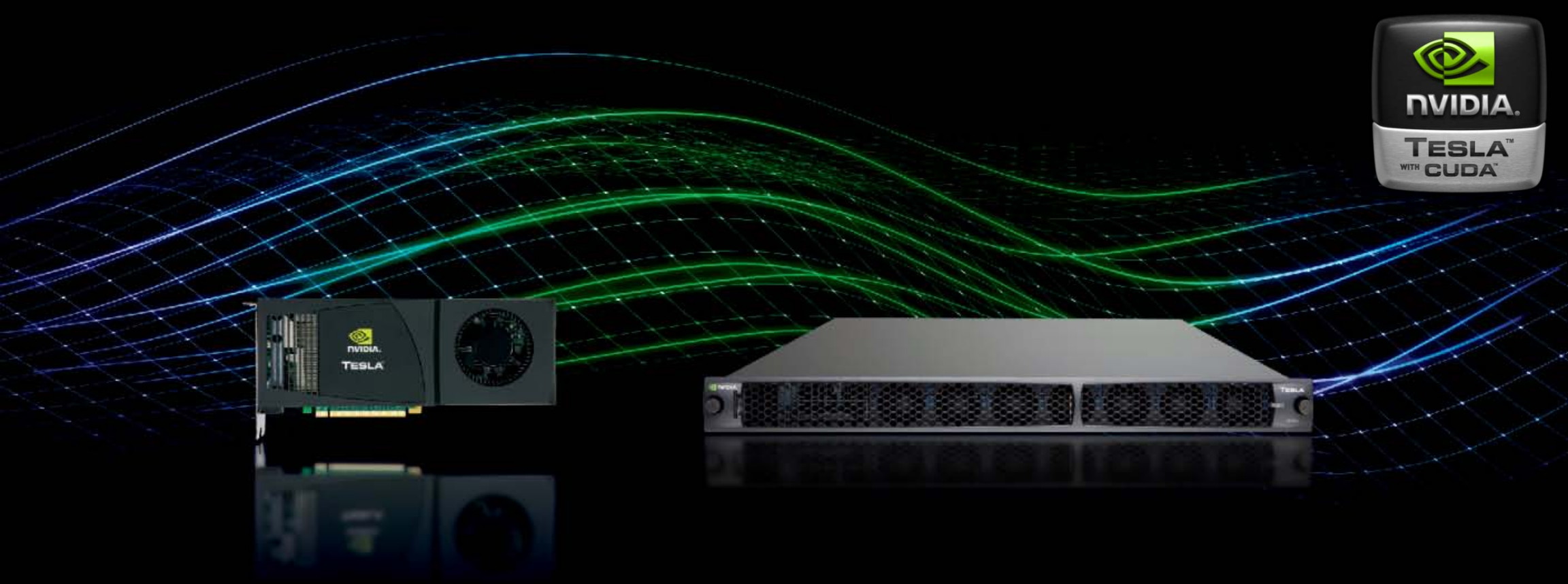



電子情報通信学会研究会　組込みシステム研究会 (IPSJ-EMB)　　2010年1月28日

# 超並列マルチコアGPUを用いた高速演算処理の実用化

NVIDIA Solution Architect
馬路　徹

© 2009 NVIDIA Corporation

# 目次

- なぜ 今GPUコンピューティングか?
  - CPUの性能向上速度が減速
  - “性能 向上= 並列処理”にGPUコンピューティングが応える
  - CUDAシステムアーキテクチャによる超並列処理の実現、普及
- NVIDIA GPUアーキテクチャの進歩
  - 3Dグラフィックスの進歩及び統合化シェーダエンジンへの変遷
  - 統合化シェーダエンジンへのCUDAの組込み
- 次世代GPUコンピューティング、Fermiアーキテクチャ
  - プロセッサ / メモリーアーキテクチャ
  - GigaThreadスレッド制御機構による更なる高性能化
  - 統合化プログラム開発環境
- まとめ

© 2009 NVIDIA Corporation

# CUDAはPerformance = Parallelismを実現

## Performance = Parallelismへの期待

- このParallelismを実現するために必要なことは?
  - 相当な数の高効率プロセッサ（4個、8個のオーダではない）
  - 並列処理を抽象化できるプログラミング・システム

- “CUDA (Compute Unified Device Architecture”
  - GPUは数百個の多機能、高速プロセッサを内蔵
  - マルチスレッド・アーキテクチャがこの超並列マルチコアを高効率に活用
  - CUDA は普及しているC言語の拡張からスタートし、現在はFortran、OpenCL や Direct Computeまで拡張されている
  - CUDA は並列プログラミングの詳細な記述からプログラマーを解放

© 2009 NVIDIA Corporation

# 最初に「C with CUDA Extension」を開発

標準Cコード

```
void  saxpy_serial(int n, float alpha, float
*x, float *y)
{
        for (int i=0; i<n; ++i)
            y[i] = alpha * x[i] + y[i];
}
// Invoke serial saxpy() kernel
saxpy_serial(n, 2.0, x, y);
```

CUDAコード

```
__global__ void saxpy_parallel(int n, float alpha, float *x, float *y)
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if (i < n) y[i] = alpha * x[i] + y[i];
}
// Invoke parallel saxpy() kernel (256 threads per block)
int nblocks = (n + 255)/256;
saxpy_parallel <<< nblocks, 256 >>>(n, 2.0, x, y);
```

CPUからコールし
GPUで実行する関数

ｸﾞﾛｰﾊﾞﾙ･ｽﾚｯﾄﾞIDを
そのままｲﾝﾃﾞｯｸｽに活用

階層的スレッド分割数（ブロック数／グリッド）

（スレッド数／ブロック）

© 2009 NVID

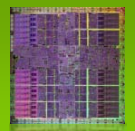
# 今日のGPU コンピューティングの普及

## 市場展開モーメンタム

- CUDA実装のGPUが市場に1億個以上出荷
- 60,000以上のGPU コンピューティング開発者
- Windows, Linux 及び MacOS プラットフォームのサポート
- 200以上の大学で GPU コンピューティングの講座

**GPU コンピューティング・アプリ**

| C + CUDA Extension | OpenCL Khronos Apple | Direct Compute Microsoft | FORTRAN | Java and Python planned |
|---|---|---|---|---|



**NVIDIA GPU**
with the CUDA Parallel Computing Architecture

© 2009 NVIDIA Corporation

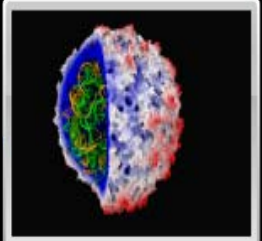
OpenCL is trademark of Apple Inc. used under license to the Khronos Group Inc.

# CUDA Not 2x or 3x, Speed-ups are 20x to 150x

Results with Telsa 8 Series relative to compute performance using CPU exclusively

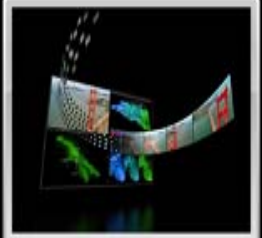
146X

Interactive visualization of volumetric white matter connectivity



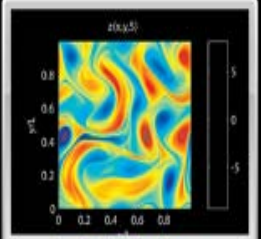
36X

Ionic placement for molecular dynamics simulation on GPU



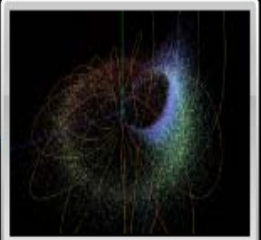
19X

Transcoding HD video stream to H.264



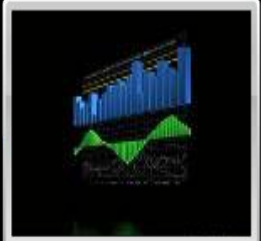
17X

Simulation in Matlab using .mex file CUDA function



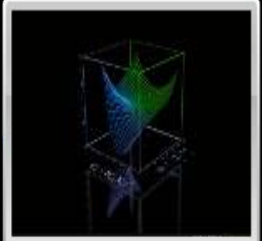
100X

Astrophysics N-body simulation



149X

Financial simulation of LIBOR model with swaptions



47X

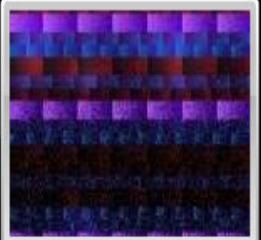
GLAME@lab: An M-script API for linear Algebra operations on GPU

