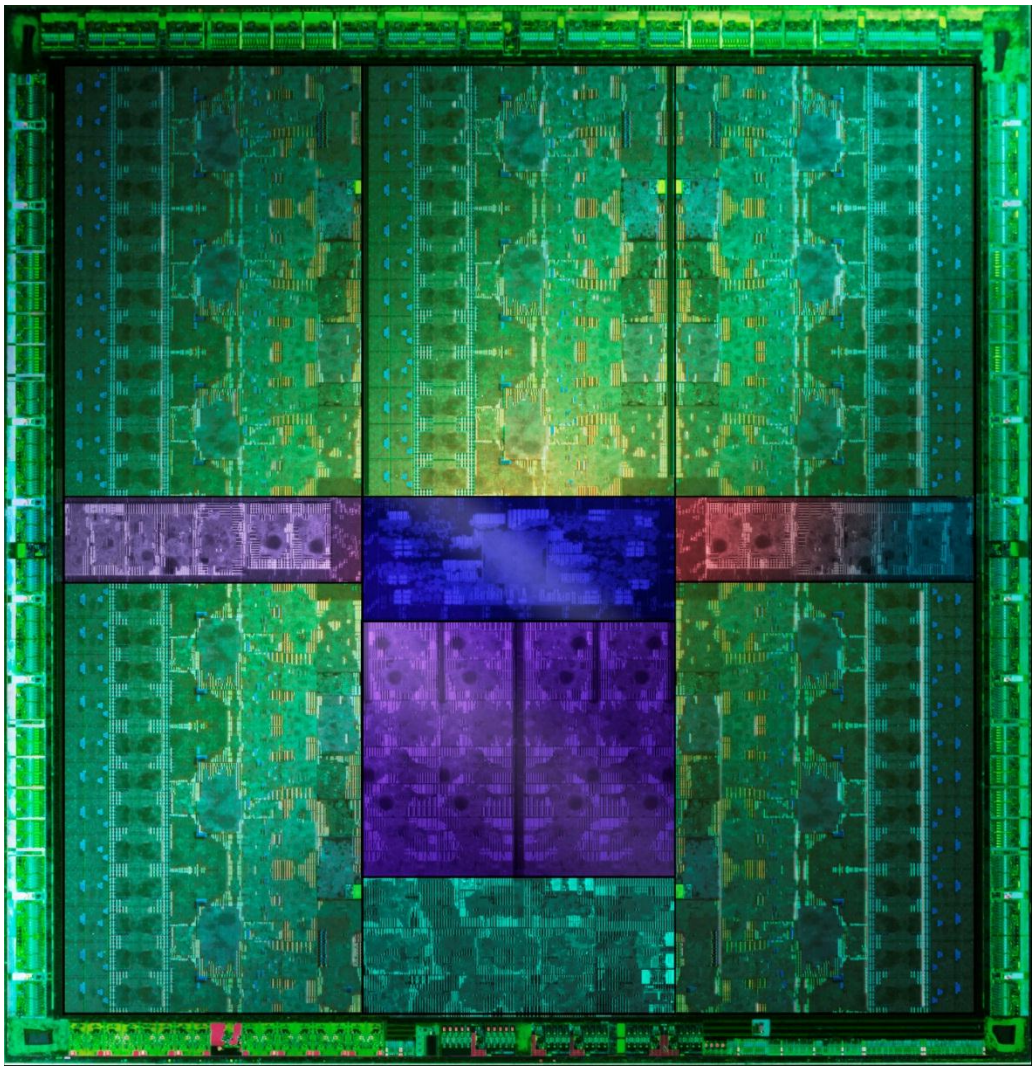


Kepler GK110 – 究極の性能と究極の効率

71 億個のトランジスターで構成される Kepler GK110 は、史上最速であるとともに、アーキテクチャ的に史上最も複雑なマイクロプロセッサーでもあります。計算処理性能に注力した革新的な新機能を数多く搭載した GK110 は、Tesla®や HPC 市場において並列処理の原動力になるように設計されているのです。

従来の Fermi アーキテクチャは DGEMM（倍精度行列乗算）効率が 60～65%でしたが、Kepler GK110 は 80%以上の効率を達成し、倍精度演算で 1 テラフロップス以上ものスループットを実現することができます。

パフォーマンスの大幅な向上だけではなく、Kepler アーキテクチャでは電力効率の面でも大きな飛躍があり、消費電力 1 ワットあたりのパフォーマンスが最大で Fermi の 3 倍になっています。



Kepler GK110 のチップ写真

Kepler GK110 の以下のような新機能により、GPU の使用効率の増加や並列プログラム開発の簡素化が達成され、パーソナル・ワークステーションからスーパーコンピュータに至る多様なコンピューティング環境での GPU の利用が容易になります。

- **ダイナミック並列処理** – CPU の関与無しで GPU 自身が自律的にワークを生成し、結果を同期し、作業スケジュールの制御を行う機能です。これらを専用の高速ハードウェアパスで行うことができます。プログラム実行中にダイナミックに並列処理の形式と並列度を最適化できる柔軟性が得られるため、計算処理の進行に合わせてプログラマーはより多様な並列処理手法を展開でき、GPU ハードウエアをより有効に活用できるようになります。この機能があれば、アプリケーションの大部分を GPU のみで実行しながら、あまり構造化されていない複雑なタスクを簡単かつ効率的に実行できるようになります。さらに、プログラムの開発も容易になりますし、他のタスクを実行できるように CPU を解放することもできます。

- **Hyper-Q** – Hyper-Q は、複数の CPU コアからひとつの GPU に同時にワークを生成できる技術で、GPU の利用率が劇的に向上するとともに CPU のアイドル時間を大幅に削減できます。この機能を活用すると、ホストと Kepler GK110 の間のコネクション（ワーク・キュー）の総数を増やすことができます。ハードウェア管理により 32 本のコネクションを同時に実現できるのです（Fermi の場合、コネクションはひとつだけでした）。Hyper-Q は、複数の CUDA ストリームから、複数の MPI（Message Passing Interface）プロセスから、あるいは、同一プロセスの複数のスレッドからでも個別にコネクションを実現できる柔軟なソリューションです。いままでは一部のアプリケーションにおいてタスク間で本来必要でないシリアル化がおき、GPU の利用率を高められないケースがありましたが、そのようなアプリケーションは、コードを変更することなく、劇的なパフォーマンスの改善が見込まれます。

- **グリッド管理ユニット** – 動的並列処理を実現するためには、グリッド管理とディスパッチを行う高度で柔軟な制御システムが必要になります。GK110 に新規に導入されたグリッド管理ユニット（GMU）は GPU で実行されるグリッドを管理し、優先順位をつけます。GMU は実行可能な状態になるまで新しいグリッドのディスパッチを中断し、実行待ちや中断中のグリッドをキューに保持できます。これによりダイナミック並列処理のようにパワフルなランタイム処理を実現することができます。GMU によって、CPU と GPU で生成されたワークがどちらも適切に管理・ディスパッチされるのです。

- **NVIDIA GPUDirect™** – NVIDIA GPUDirect™とは、1 台のコンピューターに搭載された複数の GPU 間、あるいは同一ネットワークに接続された複数サーバの GPU 間で CPU やシステムメモリーを介さず直接データを交換できる機能です。GPUDirect の RDMA 機能により、SSD や NIC、IB アダプターなどのサードパーティー機器が同一システムに搭載された複数 GPU のメモリーへ直接アクセスすることが可能になります。これにより GPU メモリとの MPI 送受信レイテンシーが大幅に低減されました。この機能にはまた、システムメモリーに対する必要バンド幅の引き下げや、GPU DMA エンジンのワークロードの他の CUDA タスクへの振り分けが行えるようになるなどのメリットもあります。Kepler GK110 は、ピア・ツー・ピアおよび GPUDirect for Video など従来の GPUDirect 機能もサポートしています。

