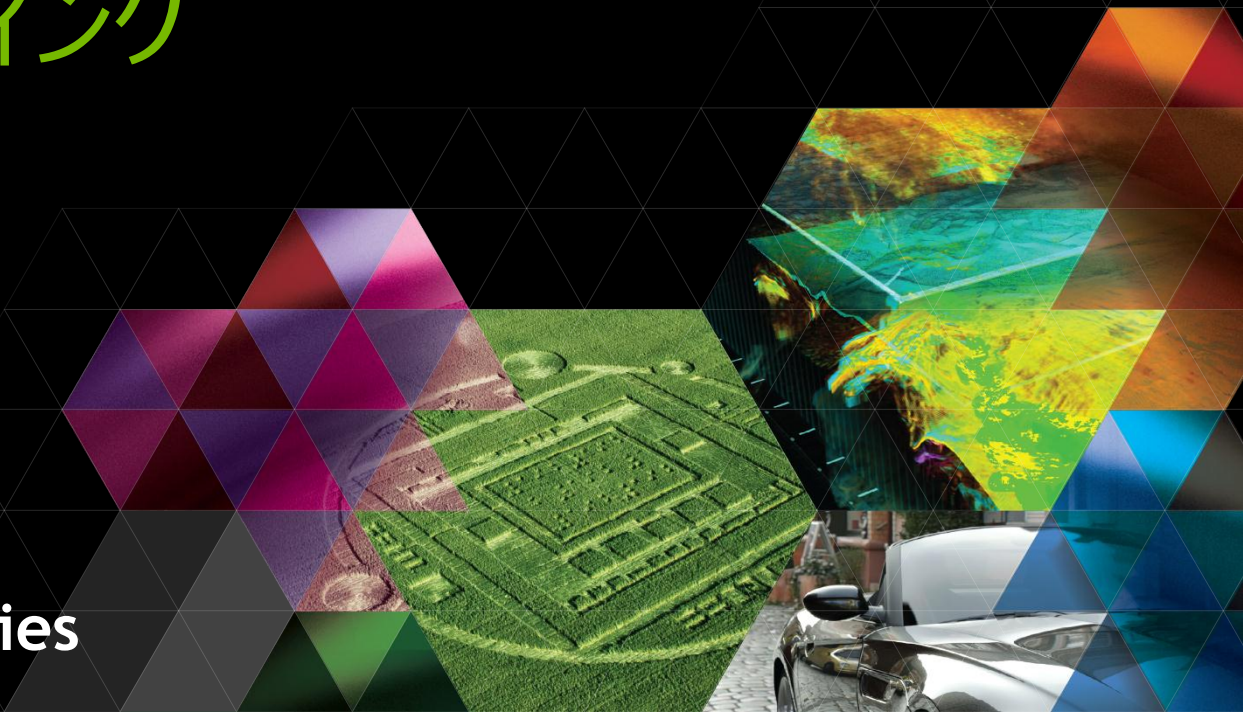
GPU TECHNOLOGY CONFERENCE

NVIDIA

OPENACC・CUDAによる GPUコンピューティング



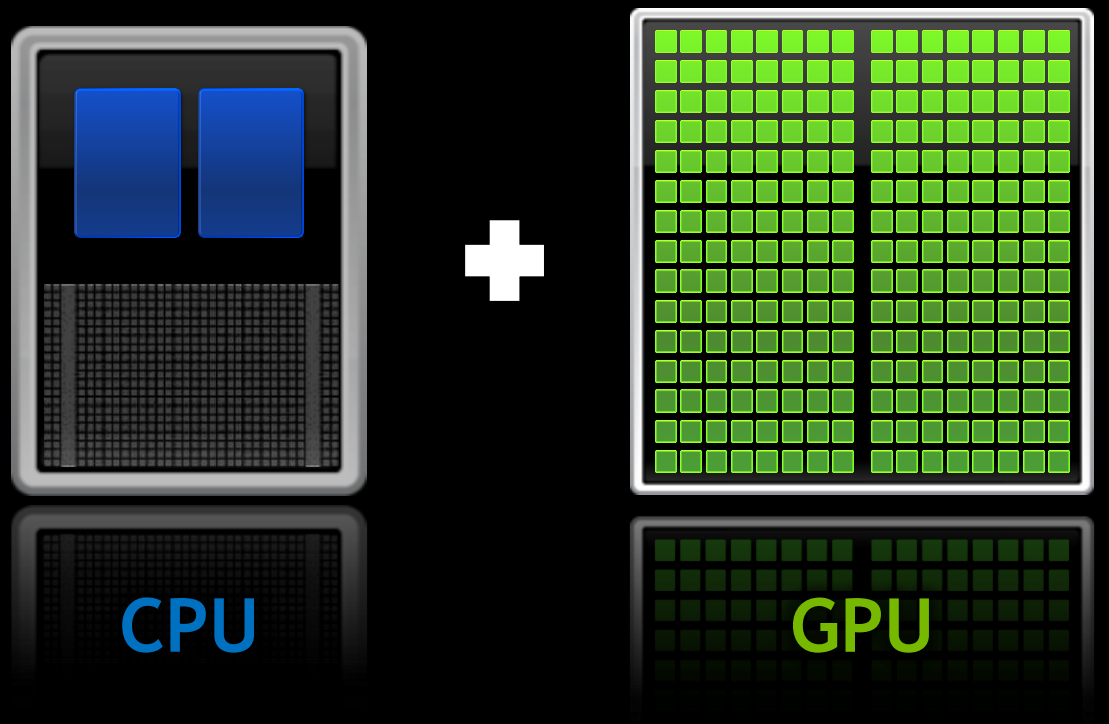
Akira Naruse
NVIDAI Developer Technologies