20X

Ultrasound medical imaging for cancer diagnostics

24X

Highly optimized object oriented molecular dynamics



30X

Cmatch exact string matching to find similar proteins and gene sequences

© 2009 NVIDIA Corporation

# 並みの性能向上ではない

- 2-3倍の性能向上は単なる「高性能化」
  - 顧客の基本的なワークフローに変化は無い
- 5-10倍の性能向上は「画期的」
  - 装置のアップグレードの価値は十分にある
  - （一部または大部分の）アプリ・ソフトを書き換える意味はある
- 100倍以上の性能向上は「世界観を変える！」
  - プラットフォームの取替えの価値はある
  - アプリケーションのアーキテクチャまでを見直す意味がある
  - 今まで実用的に不可能であったアプリの開発が可能になる
  - 科学技術において「新発見までの時間」を短縮する画期的な変化をもたらす

© 2009 NVIDIA Corporation

# GPUコンピューティングが導入されたOS

## 身近になるGPUコンピューティング

Mac OS X Snow Leopard

Windows®7

# 目次

- なぜ 今GPUコンピューティングか?
  - CPUの性能向上速度が減速
  - “性能 向上= 並列処理”にGPUコンピューティングが応える
  - CUDAシステムアーキテクチャによる超並列処理の実現、普及
- NVIDIA GPUアーキテクチャの進歩
  - 3Dグラフィックスの進歩及び統合化シェーダエンジンへの変遷
  - 統合化シェーダエンジンへのCUDAの組込み
- 次世代GPUコンピューティング、Fermiアーキテクチャ
  - プロセッサ / メモリーアーキテクチャ
  - GigaThreadスレッド制御機構による更なる高性能化
  - 統合化プログラム開発環境
- まとめ