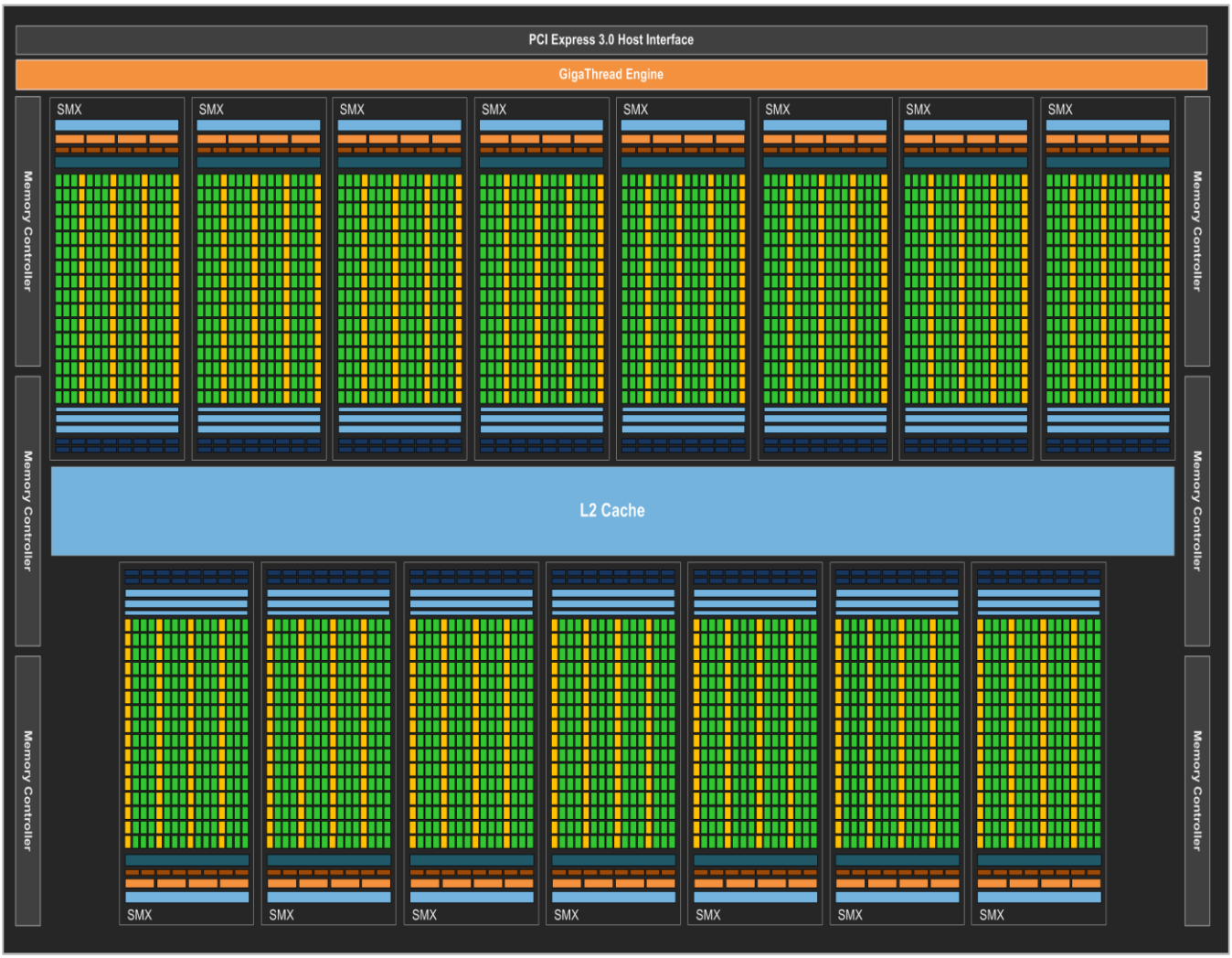
GK110 Kepler アーキテクチャの概要

Kepler GK110 はなんといっても Tesla 用に作られた製品であり、その目標は、世界最高のパフォーマンスを誇る並列処理マイクロプロセッサーとなることです。GK110 は Fermi の理論計算処理能力を大きく凌駕するだけでなく、効率も大幅に向上しており、消費電力も発熱量も大幅に削減されています。

フル実装の Kepler GK110 には、SMX ユニットが 15 個と 64 ビットのメモリー・コントローラが 6 個用意されます。ただし、GK110 の構成は製品によって異なります。たとえば、SMX を 13 個や 14 個にした製品もあり得ます。

下記リストにあげた本アーキテクチャの主要機能に関して詳しく説明していきます：

- 新 SMX プロセッサーアーキテクチャ
- 強化されたメモリーサブシステム　–キャッシング機能の追加、各メモリー階層でのバンド幅拡大、新規設計により大幅な高速化を実現した DRAM I/O アクセス
- 新しいプログラミングモデルを実現可能とするハードウェア仕様



Kepler GK110 チップの全体ブロックダイヤグラム

Kepler GK110 は、新しい CUDA Compute Capability 3.5 をサポートしています（CUDA の概要については、**付録 *A – CUDA* の簡単なおさらい**をご覧ください）。Fermi と Kepler の Compute Capability の主な違いは、次の表に示すとおりです。

	FERMI GF100	FERMI GF104	KEPLER GK104	KEPLER GK110
Compute Capability	2.0	2.1	3.0	3.5
スレッド / ワープ	32	32	32	32
ワープ / マルチプロセッサの最大値	48	48	64	64
スレッド / マルチプロセッサの最大値	1536	1536	2048	2048
スレッドブロック / マルチプロセッサの最大値	8	8	16	16
32 ビットレジスタ数 / マルチプロセッサ	32768	32768	65536	65536
レジスタ 数/ スレッドの最大値	63	63	63	255
スレッド / スレッドブロックの最大値	1024	1024	1024	1024
共有メモリーサイズのコンフィギュレーション（バイト）	16K 48K	16K 48K	16K 32K 48K	16K 32K 48K
Max X Grid Dimension	2^16-1	2^16-1	2^32-1	2^32-1
Hyper-Q	NO	NO	NO	YES
ダイナミック並列処理	NO	NO	NO	YES

Fermi の GPU と Kepler GPU の Compute Capability

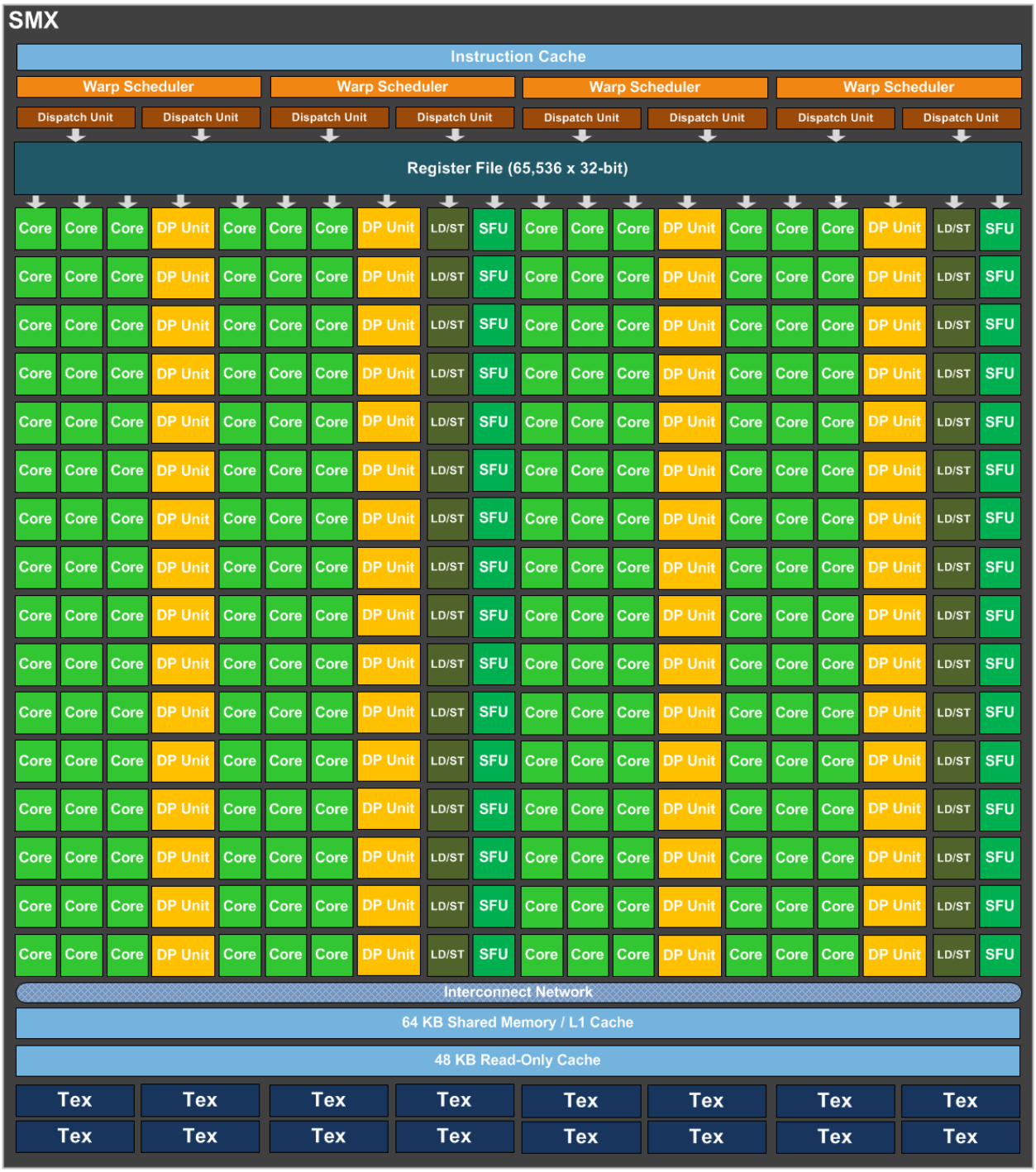
消費電力(ワット)あたりのパフォーマンス

電力効率の向上が Kepler アーキテクチャの最も大切な設計目標でありました。高効率的な処理が実現できるように、Fermi で学んだ全ての技術、経験を投入して Kepler アーキテクチャの最適化を行いました。消費電力の低減には TSMC の 28nm 製造プロセスが重要な役割を果たしていますが、優れたパフォーマンスを保ちつつ消費電力をさらに削減するためには、GPU アーキテクチャもさまざまな面で改良する必要がありました。

Kepler の全てのハードウェア・ユニットは、ワットあたり最高のパフォーマンスを出せるように設計され磨き抜かれています。優れたパフォーマンス/ワットの最も良い設計例は、Kepler GK110 に新しく搭載されたストリーミング・マルチプロセッサー（SMX）です。この SMX は、先に Kepler GK104 に導入された SMX ユニットといろいろな面で類似点がありますが、演算アルゴリズムで多用される倍精度浮動小数点演算ユニットの数が大幅に増えています。

ストリーミング・マルチプロセッサー（SMX）アーキテクチャ

Kepler GK110 に新しく搭載された SMX には、アーキテクチャレベルのイノベーションが数多く採用されています。その結果、NVIDIA マルチプロセッサとして過去最高の性能を達成したばかりでなく、プログラミング性と電力効率の面でも過去最高の製品になりました。