AGENDA

- GPUコンピューティング、CUDAの概要
- OpenACC

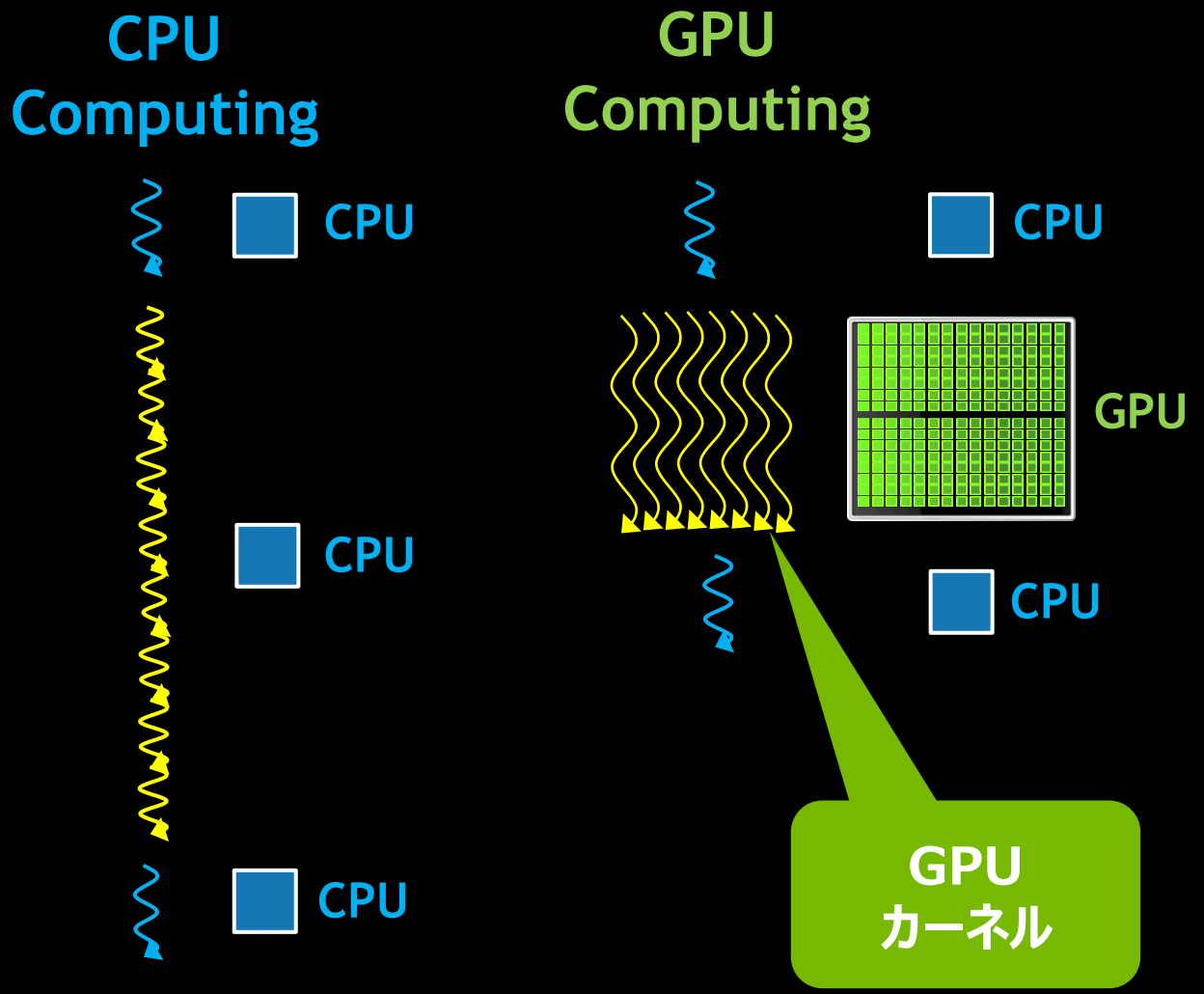
GPUコンピューティング

Low latency + High throughput

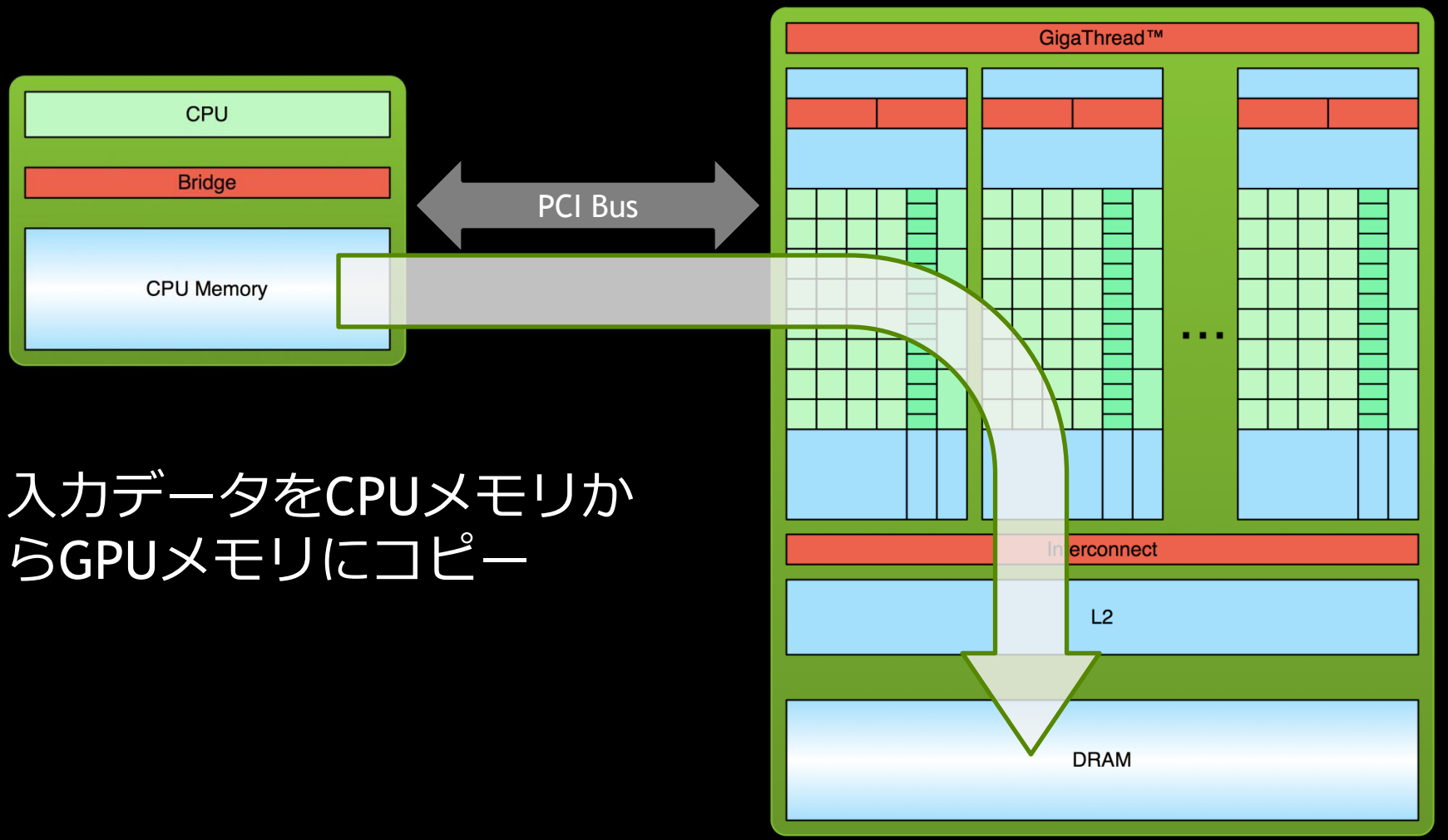


アプリケーション実行の流れ

```
Program myscience

   ... serial code ...

   do k = 1,n1
      do i = 1,n2
         ... parallel code ...
      enddo
   enddo

   ... serial code ...

End Program myscience
```