© 2009 NVIDIA Corporation

# NVIDIA GPUによる3Dグラフィックスの進歩

## ﾌﾟﾛｸﾞﾗﾏﾋﾞﾘﾃｨｰの大幅向上 (統合化ｼｪｰﾀﾞｰｴﾝｼﾞﾝ) → 汎用演算器

**1995**
NV1
**1 Million**
Transistors

**1999**
GeForce 256
DirectX 7
**22 Million**
Transistors

**2002**
GeForce4
Direct X 8
**63 Million**
Transistors

**2003**
GeForce FX
DirectX 9
**130 Million**
Transistors

**2004**
GeForce 6
DirectX 9c
**222 Million**
Transistors

**2005**
GeForce 7
DirectX 9c
**302 Million**
Transistors

**2006**
GeForce 8
DirectX 10 (Vista)
**681 Million**
Transistors

**2008**
GeForce
GTX200
**1.4 Billion**
Transistors

ﾌﾟﾛｸﾞﾗﾏﾋﾞﾘﾃｨｰの向上

Vertex(FP)
+Pixel Shader

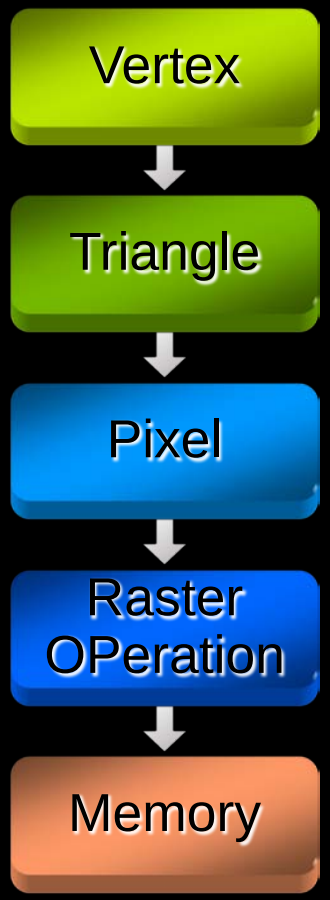
Vertex(FP)
+Pixel(FP) Shader

統合化Shader
**CUDA**

© 2009 NVIDIA Corporation

# 20年間続いたグラフィックス・アーキテクチャ

- 各機能に固定された用途のハードウエア




座標変換及び光源処理
DirectX8よりプログラマブル頂点シェーダーとなる

三角形、点、線セットアップ

フラット・シェーディング、テクスチャーマッピング等
DirectX8よりプログラマブル・ピクセルシェーダーとなる

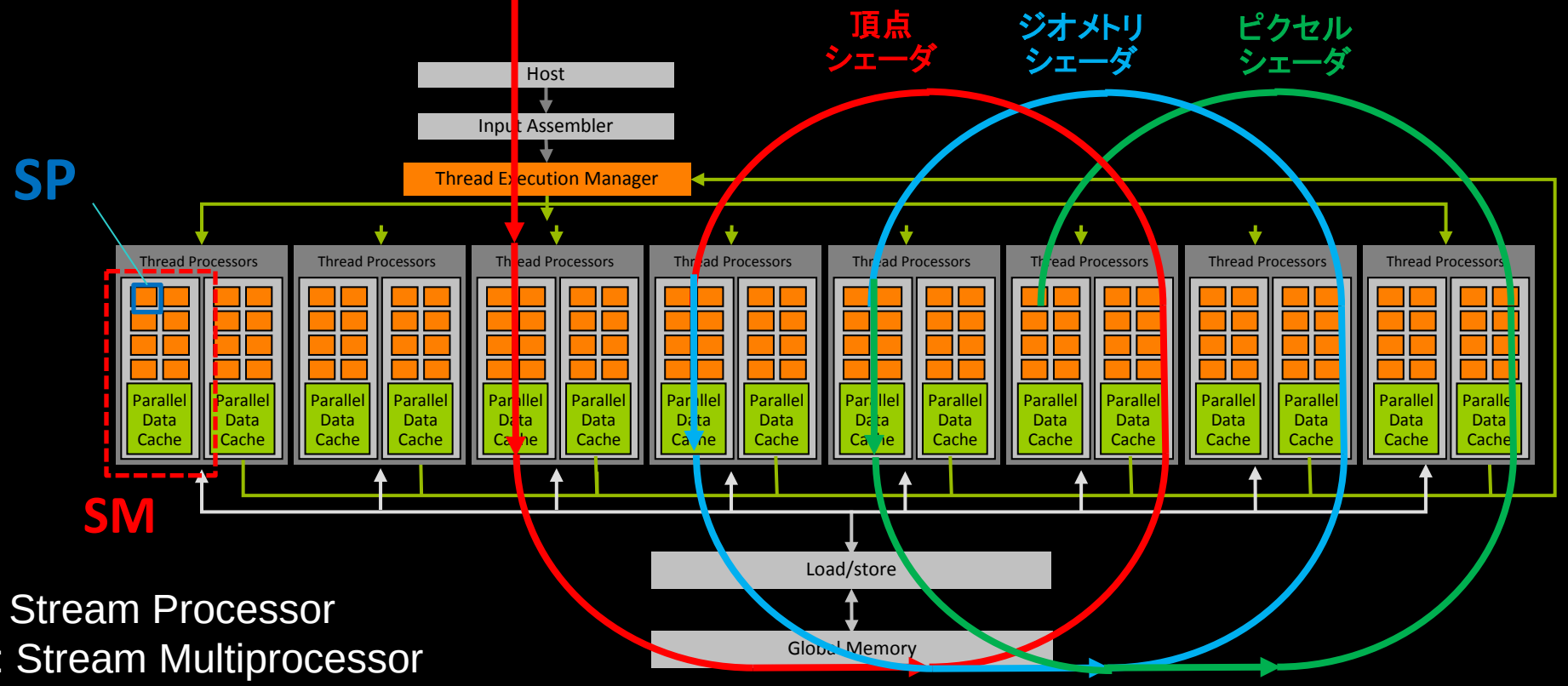
ブレンディング、Zバッファー、アンチエイリアシング

メモリ

© 2009 NVIDIA Corporation

# DirectX 10 (Vista) 以降のGPU
## 統合型プログラマブル・シェーダエンジン

- GPUはマルチスレッドを処理する超並列マルチコアプロセッサとなった。
- 頂点シェーダ、ジオメトリ・シェーダ及びピクセル・シェーダは上記により実行される。



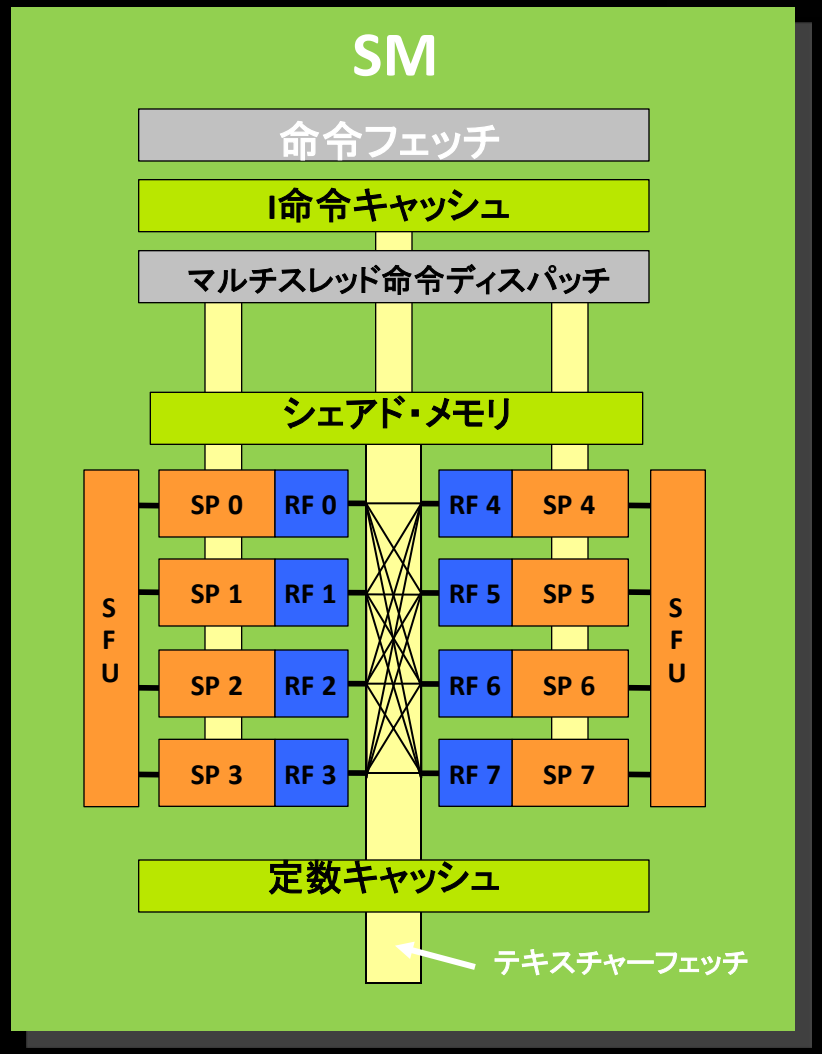

SP: Stream Processor
SM: Stream Multiprocessor

© 2009 NVIDIA Corporation

# 第1世代のSM (Streaming Multiprocessor)

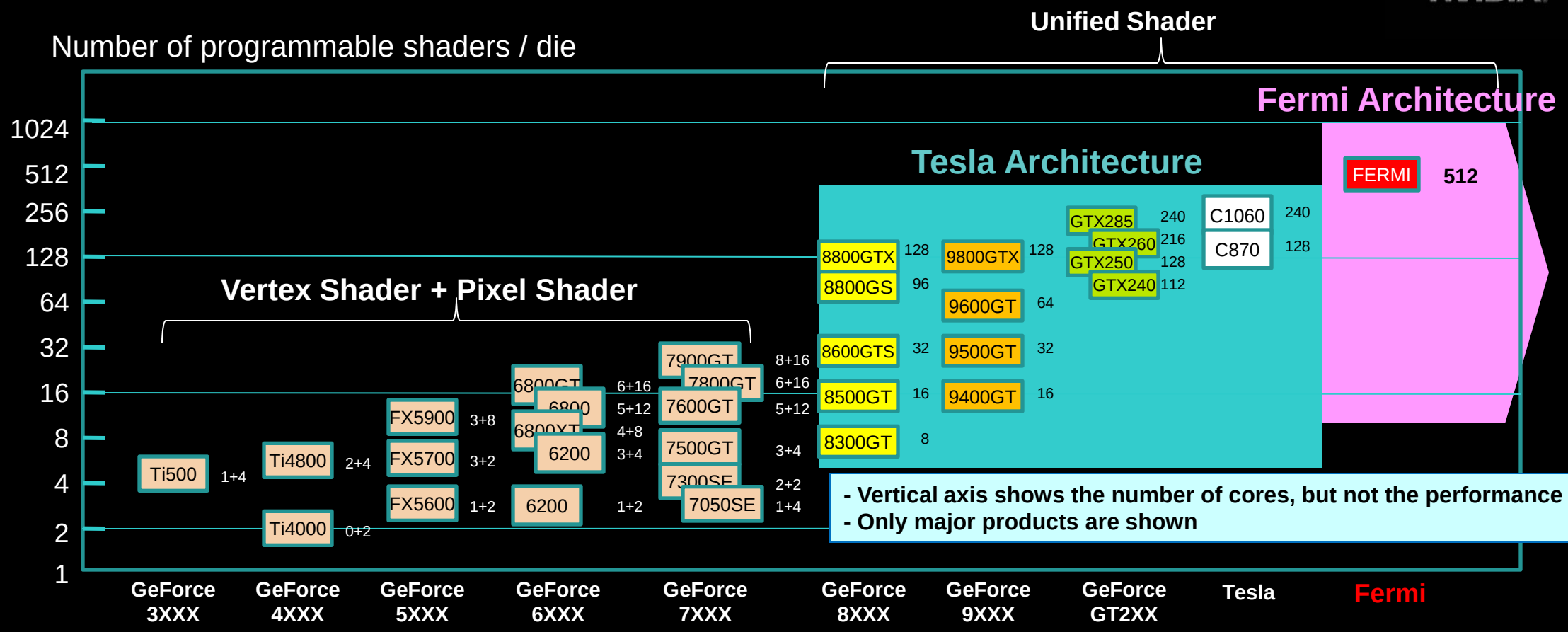
- Streaming Multiprocessor (SM)
  - 8 Streaming Processors (SP)
    各SPは32b単精度浮動小数点、整数演算サポート
  - 2 Super Function Units (SFU)
    三角関数、平方根等サポート
  - 全てのSPは同一命令を実行開始（途中分岐あり）
    SIMT (Single instruction Multiple Thread)

- マルチスレッド命令ディスパッチ
  - 1 から768 スレッドがアクティブ
  - 32スレッド単位(warp) でSIMT命令実行
  - 巨大なローカル・レジスタファイルRF
    8,192 Registers / SM
    ハードウエア・コンテキストスイッチを容易化
- 16 KB シェアード・メモリ