SMX：単精度 CUDA コア 192 個、倍精度ユニット 64 個、特殊関数ユニット（SFU）32 個、ロード/ストア・ユニット（LD/ST）32 個

SMX プロセッシング・コアアーキテクチャ

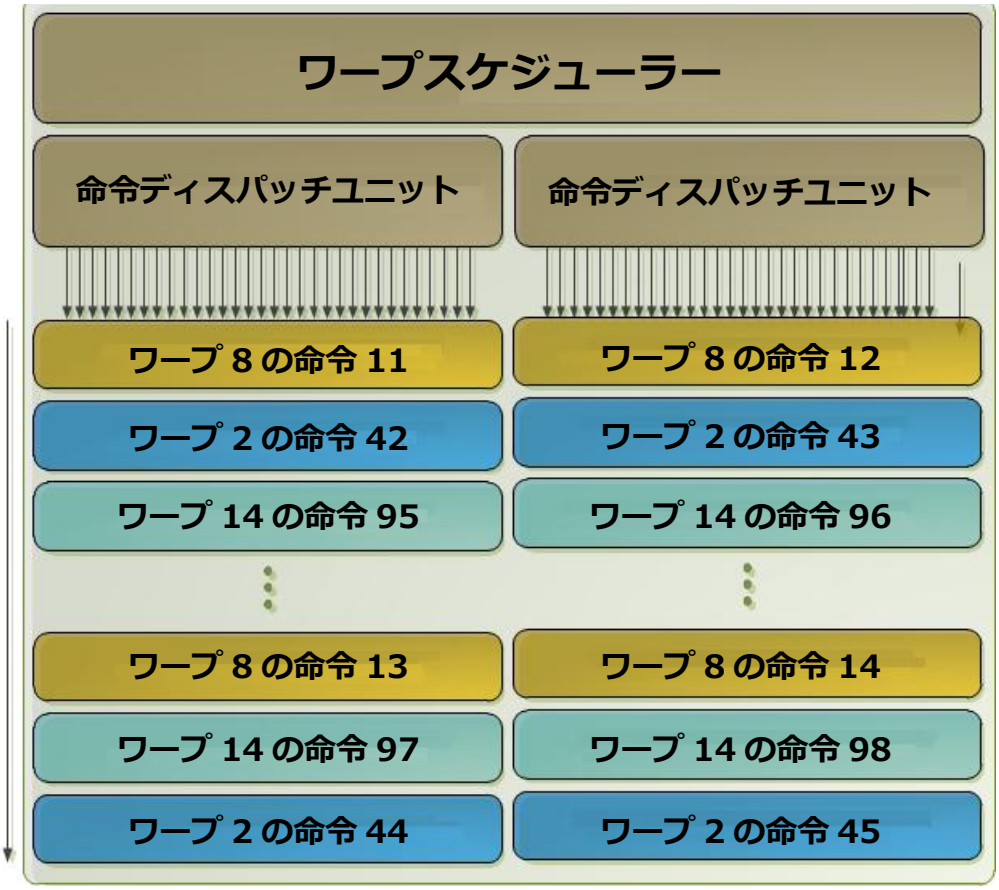
Kepler GK110 の SMX ユニットには、単精度 CUDA コアが 192 個搭載されていて、各コアには完全パイプライン化された浮動小数点演算ユニットと整数演算ユニットが設けられています。FMA（Fused Multiply-Add）命令も含め、Kepler も Fermi と同様に、単精度算術演算と倍精度算術演算がともに **IEEE 754-2008 完全準拠**となっています。

Kepler GK110 の SMX は、GPU の倍精度パフォーマンスを大幅に高めることを目標の一つとして設計されました。これは倍精度算術演算が HPC アプリケーションの要になるからです。Kepler GK110 の SMX にはまた、旧世代 GPU と同じように、超越関数命令（三角関数、対数関数等）を高速で近似計算できる特殊関数ユニット（SFU）が用意されていますが、その数は、Fermi GF110 SM の 8 倍に達します。

GK104 SMX ユニットと同じように GK110 SMX ユニットも、コアはシェーダ・クロックの 2 倍ではなくプライマリ GPU クロックで動作します。シェーダ・クロックの 2 倍動作は、G80 Tesla アーキテクチャの GPU で導入され、その後の Tesla アーキテクチャおよび Fermi アーキテクチャの GPU、すべてに採用された仕組みです。高い周波数で演算ユニットを動作させると少ない演算ユニット数で目標とするスループットを実現できます（これは面積の最適化だと言えます）。ただし、高速コアのクロック同期回路は電力を多く消費するという問題があります。一方、Kepler においては、ワットあたりのパフォーマンスが優先でした。面積と消費電力の両方にメリットのある最適化を数多く試みましたが、最終的に SMX では、GPU クロックを消費電力が少ない低い周波数にして演算コアの数を増やすという形で、面積コストが増えても消費電力の最適化を優先する選択をしたわけです。

クワッド・ワープスケジューラ

SMX では、32 本の並列スレッドをグループ化したワープを単位にスレッドのスケジューリングを行います。各 SMX にはワープスケジューラが 4 個と命令ディスパッチ・ユニットが 8 個あり、4 つのワープを並列に発行・実行することができます。Kepler のクワッド・ワープスケジューラは、4 つのワープを選択し、1 ワープにつき 1 サイクルに独立した命令を 2 つ発行できるのです。Fermi の場合、倍精度命令を他の命令と組み合わせることはできませんでしたが、Kepler GK110 では、ロード/ストア命令、テクスチャー命令、一部の整数命令など、レジスタファイルのリードがない命令とであれば倍精度命令をペアにすることができます。



Kepler SMX ひとつにワープスケジューラーが 4 個あり、そのそれぞれに命令ディスパッチユニットがふたつ用意されています。この図はワープスケジューラー・ユニットひとつを示したものです。

SMX ワープスケジューラのロジックについても、消費電力の最適化を行いました。たとえば以下のように、Kepler も Fermi もスケジューリング機能を処理するスケジューラのハードウェア・ユニットはよく似ています。

a) レイテンシが大きい操作（テクスチャーやロード）のレジスタ・スコアボーディング
b) ワープ間のスケジューリングに関する決定（実行可能なワープから、次に行うべきベストなワープを選ぶなど）
c) スレッドブロックレベルのスケジューリング（GigaThread エンジンなど）

Fermi のスケジューラには、このほかに、計算データパス自体でデータハザードが起きないようにする複雑なハードウェアステージが用意されていました。マルチポートのレジスタスコアボードがまだ有効なデータの準備が整っていないレジスタの履歴を保持するとともに、完全にデコードされた複数のワープ命令に対してデータ依存性をチェックするブロックがレジスタの使用状況を分析し、次に実行すべき命令を判断しました。

Kepler の開発においては、この情報は確定的であり（計算処理パイプラインのレイテンシは確定値である）、命令をいつ発行すればいいのかをコンパイラ側で事前に求め、その情報を命令自体に組み込むことが可能であることが分かりました。。その結果、いくつかの複雑で消費電力の大きなブロックをシンプルなハードウエアブロックで置き換えることが可能となりました。このハードウエアブロックでは、事前に求められ、命令に組み込まれたレイ

テンシ情報を抽出し、その情報を活用して、ワープ間スケジューラ・ステージにおいて適格なワープの実行順序を決めています。

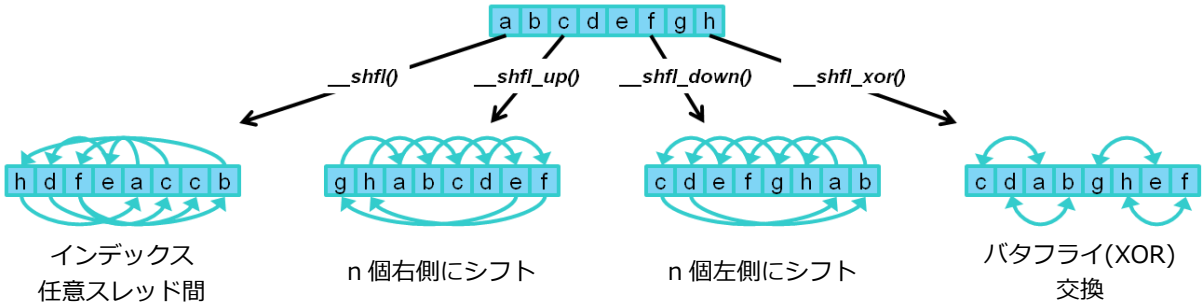
新しい ISA エンコーディング：255 レジスタ／スレッド

GK110 では、1 スレッドからアクセスできるレジスタの数が最大で 255 レジスタと 4 倍に増強されました。1 スレッドあたりで利用できるレジスタの数が増えた結果、Fermi では多くのレジスタを使用したり、使用可能レジスタ数を超えてしまったコードは Kepler ではこの機能によって大きくスピードアップする可能性があります。この効果がはっきりと感じられる例が QUDA ライブラリーにあります。CUDA を使った格子 QCD（量子色力学）の計算で、QUDA fp64 をベースとしたアルゴリズムではパフォーマンスが最大で 5.3 倍にも向上します。1 スレッドあたりで利用できるレジスタの数が増え、ローカルメモリーへのデータの退避が減るからです。