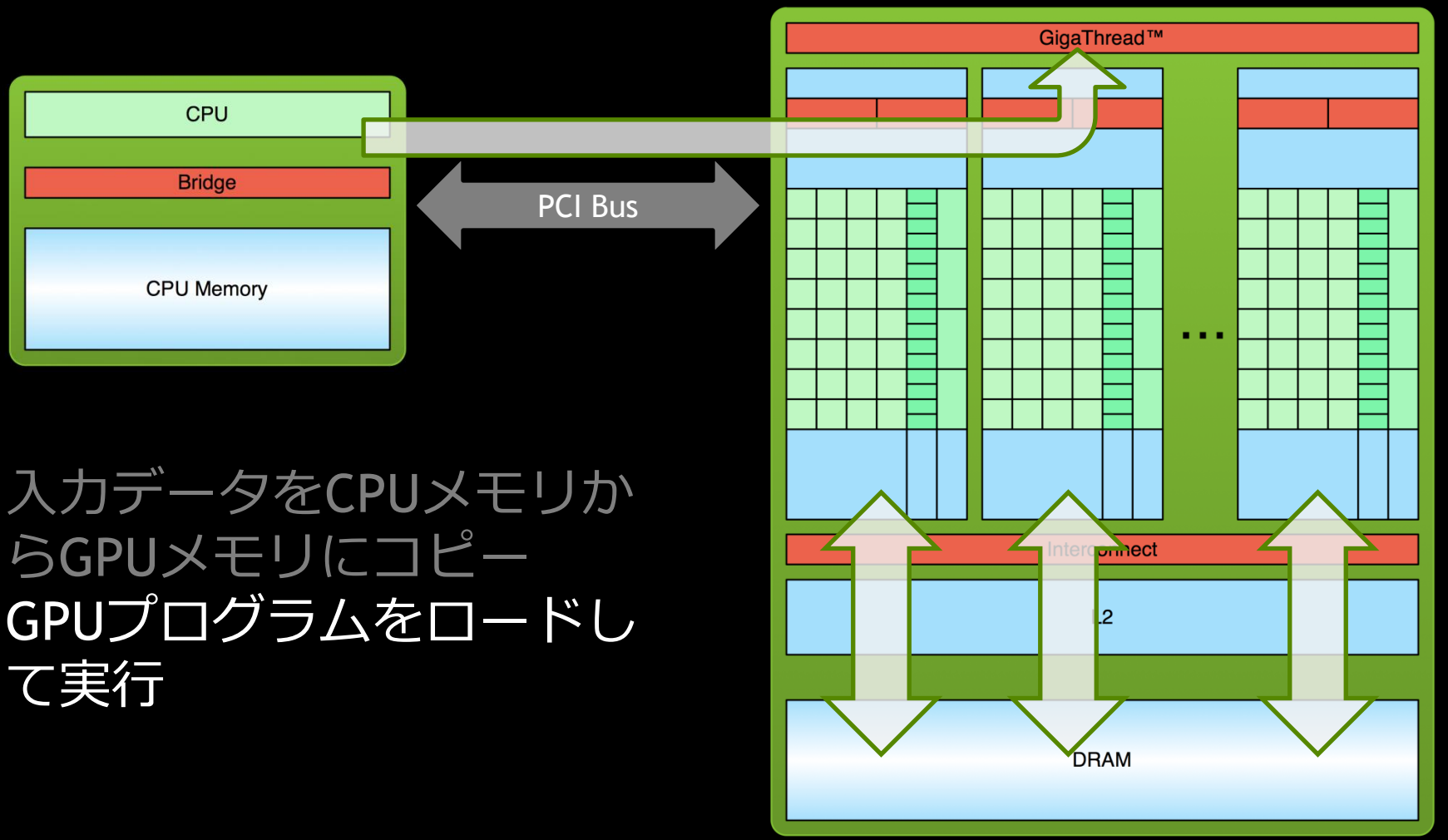


GPUカーネル実行の流れ



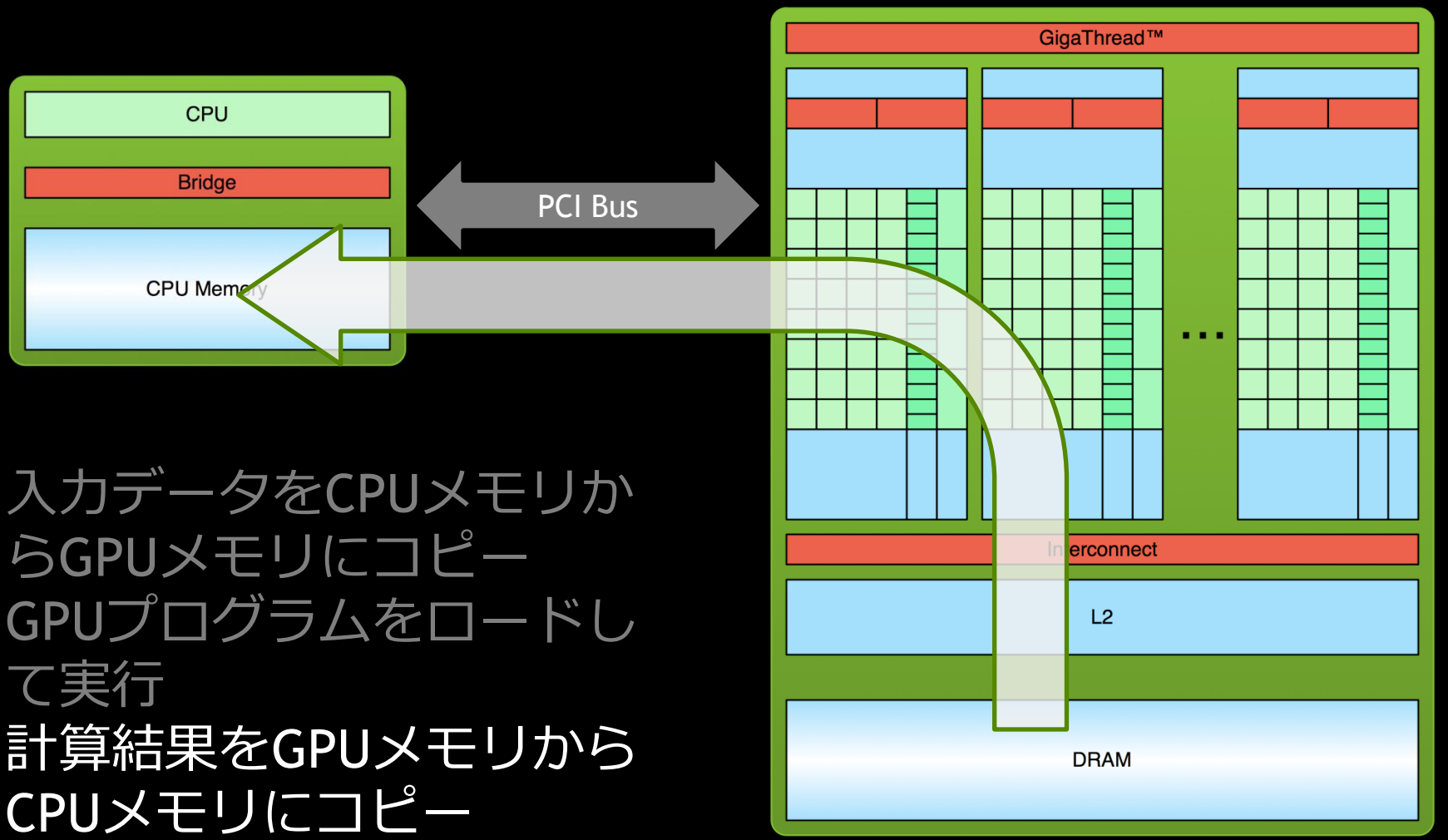
1. 入力データをCPUメモリからGPUメモリにコピー