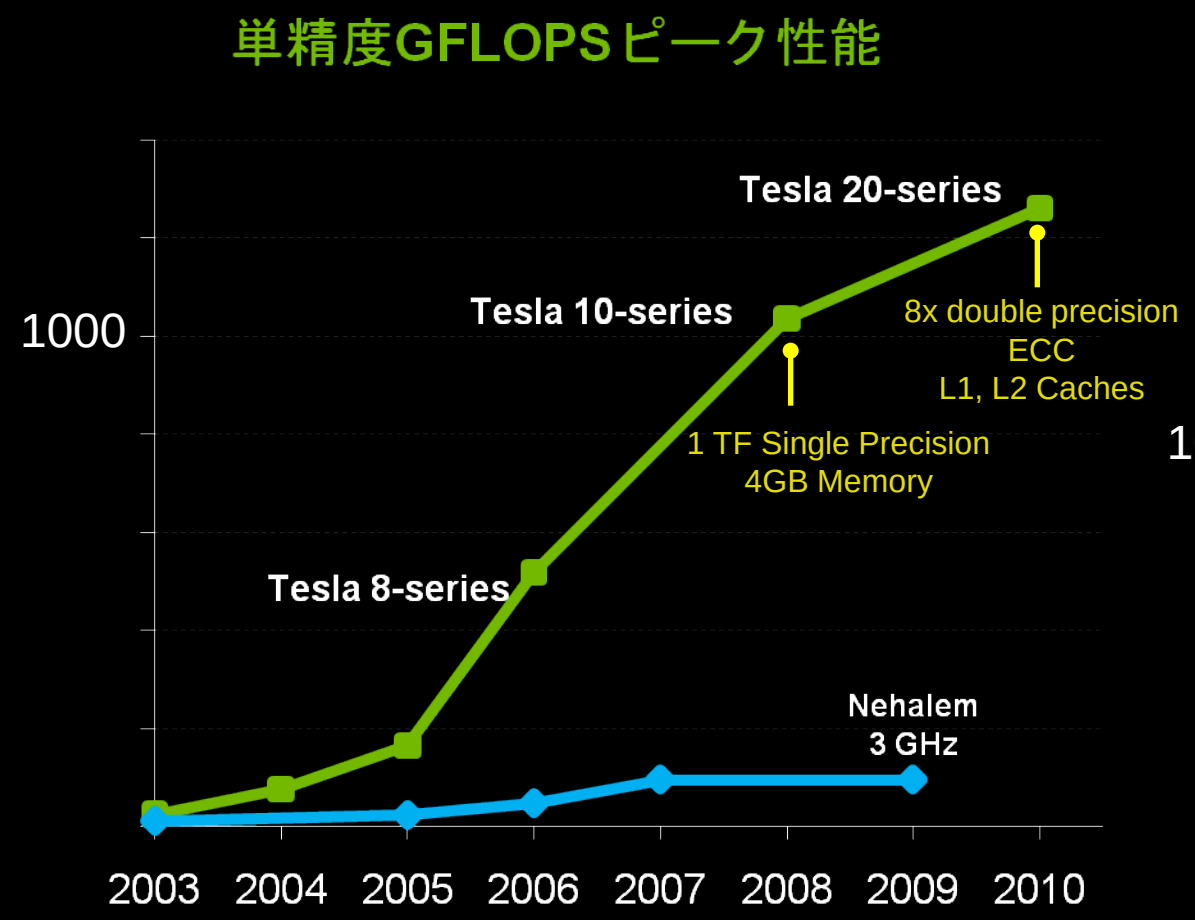

© 2009 NVIDIA Corporation

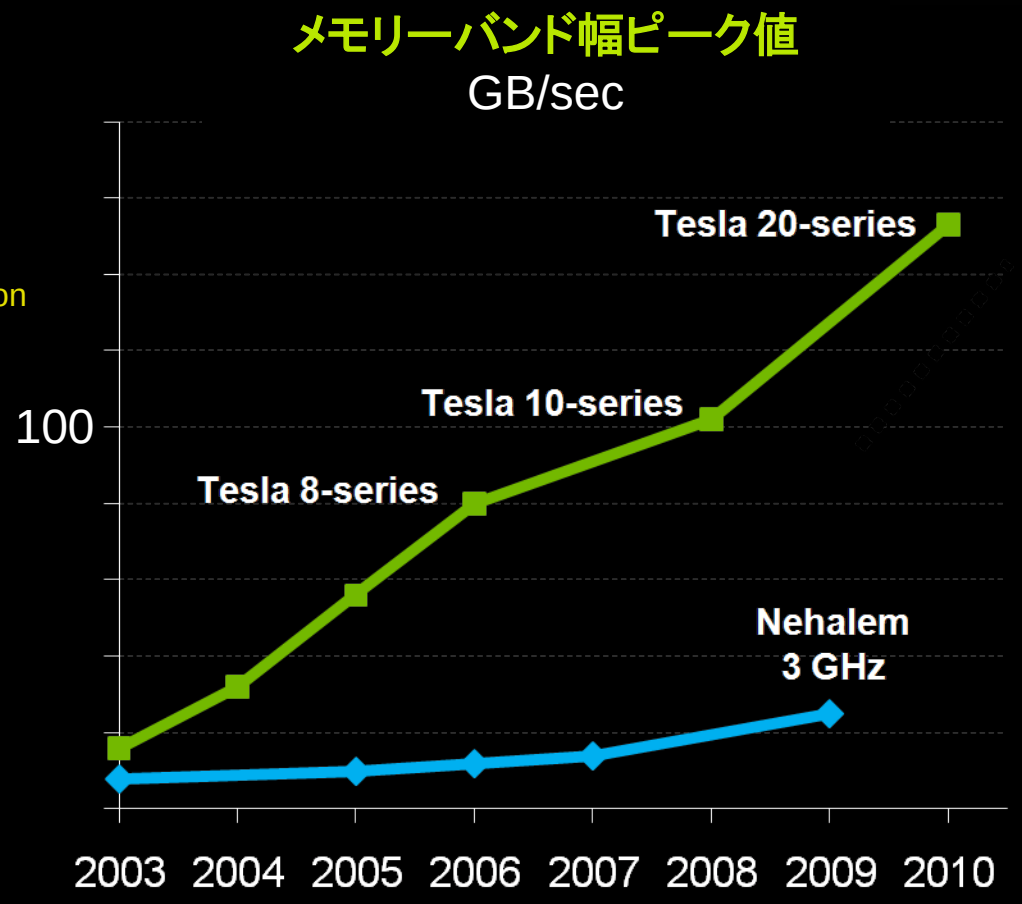
# NVIDIA GPU コア数の変遷




© 2009 NVIDIA Corporation

# GPU-CPU 性能及びメモリーバンド幅の 差は拡大




NVIDIA GPU
X86 CPU

© 2009 NVIDIA Corporation

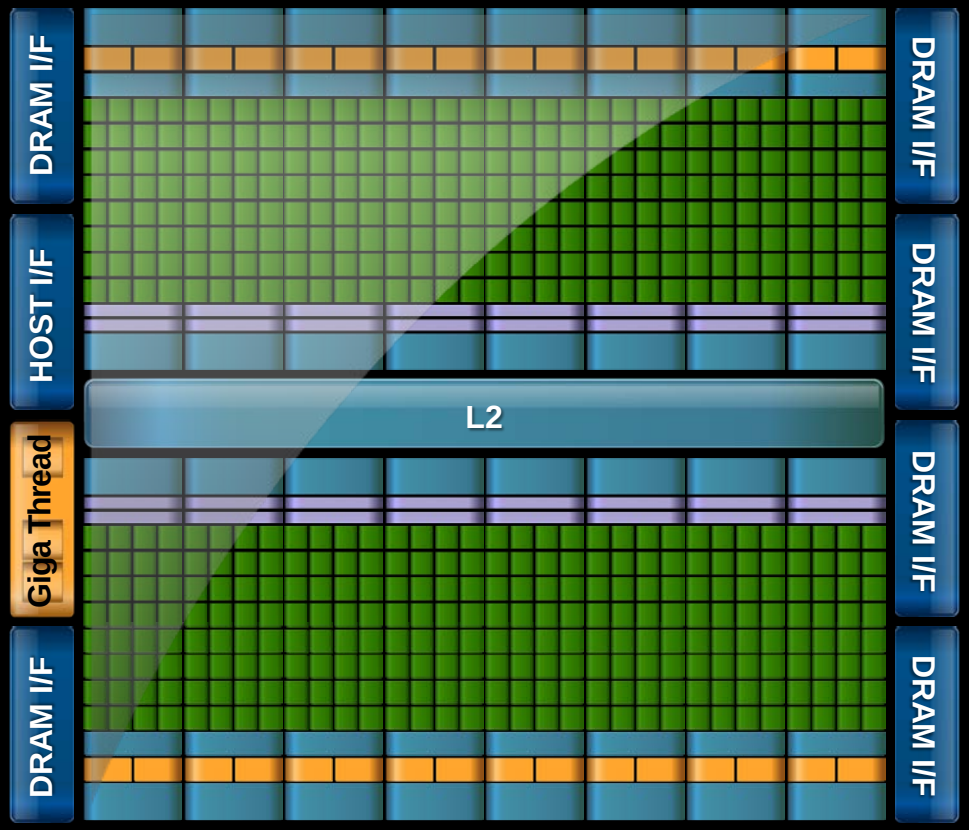
# 目次

- なぜ 今GPUコンピューティングか?
  - CPUの性能向上速度が減速
  - “性能 向上= 並列処理”にGPUコンピューティングが応える
  - CUDAシステムアーキテクチャによる超並列処理の実現、普及
- NVIDIA GPUアーキテクチャの進歩
  - 3Dグラフィックスの進歩及び統合化シェーダエンジンへの変遷
  - 統合化シェーダエンジンへのCUDAの組込み
- 次世代GPUコンピューティング、Fermiアーキテクチャ
  - プロセッサ / メモリーアーキテクチャ
  - GigaThreadスレッド制御機構による更なる高性能化
  - 統合化プログラム開発環境
- まとめ