シャッフル命令

パフォーマンスをさらに高めるため、Kepler では、ひとつのワープに属する複数のスレッドでデータを共有できるシャッフル命令が新しく用意されました。いままで、ひとつのワープに属する複数スレッドでデータを共有するためには、ストア操作とロード操作を行って共有メモリー経由でデータを受けわたす必要がありました。新しく用意されたシャッフル命令では、ひとつのワープに属するスレッドは、同じワープの他のスレッドから好きな順番で値を読むことができます。シャッフルのパターンとしては、まず、任意のインデックスによる参照（どのスレッドも他のどのスレッドからも読める）がサポートされています。このほか、ネクストスレッド・シャッフル（一定オフセットの前後スレッド移動）や同一ワープ内のスレッドを「バタフライ」スタイルで入れ替える XOR シャッフルが CUDA の組み込み関数として用意されています。

シャッフルでは、ストアからロードまでの操作が 1 ステップで実行されるため、共有メモリー利用の場合に比較して、大幅にパフォーマンスが向上します。また、スレッドブロックあたりに必要とする共有メモリーの量も少なくなります。ワープレベルのデータ交換で共有メモリーを使う必要がなくなるからです。高速フーリエ変換(FFT)処理を行う場合にはワープ内でデータ共有が必要になるため、シャッフル機能を使うだけでパフォーマンスが 6%も向上します。



Kepler で新しく実装されたシャッフル命令を使えば実現できるパターンの一部

アトミック操作

並列プログラミングにおいては、同時並行で処理を進めるスレッドが共有データリソースに対して読込み-変更-書込みの一連の処理を正しく行うためには、アトミックなメモリー操作が重要となります。加算、最小値、最大値、比較-入換えなどのアトミック演算処理は、他のスレッドから割込まれることなくデータの読出しから変更、書込みまでの一連の処理が完了できなければならないため、アトミック演算と呼ばれます。アトミックなメモリー操作は、並列ソート、リダクション演算のほか、シリアライズ・スレッド実行をロックすることなくデータ構造を並列に構築する場合など、さまざまな場面で利用されます。

Kepler GK110 では、グローバルメモリーのアトミック操作スループットが Fermi に比べて大幅に改善されました。共通グローバルメモリーアドレスに対するアトミック操作のスループットは、1 クロックあたり 1 操作と 9 倍にも高速化されています。独立したグローバルアドレスに対するアトミック操作のスループットも大幅に高速化され、アドレスの競合を処理するロジックの効率も改善されました。Kepler では、多くの場合、グローバルなロード操作と同じくらいのスピードでアトミック操作が行えます。スピードがこれほど改善された結果、カーネルの内部ループでもアトミック操作を頻繁に使うことが可能になり、いままでアルゴリズムによっては最終結果の算出に必要だった個別のリダクションパスが不要になりました。Kepler GK110 では、グローバルメモリーにおける 64 ビットのアトミック操作もネイティブサポートされました。Fermi や Kepler GK104 でもサポートされていた atomicAdd、atomicCAS、atomicExch のほかに、GK110 では、以下の操作がサポートされました。

- atomicMin
- atomicMax
- atomicAnd
- atomicOr
- atomicXor

ネイティブでサポートされていない 64 ビット浮動小数点アトミックのような操作は、比較-入換え（CAS：Compare And Swap）命令などを使ってエミュレーションできます。

テクスチャーの改良

GPU が持つテクスチャーユニットという専用ハードウェアは、画像データのフィルタリングやサンプリングを行う計算プログラムにおいて大きな役割を果たします。Kepler では、テクスチャー・スループットも Fermi に比べて大幅に高められました。Kepler の SMX ユニット 1 個には、Fermi GF110 SM に対して 4 倍となる 16 個ものテクスチャー・フィルタリング・ユニットが用意されています。

また、Kepler では、テクスチャーの状態を管理する方法も改善されました。Fermi 世代の GPU でテクスチャーを参照する場合、グリッド起動前に、固定サイズのバインディングテーブルに「スロット」を割り当てる必要がありました。このテーブルに用意されたスロットの数が、ランタイムにプログラムが読めるテクスチャーの数の上限を規定することになります。よって、Fermi の場合、アクセスできるテクスチャーに 128 個という上限があったわけです。

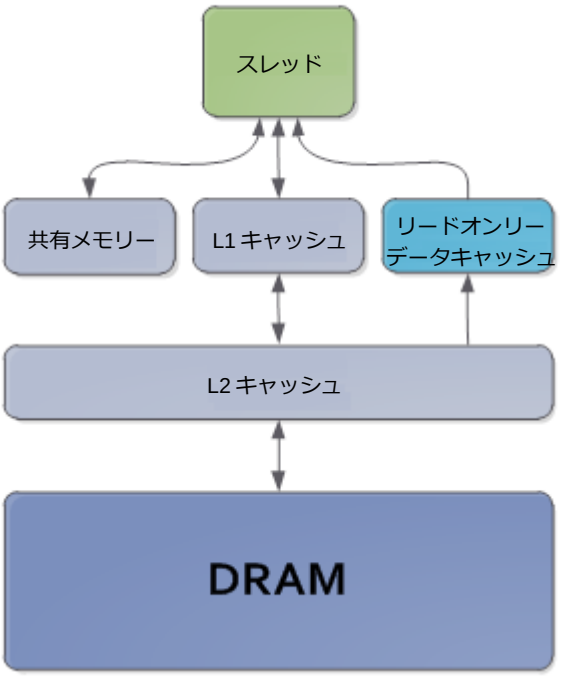
Kepler はバインドレス・テクスチャーとなったため、スロットを使うという面倒なステップは不要になりました。テクスチャーステートはメモリーのオブジェクトとして保存でき

ますし、このステート・オブジェクトは、オンデマンドにハードウェアから取得できます。この結果、プログラムが参照できるテクスチャーの数に上限はなくなりました。プログラムはいつでもテクスチャーをマッピングし、ごく普通のポインタとしてテクスチャーハンドルをやりとりすることが可能になったのです。

Kepler メモリーサブシステム－L1、L2、ECC

Kepler のメモリー階層は Fermi と同じような形で構成に整理されています。Kepler のアーキテクチャでは、各 SMX マルチプロセッサーごとの L1 キャッシュによって、ロードとストアの両方をカバーする形でメモリーリクエストの統合パスを実装しました。Kepler GK110 はまた、以下に説明するように、リードオンリーのデータ用に新たなキャッシュが設けられコンパイラからの指令で利用できるようになっています。

Kepler のメモリー階層



共有メモリー/L1 キャッシュに構成可能な 64KB メモリー

Kepler GK110 アーキテクチャは、旧世代の Fermi アーキテクチャと同じように、各 SMX に 64KB のオンチップメモリーが用意されており、これを 48KB の共有メモリーと 16KB の L1 キャッシュ、あるいは、16KB の共有メモリーと 48KB の L1 キャッシュに構成することができます。Kepler は、このほか、メモリーを 32KB/32KB という形で共有メモリーと L1 キャッシュに振りわける構成も可能で、柔軟性がさらに高くなりました。また、SMX ユニット単位のスループットを高めるため、64b 以上のロード操作で使用する共有メモリーのバンド幅も、Fermi SM から倍増され、1 コア・クロックごとに 256B となりました。

リードオンリーの 48KB データキャッシュ

L1 キャッシュ以外にも、Kepler には、ある関数を処理する間リードオンリーだとわかっているデータに使う 48KB のデータキャッシュが用意されています。Fermi の場合、このキャッシュはテクスチャーユニットからしか読むことができませんでした。熟練プログラマーのなかには、データをテクスチャーとしてマッピングし、このパスでデータをロードするという高等技術を使う人が多くいましたが、この方法にはさまざまな面で制約がありました。

Kepler は、テクスチャー処理能力の増強に伴いこのキャッシュの容量を大幅に増やすとともに、一般的なロード操作でもこのキャッシュに SM から直接アクセスできるようにしました。リードオンリーのパスを使うと、共有/L1 キャッシュのパスから負荷もメモリ使用領域も軽減できるというメリットがあります。さらに、リードオンリーのデータキャッシュはタグの帯域幅が広く、アラインメントされていないメモリーアクセスがフルスピードで行えるなどのメリットもあります。

このパスの利用はコンパイラ側で自動的に管理されます。つまり、C99 標準の“const __restrict”キーワードをプログラマーが指定し、特定の値であるとわかっている変数やデータ構造にアクセスする場合、リードオンリー・データキャッシュ経由でロードするようにコンパイラがタグをつけるのです。

改良された L2 キャッシュ

Kepler GK110 GPU には、Fermi アーキテクチャの倍にあたる 1536KB の専用 L2 キャッシュが用意されています。L2 キャッシュは SMX ユニット間でデータを統合する要のポイントです。全てのロード命令、ストア命令、テクスチャリクエスト命令をサポートし、GPU 内でのデータ共有を高速かつ効率的に実現します。Kepler の L2 キャッシュは Fermi に対してクロックあたりのバンド幅が最大で 2 倍になっています。このキャッシュ階層は、物理演算ソルバー、レイトレーシング、疎行列乗算など、データのアドレスがあらかじめわかっていないアルゴリズムに大きなメリットをもたらします。複数の SMX が同じデータを読まなければならないフィルターカーネルやコンボリューションカーネルにもメリットがあります。

メモリー保護のサポート

Fermi と同じように Kepler も、レジスタファイル、共有メモリー、L1 キャッシュ、L2 キャッシュ、DRAM メモリーに対して、SECDED（Single-Error Correct, Double-Error Detect）ECC コードによる保護が施されています。さらに、リードオンリーのデータキャッシュについては、パリティチェックによる単一誤り訂正がサポートされています。パリティエラーが発生すると、障害が発生したラインをキャッシュユニットが自動的に無効化し、L2 キャッシュから正しい値を読みなおします。