GPUカーネル実行の流れ



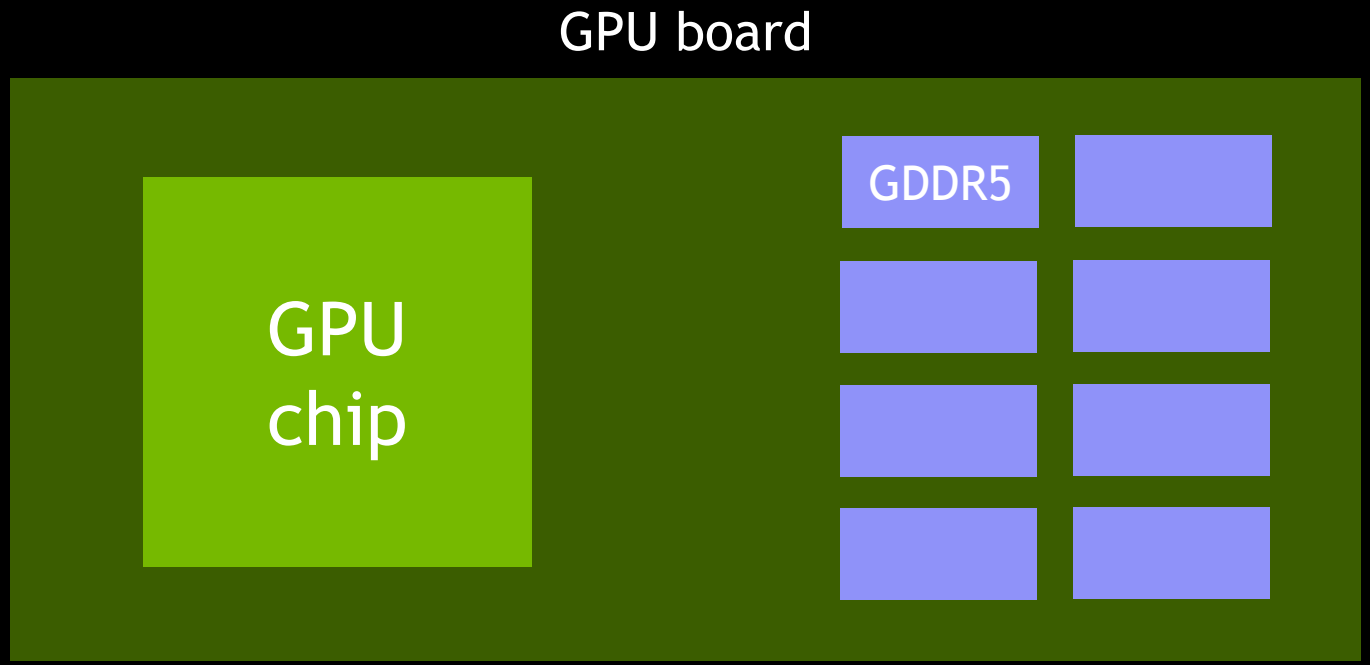
1. 入力データをCPUメモリからGPUメモリにコピー
2. GPUプログラムをロードして実行