© 2009 NVIDIA Corporation

# ‘Fermi’ アーキテクチャのご紹介

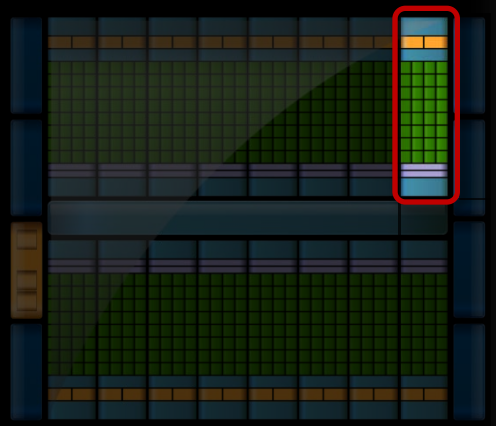
## スーパコンピュータの魂を持つGPU

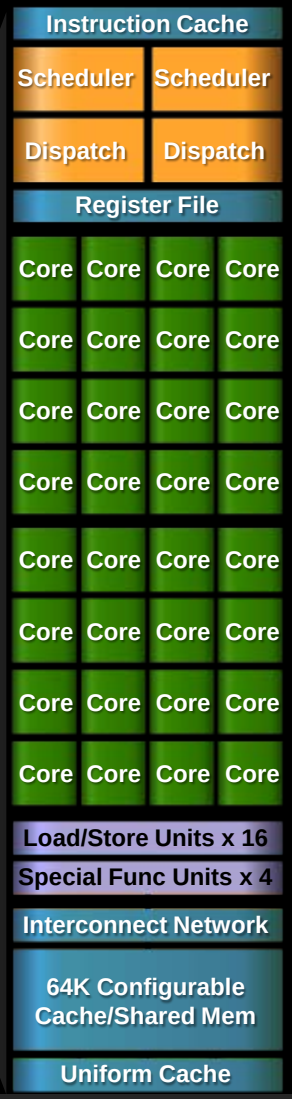



- 30億個のトランジスター
- コア数を倍増(512 コア)
- 倍精度浮動小数点演算ピーク性能が8倍
- GPUとして初めてECCを導入
- L1 及び L2 キャッシュを内蔵
- 約2倍のメモリーバンド幅(GDDR5)
- 最大1 Terabyte の GPU メモリ
- 複数Kernel同時実行、C++サポート

# SM (Streaming Multiprocessor) アーキテクチャ

- SM当たり32 CUDAコア (総数512個)
- 倍精度浮動小数点演算ピーク性能は8倍
  - 単精度演算ピーク性能の50%
- 2個の Thread Scheduler
  - 2ワープを2組の16個のCUDA Core, 16個のLoad/Store Unit, 4個のSFUに同時にディスパッチする
- シェアードメモリ、L1キャッシュとして使用する 64 KB RAM（構成可変）