DRAM から ECC のチェックビットを読み出さなければならないということは、DRAM バンド幅を消費するということなので、ECC を有効にした場合と無効にした場合で操作のパフォーマンスは異なります。メモリーバンド幅の影響を受けやすいアプリケーションでは、この違いに注意を払う必要があります。Kepler GK110 では、Fermi の経験を生かし、ECC チェックビットの読み出しにさまざまな最適化を施しました。その結果、ECC をオンにした場合とオフにした場合のパフォーマンスの差を平均で 66%も削減することに成功しました（NVIDIA が社内で使用しているコンピュートアプリケーション用テストスイートを使った測定による数値です）。

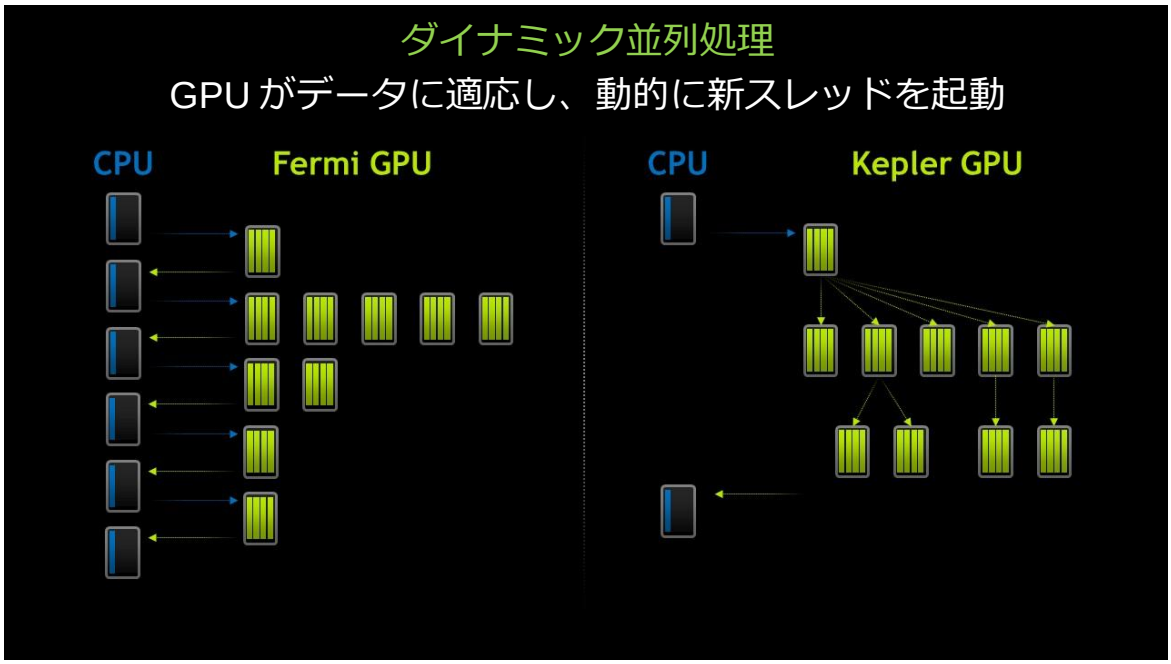
ダイナミック並列処理

ハイブリッド型の CPU-GPU システムでは、アプリケーションに含まれる膨大な並列処理コードを GPU 内のみで効率的に実行させると、GPU の高いパフォーマンス/ワット性能により、アプリケーション全体のスケーラビリティともパフォーマンスを向上させることができます。より多くの並列処理部分を高速化するため、GPU は、従来以上に多様な並列ワークロードをサポートする必要があります。

そのために Kepler GK110 へ新しく導入されたのがダイナミック並列処理で、この機能を使うと、GPU が処理するワークを GPU 自身が生成し、結果を同期し、そのワークのスケジューリングを制御することが可能になり、CPU の関与無しに、これらの一連の作業を GPU 内の専用の高速ハードウェアパスで行うことができます。

Fermi アーキテクチャでは、カーネル起動時にタスクのスケールとパラメータが予めわかっていれば、大きな並列データ構造を効率的に処理することができました。すべてのワークはホスト CPU から起動され、GPU 側の処理が完了したら、その結果がホスト CPU に戻されます。戻された結果は、最終的な結論またはその一部として解に使われるか、あるいは、CPU がその結果を解析し、必要があれば追加処理のリクエストとともに再度 GPU に送られることになります。

Kepler GK110 では、カーネルから別カーネルを起動することがで、必要なストリームやイベントを生成したり、追加ワークの処理に必要な依存関係を管理することがホスト CPU の関与なしに実行できます。このような革新的なアーキテクチャの採用により、GPU 上で再帰的な実行パターンやデータ依存の実行パターンを生成・最適化することが可能になり、開発者は今まで以上に多くの種類のプログラムを GPU 上で容易に実行することができるようになりました。この結果、システムの CPU のワークロードを別のタスクに振り分けたり、よりパフォーマンスの低い CPU を搭載したシステムで同じワークロードを処理することも出来るようになりました。



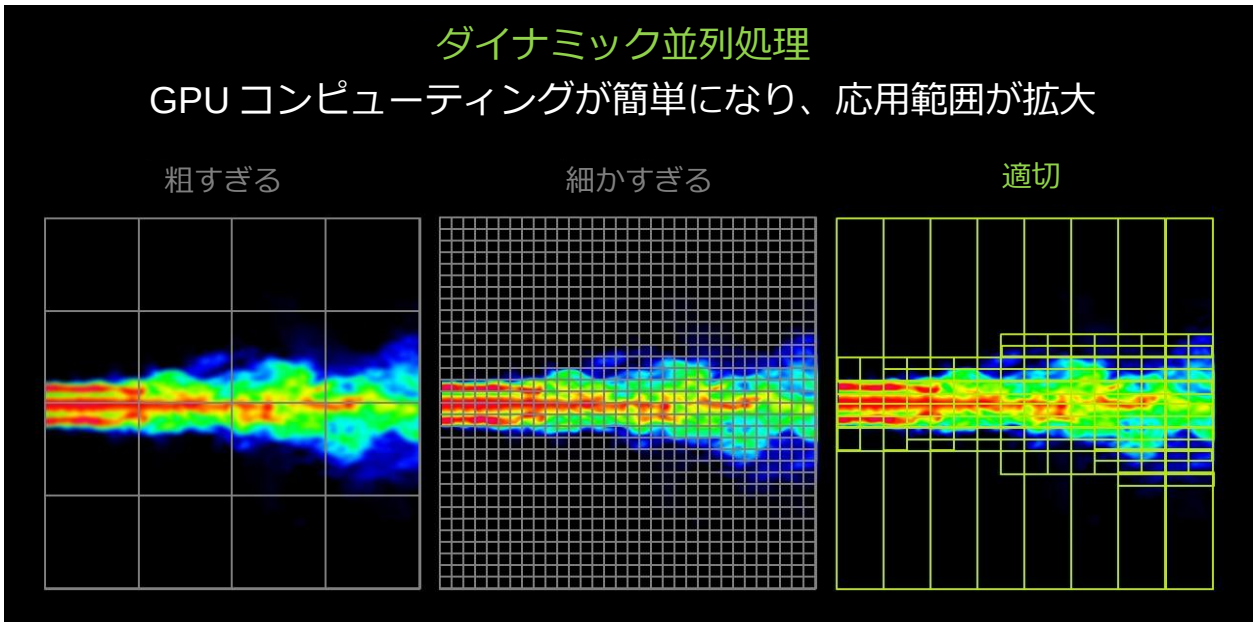
ダイナミック並列処理では GPU が GPU 自身によって処理されるコードを生成することが可能で（右側）、CPU の介入が必要な場合（左側）に比べてコードの並列度をより高めることができます。

ダイナミック並列処理を使うと、並列度が異なる入れ子構造のループ、複数の逐次処理タスクスレッドの並列処理、または単純な逐次処理コードを GPU に割り当てて、アプリケーションの並列部分とデータの局所性を共有することもできます。

GPU 側の中間結果に基づいてカーネルが新たなワークを生成できるということは、GPU のコード上で自律的にワークの負荷バランスを制御できるということで、プログラマーは、高い処理能力が要求される部分や、最終結果の導出に最も関係が深い部分にプロセッシング能力の大半を振り向けることができます。

ひとつの例として、数値シミュレーションのグリッドを状況に合わせて動的に設定するということが考えられます。通常グリッドセルの密度は、変化が最も激しい部分に集中させる必要がありますが、そのためには、多大な処理能力を費やしてデータを事前に分析する必要があります。他の方法として、GPU リソースの浪費を防ぐため全体に粗いグリッドを設定する方法、あるいは、すべての変化を把握できるよう全体に細かいグリッドを設定する方法もありますが、この場合、シミュレーション性能が低下するリスクや、細かいところまで知る必要のない領域に処理能力を「浪費」するリスクが発生します。

ダイナミック並列処理では、このグリッド解像度を中間データに依存する形でランタイムに動的に決めることができます。粗いグリッドでシミュレーションをスタートし、細かなデータが必要な部分のみ「ズームイン」するという形になるので、変化が少ない領域で無駄な計算をする必要がありません。もちろん、同じことは CPU から生成するカーネルのシーケンスでも行えますが、ひとつのシミュレーションカーネルとして、GPU 側でデータを解析し、必要な追加ワークを GPU 自身が起動するという形でグリッドを細密化して行く方がずっとシンプルになります。CPU の関与も不要になり、CPU と GPU の間でデータを送受信する必要もなくなります。