GPUカーネル実行の流れ



1. 入力データをCPUメモリからGPUメモリにコピー
2. GPUプログラムをロードして実行
3. 計算結果をGPUメモリからCPUメモリにコピー

GPUの構造



1.43 TFLOPS (DP)
4.29 TFLOPS (SP)

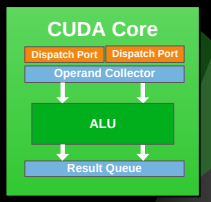
12 GB
288 GB/s

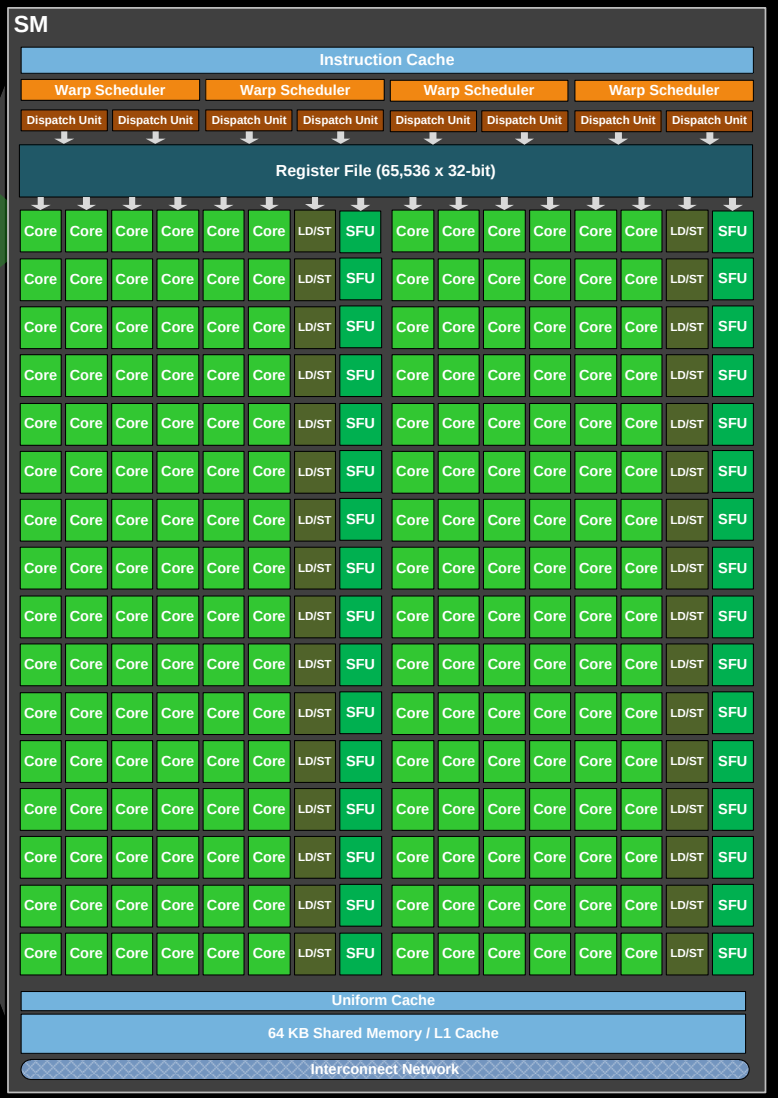
(*) Tesla K40

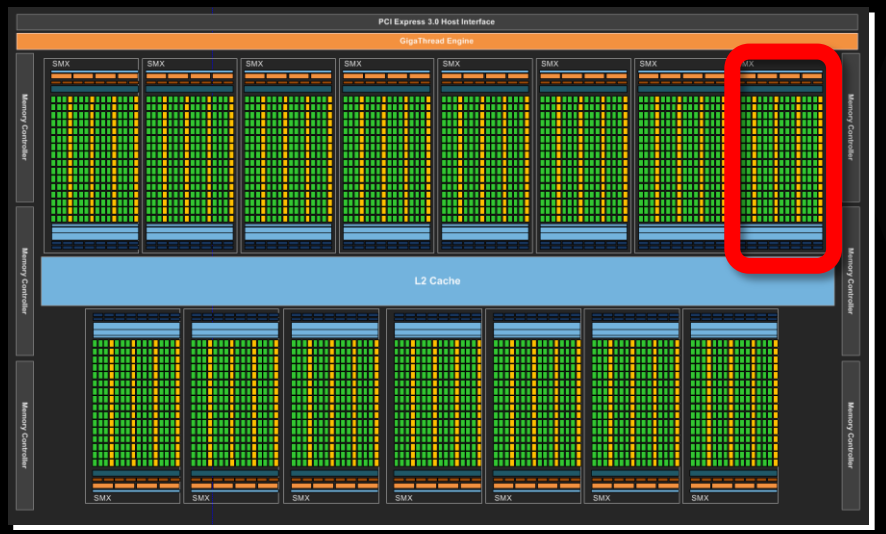
GPUの構造

192 CUDA core/SMX

2880 CUDA core
並列性の抽出が鍵



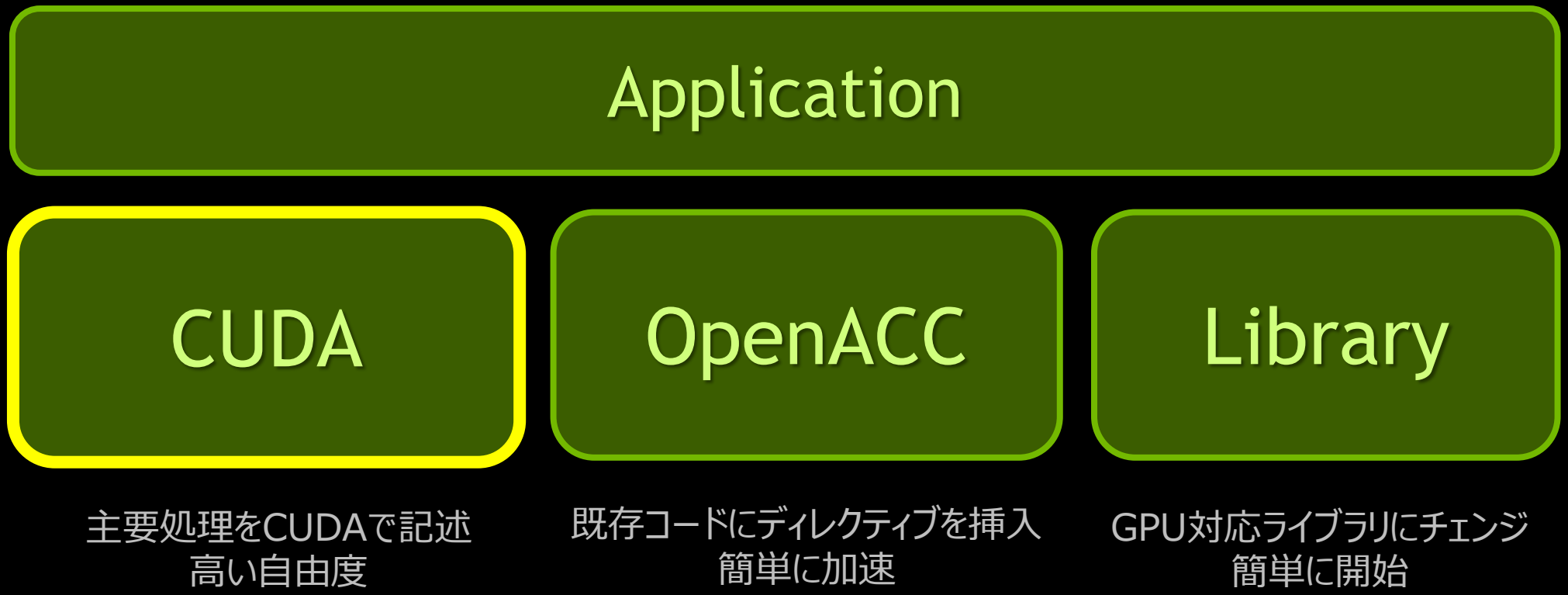