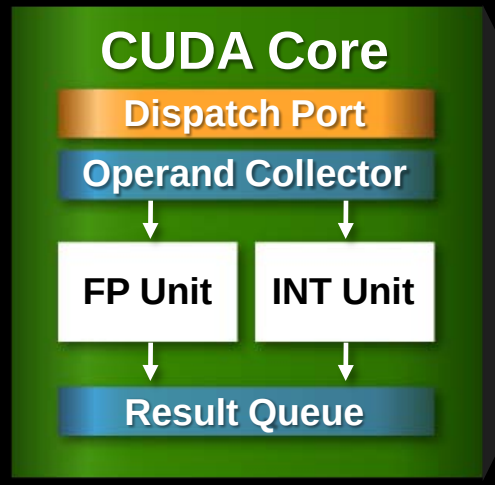

© 2009 NVIDIA Corporation

# CUDA コア・アーキテクチャ

- 最新の浮動小数点演算規格 IEEE 754-2008 準拠
（最新CPUをも凌駕）

- Fused multiply-add (FMA) 命令を
倍精度、単精度でサポート
（積和の最終段でラウンディングするため、
各段で各々行うより精度が向上）

- 新規設計の全命令32bサポート整数ALU
64-bit 及びそれ以上の精度に対しても最適化設計

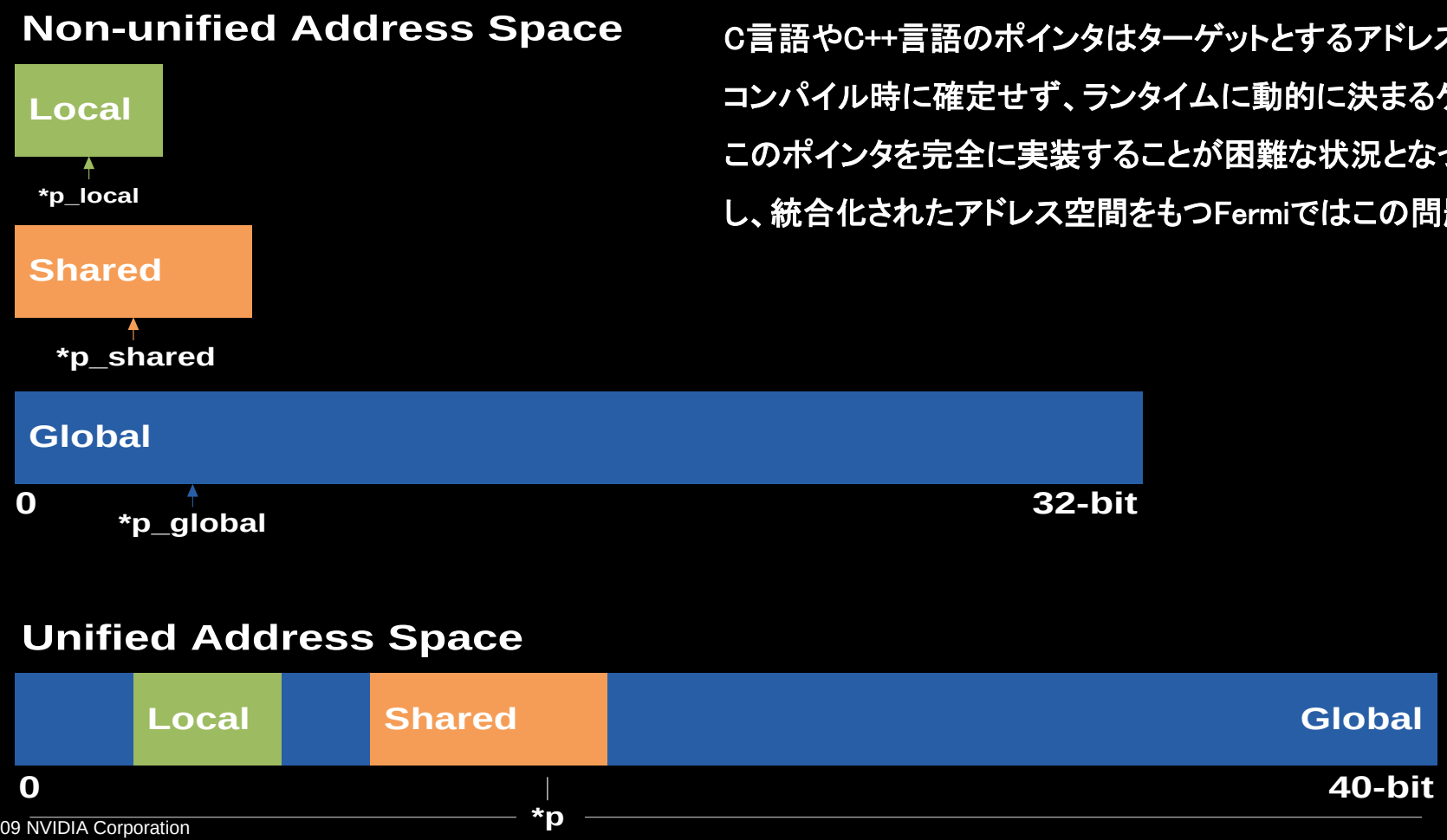


© 2009 NVIDIA Corporation

# 統合化された64ビットメモリー空間

- C／C++ ポインターのフルサポート

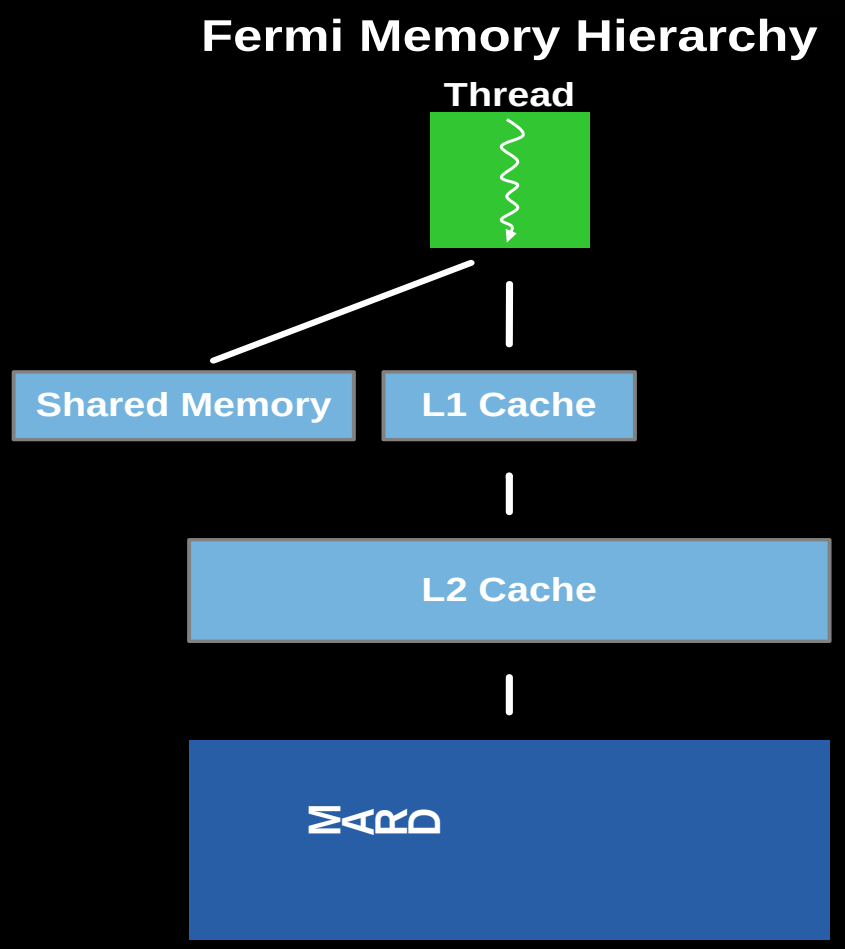
C言語やC++言語のポインタはターゲットとするアドレス空間が必ずしもコンパイル時に確定せず、ランタイムに動的に決まるケースがあるため、このポインタを完全に実装することが困難な状況となっていました。しかし、統合化されたアドレス空間をもつFermiではこの問題もなくなりました。




© 2009 NVIDIA Corporation

# キャッシュメモリ階層構造

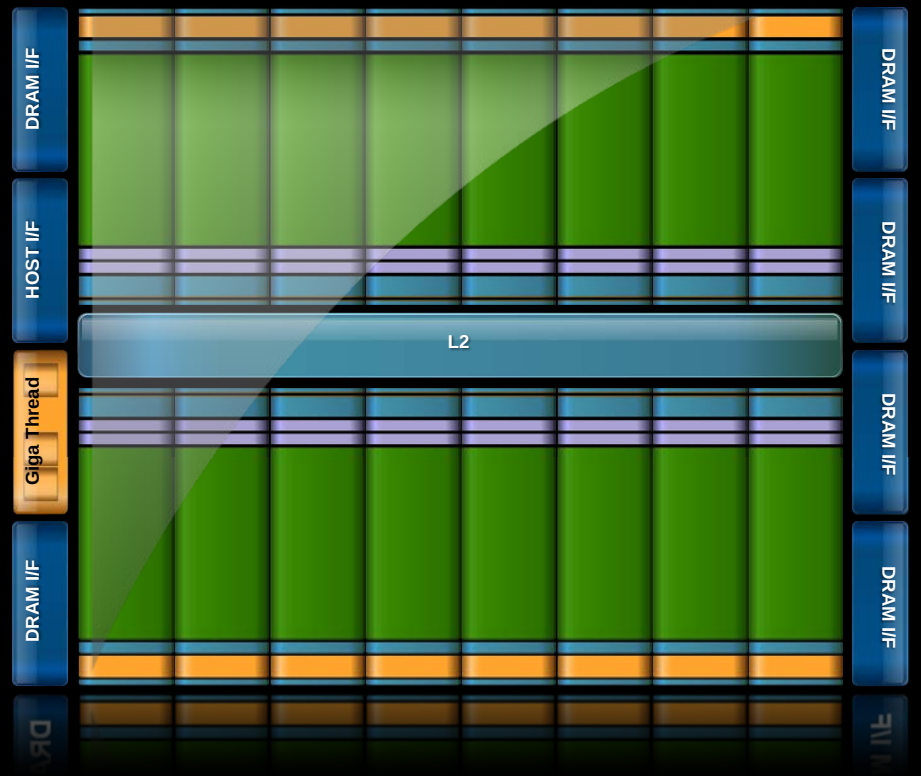
- オンチップ・シェアードメモリとともに本格的なキャッシュ階層構造を有する初めてのGPU
- 各SM (32 cores)にL1キャッシュ（48KBまたは16KB）
  - メモリバンド幅の改善及びアクセス遅延の低減
- ユニファイド L2 キャッシュ(768 KB)
  - GPU上の全コアにわたり高速に、コーヒレンント・データをシェア




© 2009 NVIDIA Corporation

# 拡張され、高速化されたメモリ・インタフェース

- GDDR5 メモリーインタフェース
  - GDDR3の2倍の速度
- 最大1 Terabyte のGPUメモリ
  - 大きなデータセットを取り扱うことが可能
  - 現在40bアドレス。命令は64bをサポート
- 384ビットのメモリーインタフェース
  - 64b partitioning x 6




© 2009 NVIDIA Corporation

# ECC (Error Checking and Correction)

- **DRAMのECCによるデータエラーの除去**
  - GDDR5 メモリーのECCサポート
- **内部の主要な記憶素子もECC で保護**
  - レジスターファイル、L1キャッシュ、L2キャッシュ
- **Single-Error Correct Double-Error Detect (SECDED)サポート**
  - 2ビットの誤りはソフト処理（再度実行等）

© 2009 NVIDIA Corporation

# IEEE 754-2008 規格準拠の高精度演算

- **IEEE 754-2008 results**
  - 64-bit double precision
  - 32-bit single precision
  - full-speed denormal operands & results
  - NaNs, +/- Infinity
- **IEEE 754-2008 rounding**
  - nearest even, zero, +inf, -inf
- **IEEE 754-2008 Fused Multiply-Add (FMA)**
  - D = A*B + C;
  - No loss of precision
  - IEEE divide & sqrt use FMA

Multiply-Add (MAD): D = A*B + C;

A × B

= Product (truncate digits)

+ C

= D

Fused Multiply-Add (FMA): D = A*B + C;

A × B

= Product (retain all digits)

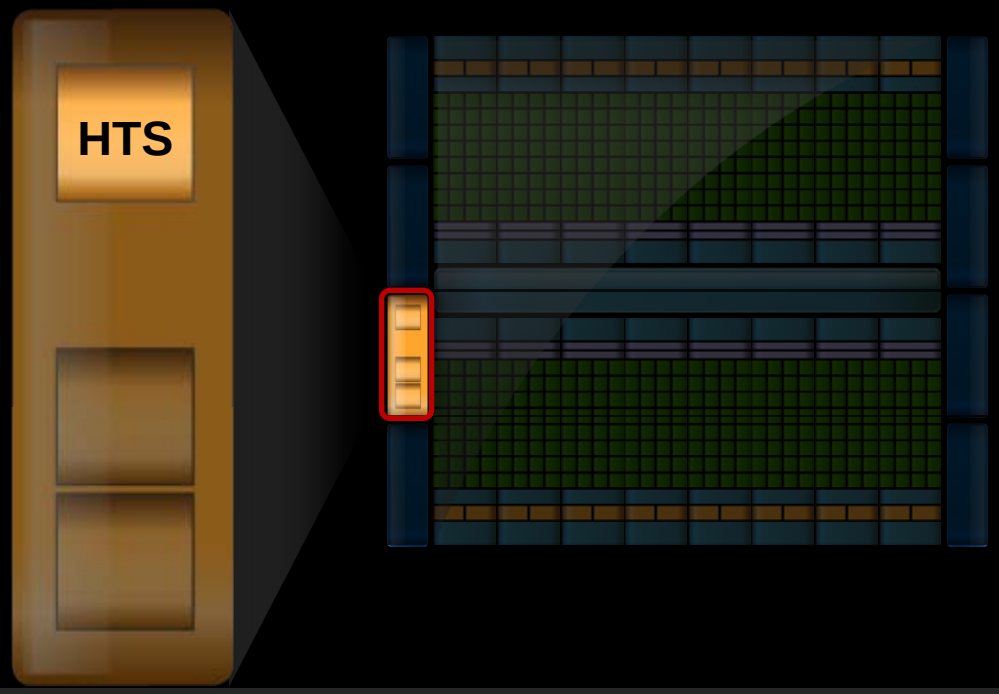
+ C

= D (no loss of precision)