– 画像はチャールズ・レイド氏提供

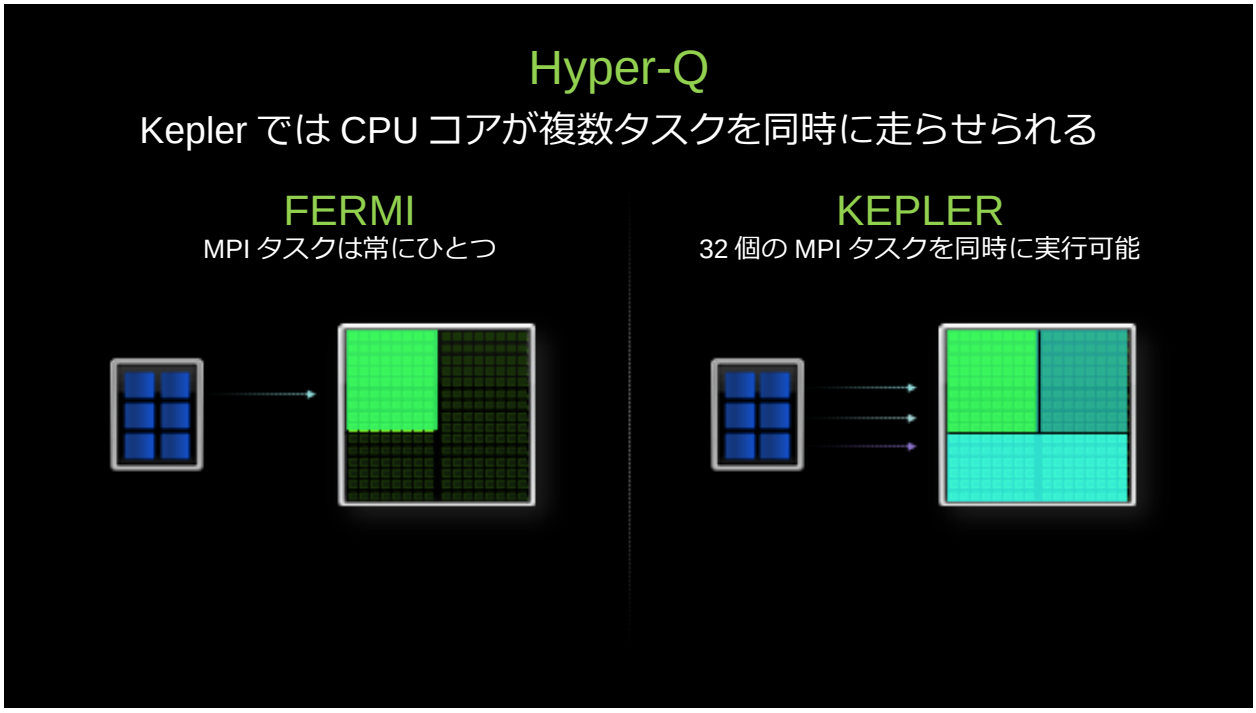
上図は、グリッドサイズの動的変更を数値シミュレーションに応用した場合の例です。固定グリッドサイズのシミュレーションでは、ピーク精度を満足するため、シミュレーション対象範囲の全体にわたって細かすぎるほどの解像度でシミュレーションを行う必要があり

ます（中央図）。これに対して複数解像度のグリッドでは、局所的な変動に応じた適切な解像度でシミュレーションを行うことができます〔右図〕。

Hyper-Q

いままで大きな課題の一つとなっていたのが、複数のストリームからいかにして適切なスケジュールでワークロードを GPU に供給し続けるかでした。Fermi アーキテクチャでは、個別ストリームから 16 個のカーネルを起動できる並列性が確保されていましたが、最終的にはこれらのストリームはすべてひとつのハードウェア・ワークキューへと集約されていました。このため、現行ストリーム内の依存性のあるカーネルを全て完了してからでないと、別ストリーム内の新規カーネルが実行できないといった、本来必要でないストリーム間の依存性が発生してしまいます。起動順を並列幅優先とすれば、この問題をある程度は緩和できますが、プログラムが複雑になると、この部分の管理を効率的に行うことはどんどん難しくなってしまいます。

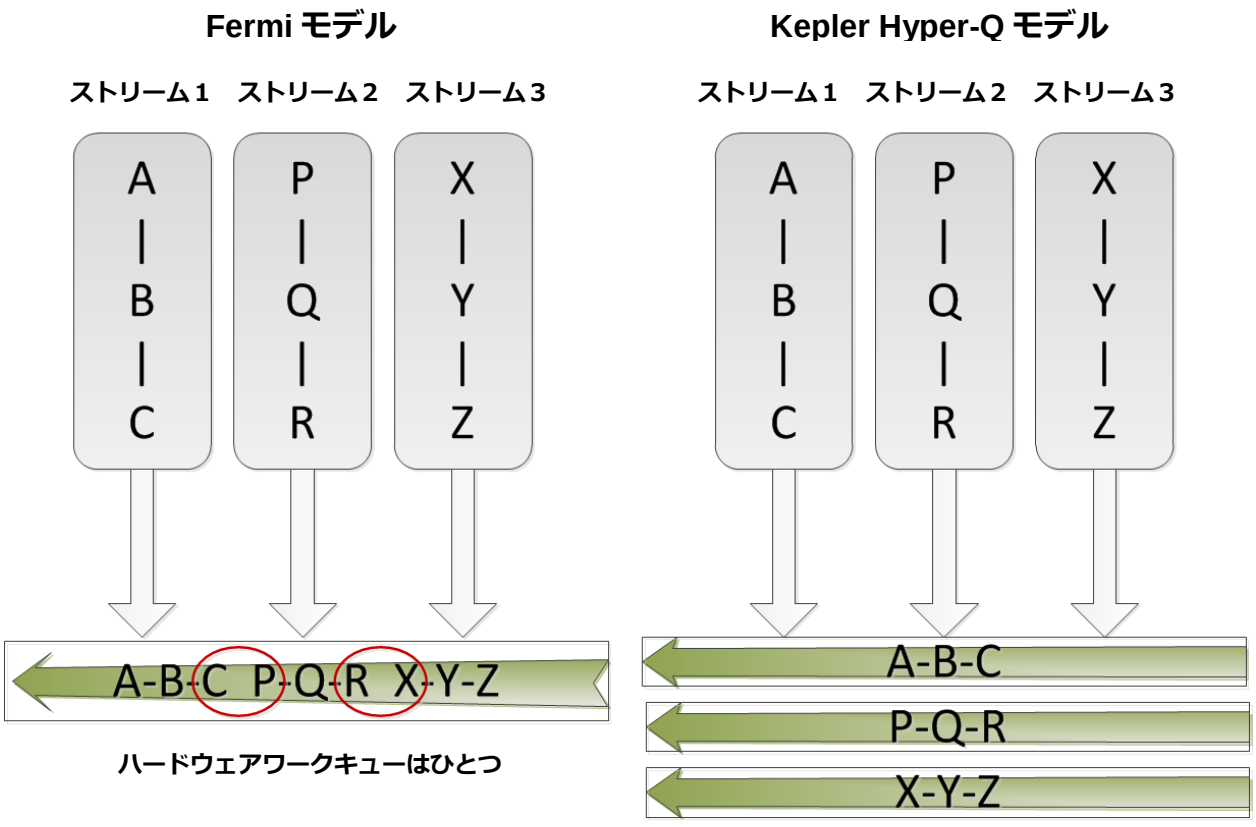
Kepler GK110 では、Hyper-Q という機能を導入し、この問題に対処しています。Hyper-Q は、ホストと GPU 側の CWD（CUDA Work Distributor）ロジックをつなぐコネクション（ワークキュー）の総数を増やし、ハードウェア管理によるコネクションを同時に 32 本、実現できるようにしました（Fermi の場合、コネクションはひとつだけでした）。Hyper-Q は、複数の CUDA ストリームや、複数の MPI（Message Passing Interface）から、あるいはまた、同じプロセスに属する複数のスレッドからでも個別にコネクションを実現できる柔軟なソリューションです。今までは一部のアプリケーションにおいて複数タスクをまたぐ形で本来必要ではないシリアル化が発生し、GPU の利用率を高められないケースがありましたが、そのようなアプリケーションは、コードを変更することなく、最大で 32 倍ものパフォーマンス向上が見込まれます。



Hyper-Q では、CPU・GPU 間の同時コネクション数が多くなりました

各ストリームはそれぞれのハードウェア・ワークキューにおいて管理されるので、ストリーム間の依存性が最適化され、あるストリームにおける操作が他のストリームをブロックすることがなくなります。この結果、本来必要でない依存性を防止するために起動順を調整する必要がなくなり、ストリームを同時並行に実行できるようになるのです。

Hyper-Q は、MPI ベースの並列コンピューターシステムで大きな効果を発揮します。レガシーな MPI ベースのアルゴリズムはマルチコア CPU システム用のものが多く、各 MPI プロセスに割り当てられるワーク量もその前提に依存します。この場合、ひとつの MPI プロセスでは GPU に対するワーク量が不足することが多く、GPU をフル稼働させることができません。もちろん、複数の MPI プロセスにひとつの GPU を共有させることは可能ですが、このような処理を行うと不必要な依存性によるボトルネックが発生しがちです。Hyper-Q はこの不必要な依存性をなくし、複数の MPI プロセスによる GPU 共有の効率を劇的に高めることができるのです。



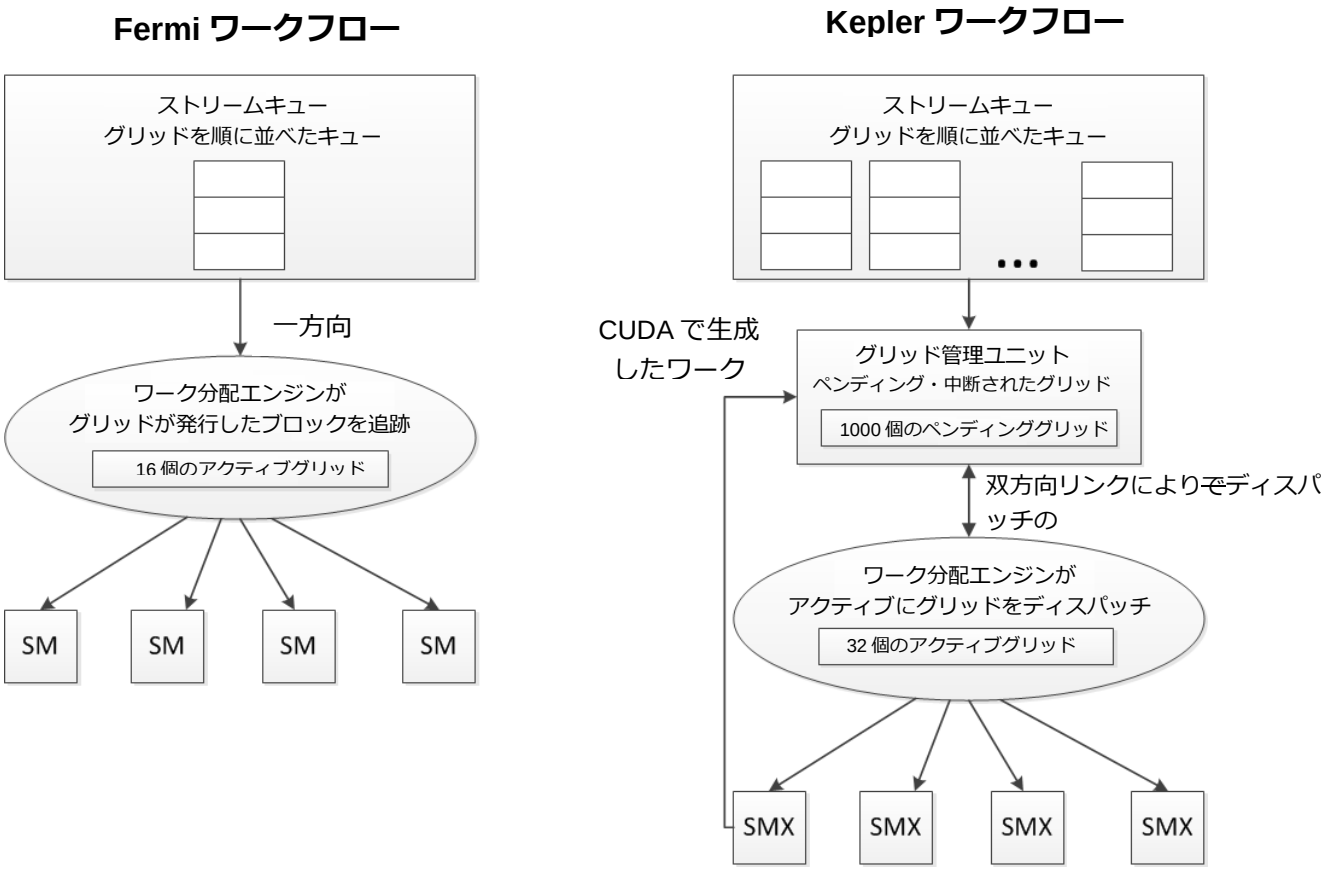
Hyper-Q と CUDA ストリームの組み合わせ – 左に示す Fermi モデルでは、ハードウェア・ワークキューがひとつでストリーム間依存性が発生するため、(C,P)と(R,X)の組しか並列実行できません。これに対して Kepler の Hyper-Q モデルでは、複数の作業キューを使ってすべてのストリームを並列実行することができます。

グリッド管理ユニット – 効率よく GPU の利用率を高いレベルで維持

GPU 上の CUDA カーネルから GPU に対する新たなワークを直接起動するダイナミック並列処理など、Kepler GK110 の新機能を実現するためには、Kepler における CPU から GPU へのワークフローは Fermi と比べてより高機能である必要があります。Fermi の場合、スレッドブロックのグリッドは CPU によって起動され、必ず完了まで実行されることになります。つまり、ホストから SM へ、CWD（CUDA Work Distributor）ユニット経由で一方向にワークが流れるシンプルな形になります。これに対し、CPU が生成したワークロードも CUDA カーネルが生成したワークロードも GPU が効率的に管理し、CPU から GPU へのワークフローを改善しようというのが Kepler GK110 のデザインです。

ここまで、Kepler GK110 GPU で新しく採用された、カーネルから GPU が行うワークを直接起動できる機能について説明してきましたが、そのような機能を可能にするために、Kepler GK110 のアーキテクチャにどのような変更が加えられたのかをも理解しておくことも重要です。Kepler も、Fermi と同じように CPU からグリッドを起動することができますが、それだけではなく、Kepler SMX ユニット内の CUDA カーネルからプログラムに従って新たなグリッドを生成することもできるのです。CUDA カーネルが生成したグリッドとホストが生成したグリッドの両方を管理するため、Kepler GK110 には新しいグリッド管理ユニット（GMU）が搭載されました。この制御ユニットがグリッドの管理と優先順位付けを行い、その順番に従って各グリッドは CWD 経由で SMX ユニットに渡され実行されます。

CWD には、ディスパッチの準備が整ったグリッドが保持されます。なお、Kepler の場合ディスパッチ可能なアクティブグリッドの数は全部で 32 個と、Fermi CWD から倍増されました。Kepler CWD は GMU と双方向のコミュニケーションが可能なため、GMU 側から新しいグリッドのディスパッチを一時停止したり、ペンディングされたグリッドや中断されたグリッドを必要になるまで保持したりすることができます。GMU は Kepler SMX ユニットとも直接接続されているため、ダイナミック並列処理を使って GPU 上に追加のワークを起動するグリッドが、新しいワークを GMU へいったん戻し、優先順位をつけてディスパッチさせるといったことが可能です。追加のワークロードをディスパッチしたカーネルが休止された場合、依存関係にあるワークが完了するまで GMU がその追加ワークロードを非アクティブ状態に保ちます。



Kepler ではホストから GPU へのワークフローが再設計され、新しいグリッド管理ユニットが用意されました。このグリッド管理ユニットにより、ディスパッチを行うグリッドの管理、ディスパッチの休止、ペンディングされたグリッドや中断されたグリッドの保持などを行うことができます。

NVIDIA GPUDirect™

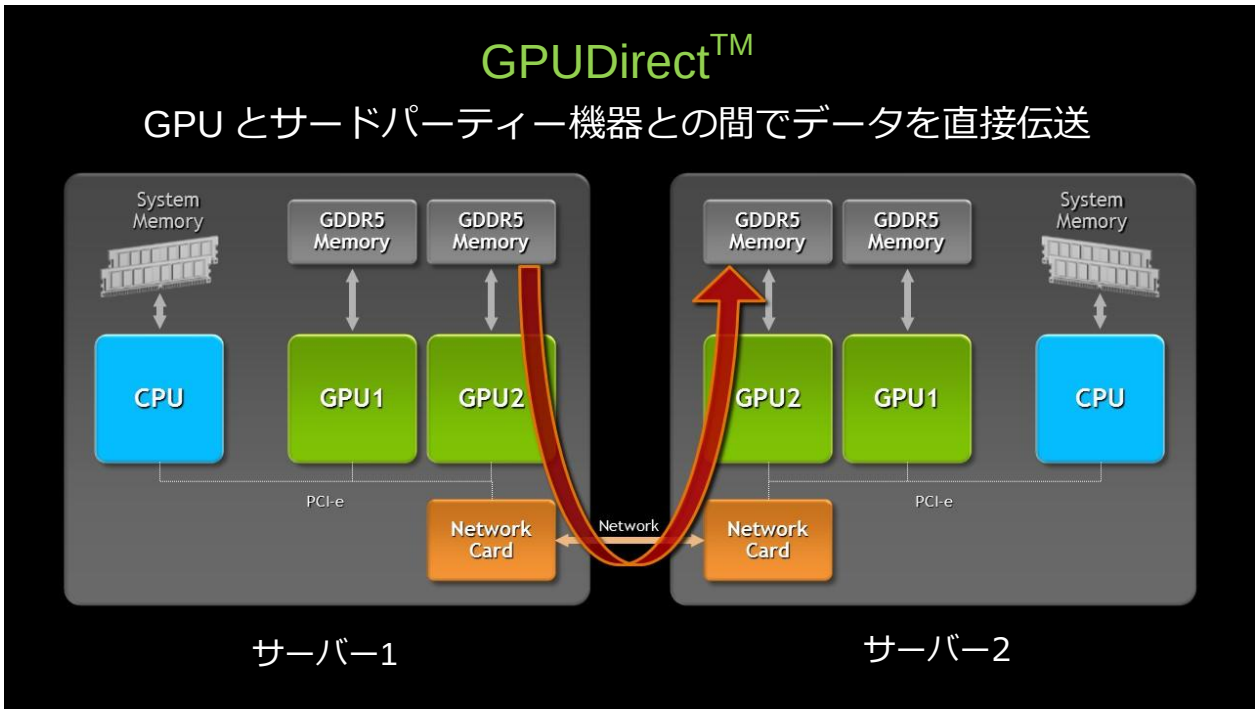
大量データを処理する場合、コンピュートパフォーマンスを高めるためにはデータスループットの向上とレイテンシの低減が不可欠になります。Kepler GK110 は、NVIDIA GPUDirect の RDMA 機能をサポートしました。これは、IB アダプター、NIC、SSD といったサードパーティー機器から GPU メモリーに直接アクセスを許すことでパフォーマンスを高めるものです。CUDA 5.0 において、GPUDirect は、以下の重要な機能を提供します。