© 2009 NVIDIA Corporation

# GigaThread™ Hardware Thread Scheduler (HTS)

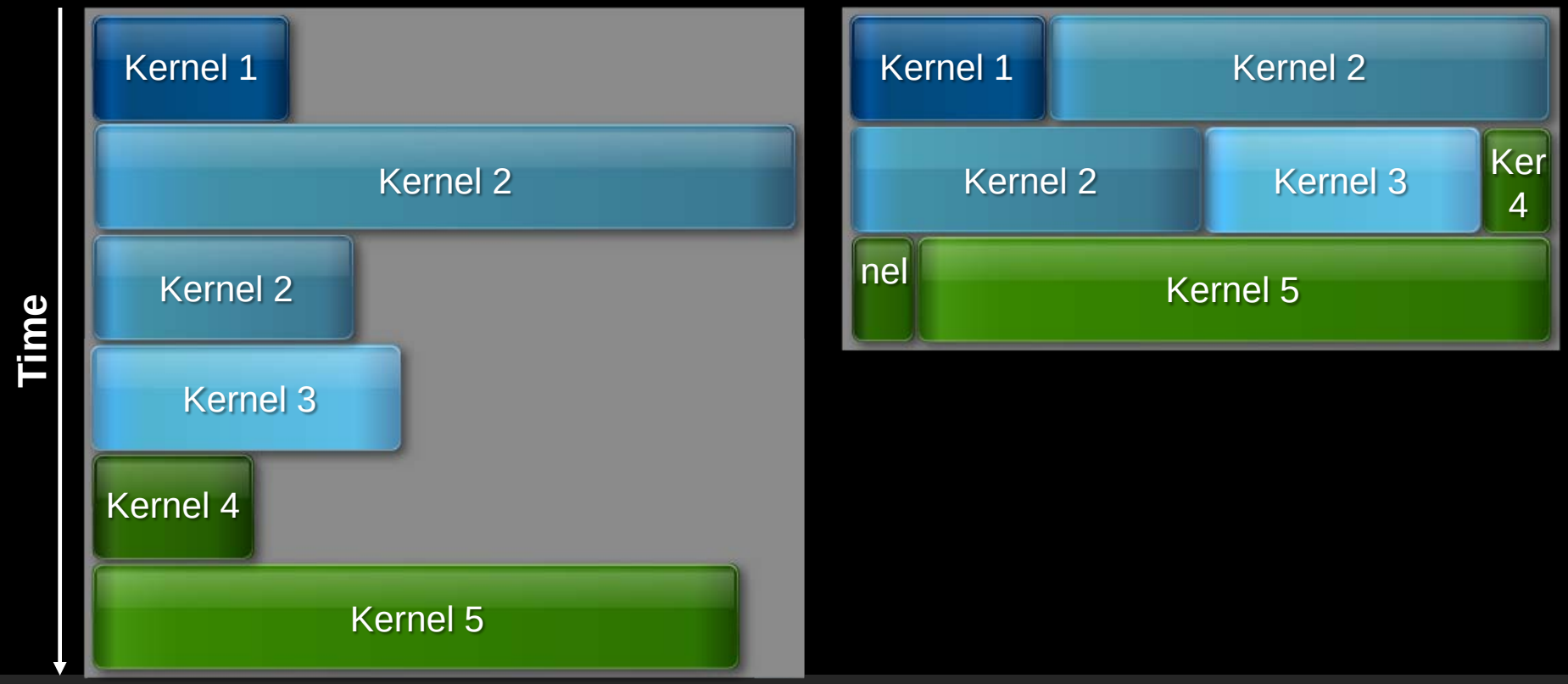
- 階層的に何千ものアクティブなスレッドを管理
- コンテキストスイッチが10倍高速
- 複数kernelの同時実行




© 2009 NVIDIA Corporation

# GigaThread Hardware Thread Scheduler

複数Kernel同時実行 + 高速コンテキストスイッチ



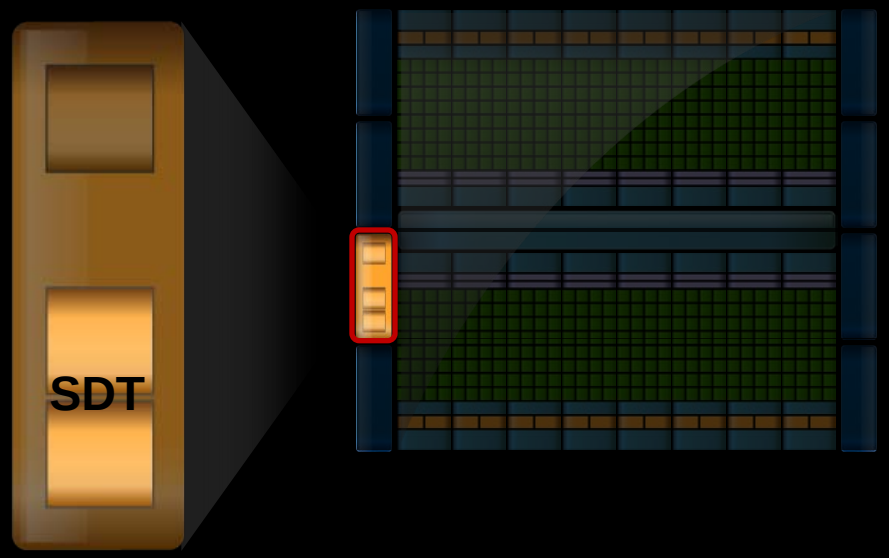

**Serial Kernel Execution**

**Parallel Kernel Execution**

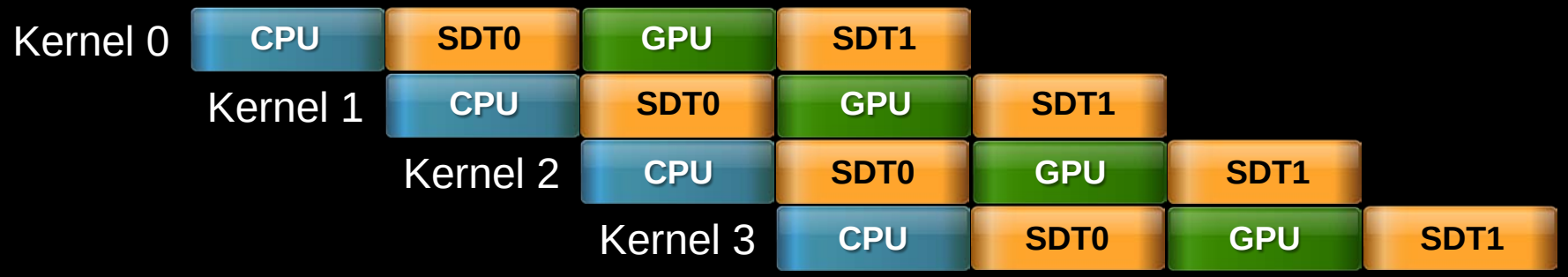
© 2009 NVIDIA Corporation

# GigaThread Streaming Data Transfer Engine

- デュアルDMA エンジン
  - CPU→GPU 及び GPU→CPU データ転送の同時実行
  - CPU と GPU演算と完全にオーバラップ可能




- 処理の流れ:




© 2009 NVIDIA Corporation

# 各世代の比較

| GPU | G80 | GT200 | Fermi |
|---|---|---|---|
| 集積トランジスタ数 | 6億8100万個 | 14億個 | 30億個 |
| CUDAコア数 | 128 | 240 | 512 |
| 倍精度浮動小数点演算能力 | × | 30 FMA<br>演算/クロック | 256 FMA<br>演算/クロック |
| 単精度浮動小数点演算能力 | 128 MAD<br>演算/クロック | 240 MAD<br>演算/クロック | 512 MAD<br>演算/クロック |
| ワープスケジューラ<br>(SMあたり搭載数) | 1 | 1 | 2 |
| 特殊関数ユニット(SFU) /SM | 2 | 2 | 4 |
| 共有メモリ/SM | 16KB | 16KB | 48KB/16KB<br>(構成可能) |
| L1キャッシュ/SM | × | × | 16KB/48KB<br>(構成可能) |
| L2キャッシュ/SM | × | × | 768KB |
| ECCメモリのサポート | × | × | ○ |
| 平行実行カーネル数 | × | × | 最大16 |
| ロード/ストアのアドレス幅 | 32ビット | 32ビット | 64ビット |

© 2009 NVIDIA Corporation

# NVIDIA Nexus IDE（統合化開発環境）

- 業界初の超並列アプリ開発用 IDE (Integrated Development Environment)

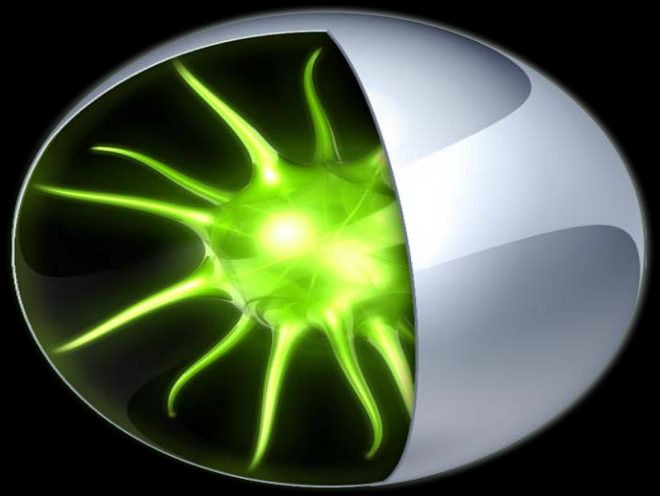
C, C++, OpenCL, DirectCompute 及びDirectX と OpenGLの両グラフィックスAPIをサポート

- 完全にVisual Studioに組込まれた 開発環境

- (CPU + GPU)コプロセッシング・アプリの開発を効率化

  両プロセッサにまたがるソース・デバッグ、性能解析

- 両プロセッサにわたるイベント、データのキャプチャ



Microsoft Visual Studio PARTNER

© 2009 NVIDIA Corporation

# NVIDIA Nexus IDE（統合化開発環境）画面

Nexus CUDA Samples.90 (Debugging) - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Nexus Tools Test Window Help

Debug | Win32 | d_vbo_buffer | CUDA (0,0,0), (0,0,0)

Process: [4560] GPU - matrixMul.e | Thread: [2772376] <No Name> | Stack Frame: Module: 60956632 - [0] _Z9n

Solution Explorer - matrixMul

- Solution 'Nexus CUDA Samples.90' (3 projects)
  - matrixMul
    - inc
    - src
      - matrixMul.cu

NVIDIA Nexus - CUDA Focus Picker

| | | Dimensions |
|---|---|---|
| Block: | 0, 0, 0 | 8, 5, 1 |
| Thread: | 0, 0, 0 | 16, 16, 1 |

Examples
#129 for block index 129
10 for coordinates 10, 0
10, 5 for coordinates 10, 5

OK Cancel

Solution Explorer | Class View

matrixMul_kernel.cu

```
// Loop over all the sub-matrices of A and B
// required to compute the block sub-matrix
for (int a = aBegin, b = bBegin;
        a <= aEnd;
        a += aStep, b += bStep) {

    // Declaration of the shared memory array As used to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each thread loads
    // one element of each matrix
    AS(ty, tx) = A[a + wA * ty + tx];
    BS(ty, tx) = B[b + wB * ty + tx];

    // Synchronize to make sure the matrices are loaded
    __syncthreads();
```

Nexus CUDA Device Summary

| Name | Details |
|---|---|
| Devices | |
| Device 0 | |
| Device 1 | |
| Context 2772376 | Device 0 |
| Module 60956632 | c:/ProgramData/NVIDIA Nexus 1.0/Samples/CUDA/Debug |
| Grid | _Z9matrixMulPfS_S_ii<<<(8,5),(16,16,1), 0>>> |
| Block 0 | Warp Mask: 0x000000FF |
| Warp 0 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, matrixMul_k |
| Warp 1 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 2 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 3 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 4 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 5 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 6 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |
| Warp 7 | Active Mask: 0xFFFFFFFF, PC: 0x000703E8, |

Locals

| Name | Value | Type |
|---|---|---|
| blockDim | {x = 16, y = 16, z = 1} | const dim3 |
| gridDim | {x = 8, y = 5, z = 1} | const dim3 |
| As | 0x00000024 {{0.20108646, 0.23432112, 0.2616657, 0.18860439, ...}, {0.8812524 | float[16][16] __shared__ |
| [0] | 0x00000024 {0.20108646, 0.23432112, 0.2616657, 0.18860439, ...} | float[16] __shared__ |
| [1] | 0x00000064 {0.88125247, 0.21982482, 0.15710929, 0.15753655, ...} | float[16] __shared__ |
| [2] | 0x000000a4 {0.55427718, 0.1802118, 0.76696068, 0.56581318, ...} | float[16] __shared__ |
| [3] | 0x000000e4 {0.60716575, 0.673513, 0.26108584, 0.37244788, ...} | float[16] __shared__ |
| Bs | 0x00000424 {{0.80645162, 0.41080967, 0.12955107, 0.26792198, ...}, {0.1797540 | float[16][16] __shared__ |
| a | 0 | int |
| b | 0 | int |
| bx | 0 | int |
| by | 0 | int |
| tx | 0 | int |

Autos | Locals | Threads | Modules | Watch 1

Memory 1

Address: 0x00000024 | Columns: Auto

```
0x00000024  9c e9 4d 3e e0 f1 6f 3e 0c f9 85 3e 82 21 41 3e 6c e7 35 3f 7f  œéM>àño>.ù.>.!A>lç5?.
0x00000039  63 3f 3f 97 4d 4b 3f 91 59 48 3f 0a e1 04 3e 1f 69 0f 3f fc 11  c??—MK?'YH?.á.>.i.?ü.
0x0000004E  fe 3d 1c d5 0d 3f 5a 3d ad 3e e4 27 72 3f 8a f9 44 3e ca c7 64  þ=.Õ.?Z=->ä'r?ŠùD>ÊÇd
0x00000063  3f c3 99 61 3f c2 19 61 3e 42 e1 20 3e 43 51 21 3e a4 11 52 3e  ?Ã™a?Â.a>Bá >CQ!>¤.R>
0x00000078  eb 43 75 3f 8a ed c4 3e ba dd dc 3e 54 2f 2a 3f 23 b5 91 3e 49  ëCu?ŠíÄ>°ÝÜ>T/*?#µ'>I
0x0000008D  75 24 3f c5 b1 62 3f 9a 3b 4d 3f a9 bf 54 3f 88 33 44 3f 47 65  u$?Å±b?š;M?©¿T?^3D?Ge
0x000000A2  a3 3e 1c e5 0d 3f 71 89 38 3e 89 57 44 3f 22 d9 10 3f 13 71 09  £>.å.?q.8>.WD?"Ù.?.q.
0x000000B7  3e ba c1 dc 3d dc e3 6d 3f 9c c1 cd 3e 76 c1 3a 3e 70 c9 37 3f  >°ÁÜ=ÜãM?œÁÍ>vÁ:>pÉ7?
0x000000CC  21 4d 10 3f d6 37 6b 3f db 7d 6d 3f d9 85 6c 3f 42 3d 21 3f 47  !M.?Ö7k?Û}m?Ù.l?B=!?G
0x000000E1  7d 23 3f 37 6f 1b 3f 59 6b 2c 3f 0b ad 85 3e 7d b1 be 3e b7 69  }#??o.?Yk,?.-.>}±.>·i
0x000000F6  5b 3e 10 35 88 3e 21 6f 10 3f 04 2d 82 3e 89 51 c4 3d 5b 99 ad  [>.5^>!o.?.-.>.QÄ=[™-
0x0000010B  3e 2e 15 97 3e cd ab 66 3f 23 91 11 3f c6 f1 62 3f 20 e9 0f 3e  >..—>Í«f?#'.?Æñb? é.>
0x00000120  7f 49 3f 3f dc 11 ee 3d 84 c9 41 3e da 01 6d 3c d8 dd 6b 3f ee  .I??Ü.î=.ÉA>Ú.m<ØÝk?î
0x00000135  01 f7 3b ee fb 76 3f 2f 81 17 3d 6e f3 36 3f 48 1d a4 3e ef b1  .÷;îûv?/..=nó6?H.¤>ï±
```

Memory 1 | Call Stack | Breakpoints | Output | Pending Checkins

# まとめ

- CPUのIPL性能向上は減速している
- Performance = Parallelism が今後の性能確保の要
- 統合化シェーダエンジンによりGPUは汎用の超並列プロセッサとなる
- CUDA GPU コンピューティングはGPUの並列性能を最大限に引き出し、並列プログラミングの抽象化によりプログラミングも容易にする
- 次世代Fermiアーキテクチャは更にコンピューティング機能、性能を向上した

© 2009 NVIDIA Corporation

Thank you for your attention