- NIC-GPU 間のダイレクトメモリーアクセス（DMA）。CPU 側でデータのバッファリングを行う必要はありません。
- GPU とネットワーク内の他ノードとの間で、MPISend/MPIRecv によるデータ転送効率を大幅に高めます。
- CPU のバンド幅とレイテンシのボトルネックを解消します。
- サードパーティー製のさまざまなネットワーク機器、キャプチャ機器、ストレージ機器で使用可能です。

石油探査やガス探査の地質画像処理で使用されるリバースタイム・マイグレーションのようなアプリケーションでは、大きな画像データを複数 GPU で分散処理します。何百個も

の GPU を協調させ、中間結果を交換しながらデータを処理する必要があります。このような場合 GPU 間の通信に GPUDirect を使用することにより、より高いバンド幅を集約することができます。同一サーバ内の GPU 間では P2P 機能が使用され、異なるサーバにある GPU 間では RDMA 機能が使用されます。

Kepler GK110 は、ピア・ツー・ピアや GPUDirect for Video などの GPUDirect 機能もサポートしています。



GPUDirect RDMA では、ネットワークアダプターなどのサードパーティー機器から直接、GPU メモリーへアクセスすることができます。つまり、異なるノードに搭載された GPU 同士もデータを直接やりとりできることになります。

まとめ

NVIDIA が 2010 年に発表した Fermi アーキテクチャは、大きな計算量を必要とする諸問題を CPU と GPU の協調により解決するハイブリッド・コンピューティング・モデルによって、ハイパフォーマンス・コンピューティング（HPC）の世界に新しい時代の幕開けを告げるものでした。その HPC 業界において技術水準を再び大きく引きあげたのが、今回新たに発表された Kepler GK110 GPU です。

Kepler GK110 は基本デザインから一貫として卓越した電力効率を目指し、それによって計算処理のパフォーマンスとスループットを最大化できるように設計されました。Kepler GK110 のアーキテクチャには、SMX やダイナミック並列処理、Hyper-Q など、ハイブリッド・コンピューティングの処理速度を劇的に高め、プログラムを容易にし、アプリケーションの応用範囲を広げるためのイノベーションが数多く採用されています。Kepler GK110 GPU は、ワークステーションからスーパーコンピューターに至るまでさまざまなシステムに搭載され、HPC 分野の中でも最も困難な課題に立ち向かって行くことになるでしょう。

付録 A－CUDA の概要

CUDA はハードウェアとソフトウェアを組み合わせたプラットフォームで、C、C++、Fortran などさまざまな言語で書かれたプログラムを NVIDIA GPU で実行することができます。CUDA プログラムはカーネルという名前の並列関数を呼び出します。各カーネルは、複数の並列スレッドによる並列実行となります。図 1 に示すように、スレッドをまとめたものをスレッドブロック、スレッドブロックをまとめたものをグリッドと呼び、プログラムやコンパイラではこれらを単位として取り扱います。スレッドブロックを構成するスレッド 1 本 1 本がそれぞれカーネルのインスタンス一つを実行するのです。各スレッドは自分が属するスレッドブロックおよびグリッドにおけるスレッド ID とブロック ID を持つほか、プログラムカウンター、レジスタ、スレッド単位のローカルメモリー、入力、出力結果を持ちます。

同時並行で処理を行うスレッドのセットがスレッドブロックです。同じスレッドブロックに属するスレッドはバリア同期と共有メモリーにより協調して動作します。スレッドブロックも自分が属するグリッドにおけるブロック ID を持ちます。スレッドブロックを行列としてまとめたものがグリッドです。グリッドは全体で一つのカーネルを実行し、グローバルメモリーからの入力データの読み取り、グローバルメモリーへの出力データの書き出し、依存関係にあるカーネルコールの同期といった処理を行います。CUDA 並列プログラミングモデルでは、レジスタスピルや関数呼び出し、C の自動配列変数などに使うローカルメモリー空間がスレッドごとに確保されます。スレッドブロックのレベルでは、並列アルゴリズムに必要なスレッド間の通信、データの共有、結果の共有に使う共有メモリー空間がブロックごとに確保されます。複数スレッドブロックで構成されるグリッドは、カーネル全体をカバーするグローバルな同期を行ったあと、グローバルメモリー空間で結果を共有します。

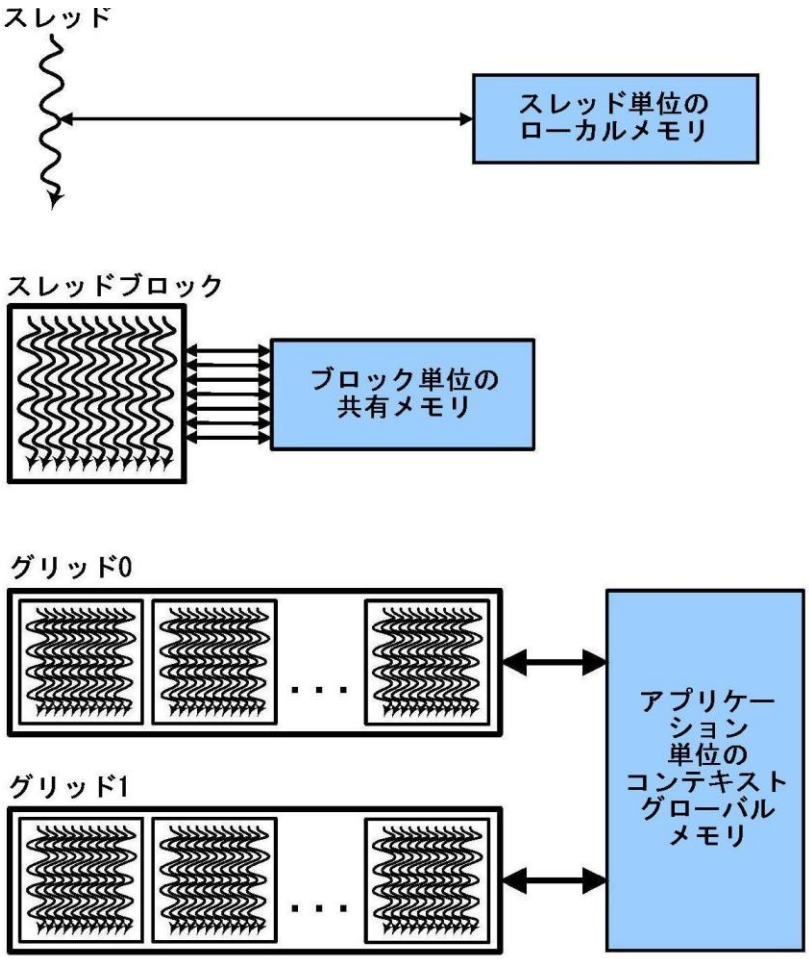


図 1：CUDA におけるスレッド、ブロック、グリッドの階層構造。それぞれに対応し、スレッド単位のプライベートメモリー空間、ブロック単位の共有メモリー空間、アプリケーション単位のグローバルメモリー空間があります。

CUDA のハードウェア実行

CUDA では、GPU が持つプロセッサの階層構造に、スレッドの階層構造が対応しています。つまり、「GPU がひとつあるいは複数のカーネルグリッドを実行する」、「ストリーミング・マルチプロセッサー（Fermi の SM／Kepler の SMX）がひとつあるいは複数のスレッドブロックを実行する」、そして、「CUDA コアを初めとする SMX の実行ユニットがスレッド命令を実行する」という形です。SMX はスレッドを 32 本ごとのグループとして実行しますが、これをワープと呼びます。プログラマーは、ワープ実行を気にせず各スカラースレッドをプログラミングするだけで、機能を実現することができます。ただし、同じワープに属するスレッドが同じコードパスを実行し、近接したアドレスのメモリーにアクセスするようにプログラミングすると、パフォーマンスが大きく向上します。

注記

コメント、意見、NVIDIA の設計仕様、レファレンスボード、ファイル、図面、診断、リストなどの文書（以下、総体として、あるいは個別に「マテリアル」と呼ぶ）に記載される情報は、すべて「現状」ベースでの提供となります。NVIDIA は、マテリアルについて、明示的にも暗示的にも法的にも、保証を一切せず、非侵害、市販性、特定の目的への適合性のいずれについてもいかなる保証もしないことをここに明示します。

NVIDIA では、信頼性のある正確なものだと判断した情報を提供しています。しかし、この情報の利用に伴っていかなる結果が発生しても、また、その利用によって特許など第三者の権利の侵害が発生しても、NVIDIA Corporation はその責めを負わないものとします。NVIDIA Corporation が保有する特許あるいは特許権について、暗示的にもライセンスを供与するものではありません。本文書に記載された仕様は、事前の通知なく変更する可能性があります。本文書は、過去に提供した情報のすべてに優先します。NVIDIA Corporation の製品は、NVIDIA Corporation から書面による承認を得ない限り、生命維持にかかわる機器やシステムの重要部品として使うことはできません。

商標

NVIDIA、NVIDIA ロゴ、CUDA、FERMI、KEPLER、GeForce は、米国およびその他の国における NVIDIA Corporation の商標または登録商標です。その他の企業名および製品名は、それぞれ各社の商標である可能性があります。