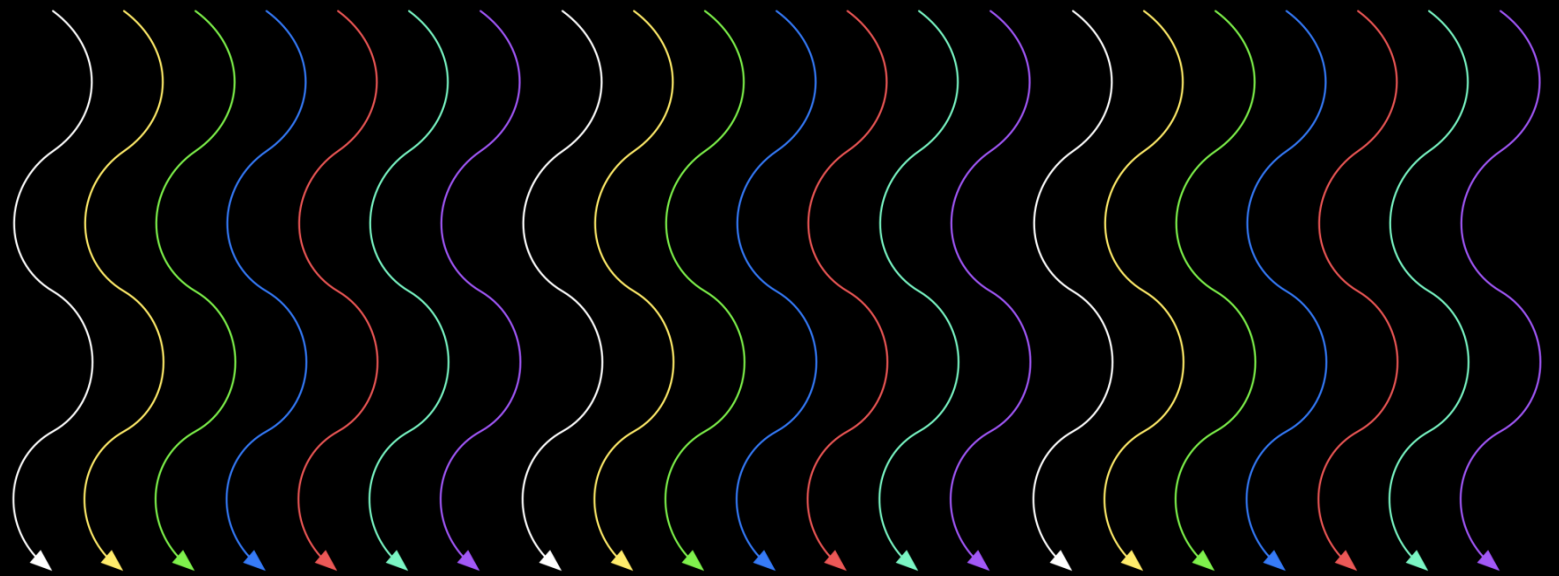
15 SMX/chip

(*) Tesla K40

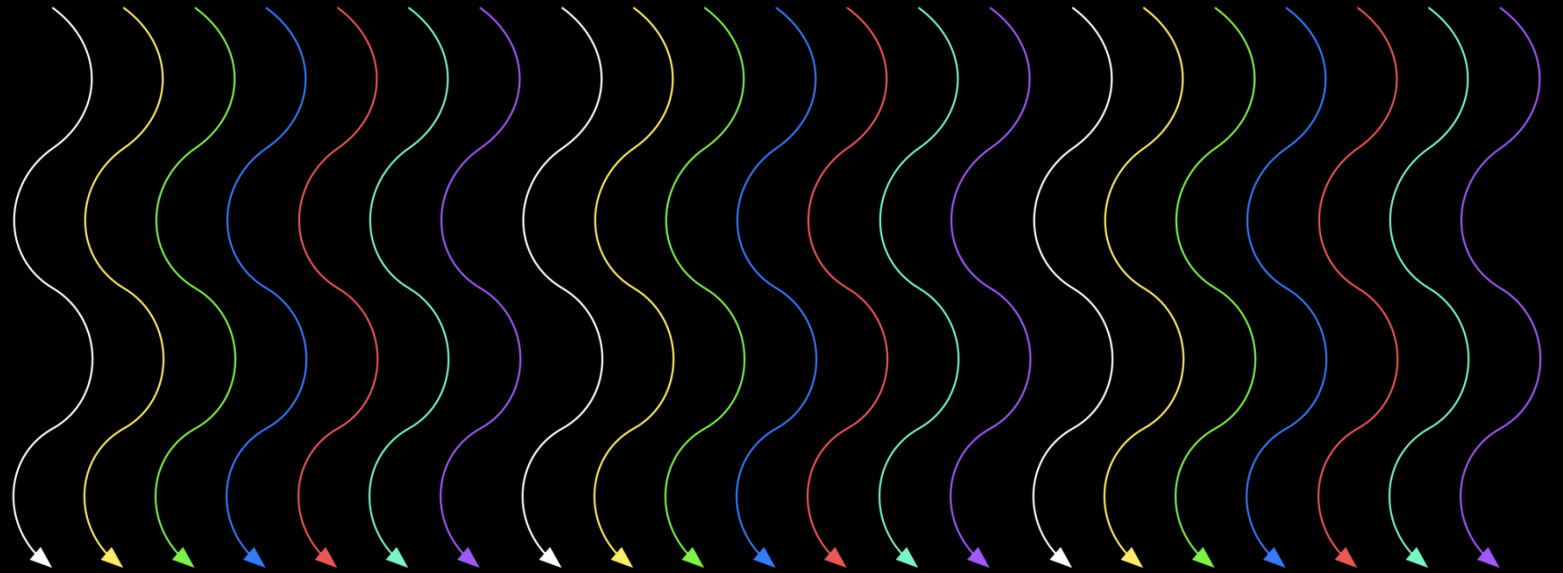
アプリをGPU対応する方法



CUDAプログラミングモデル

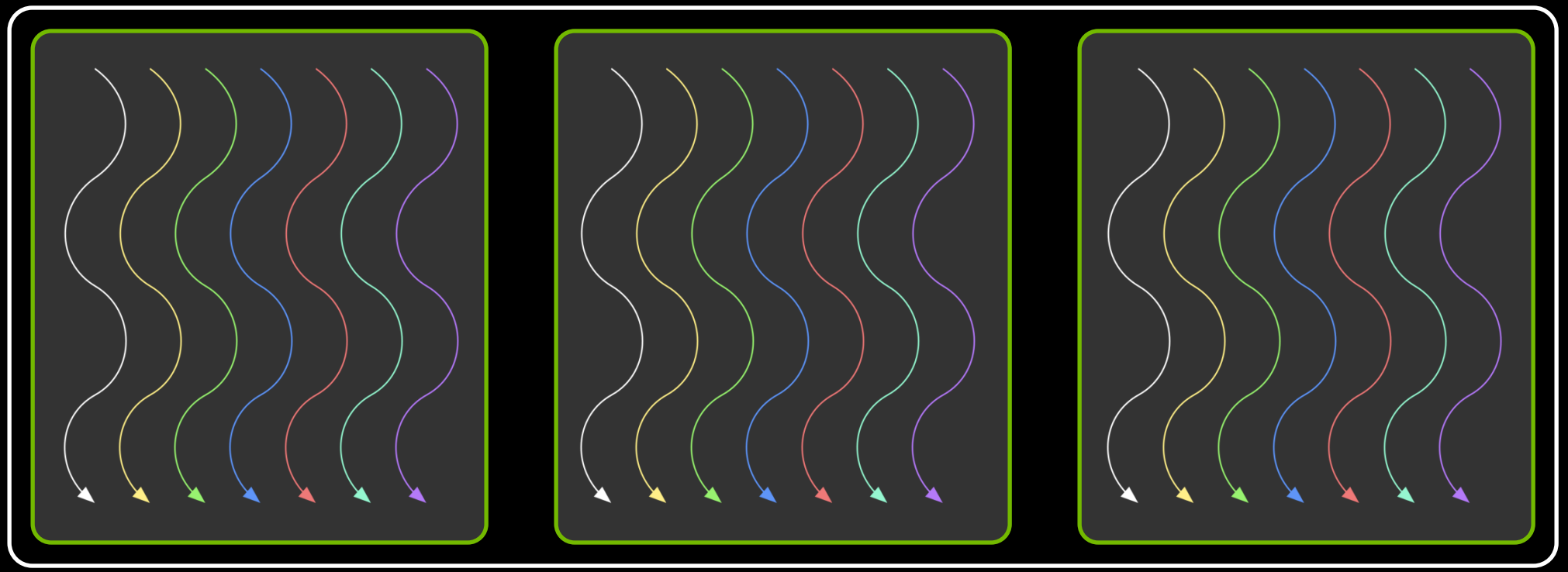


CUDAプログラミングモデル



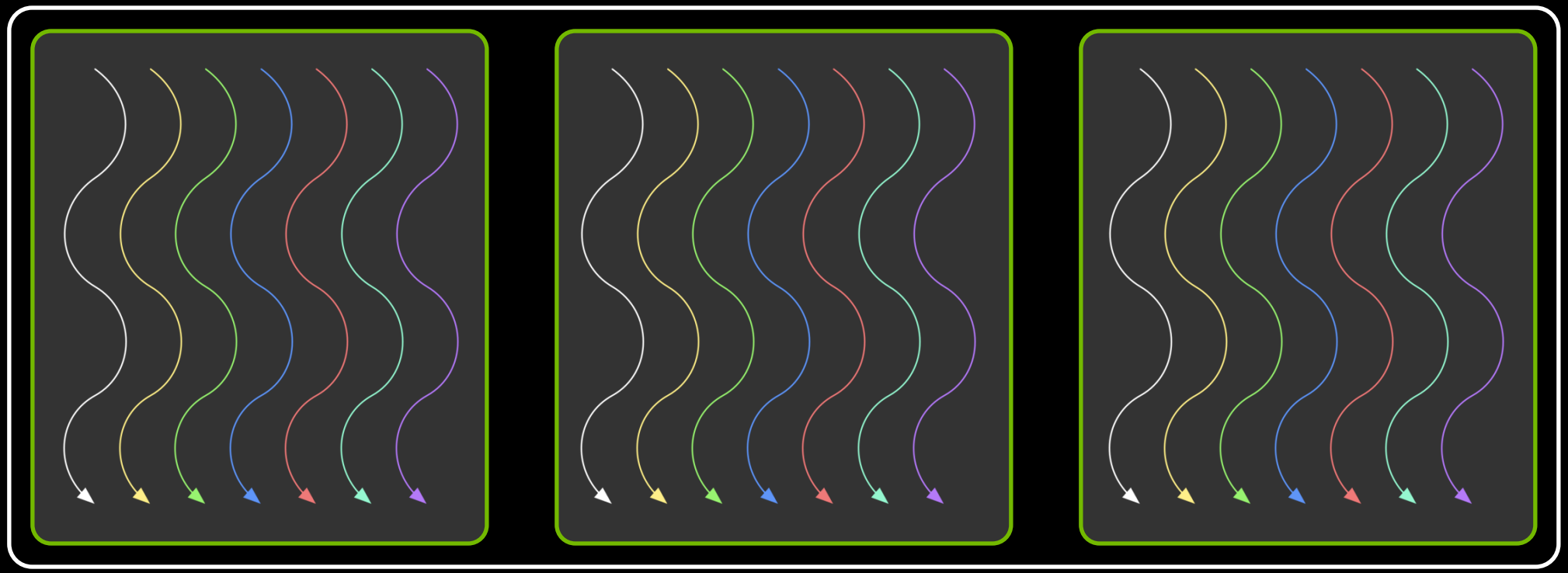
- スレッドの集合がブロック

CUDAプログラミングモデル



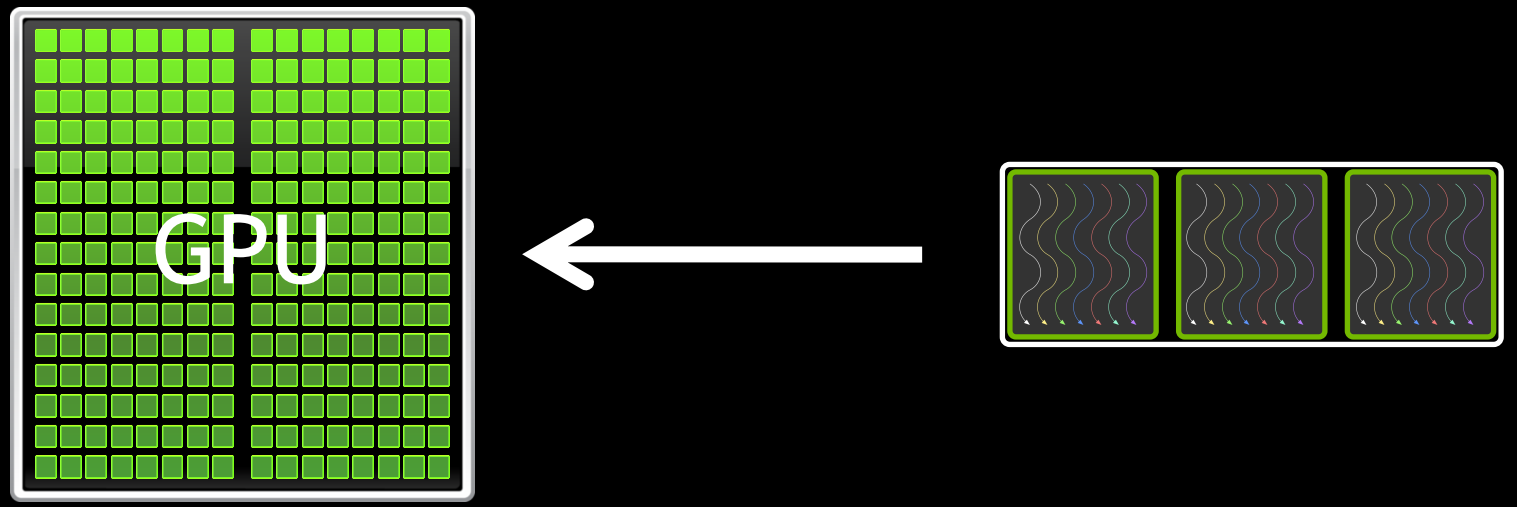
- スレッドの集合がブロック
- ブロックの集合がグリッド

CUDAプログラミングモデル



- スレッドの集合がブロック
- ブロックの集合がグリッド
- CUDAカーネルは1つのグリッドとしてGPU上で実行される

CUDAプログラミングモデル



- スレッドの集合がブロック
- ブロックの集合がグリッド
- CUDAカーネルは1つのグリッドとしてGPU上で実行される

SAXPY (Y=A*X+Y)

CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}




...
saxpy(N, 3.0, x, y);
...
```

SAXPY (Y=A*X+Y)

CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

CUDA

```
__global__ void saxpy(int n, float a,
           float *x, float *y)
{
  int i = threadIdx.x + blodkDim.x * blockIdx;
  if (i < n)
    y[i] += a*x[i];
}

...
size_t size = sizeof(float) * N;
cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/256, 256 >>>(N, 3.0, d_x, d_y);
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
...
```

SAXPY (Y=A*X+Y)

CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}
```

データ転送(H2D)

カーネル投入 <<<GS,BS>>>
GS: グリッドサイズ(ブロック数)
BS: ブロックサイズ(スレッド数)

データ転送(D2H)

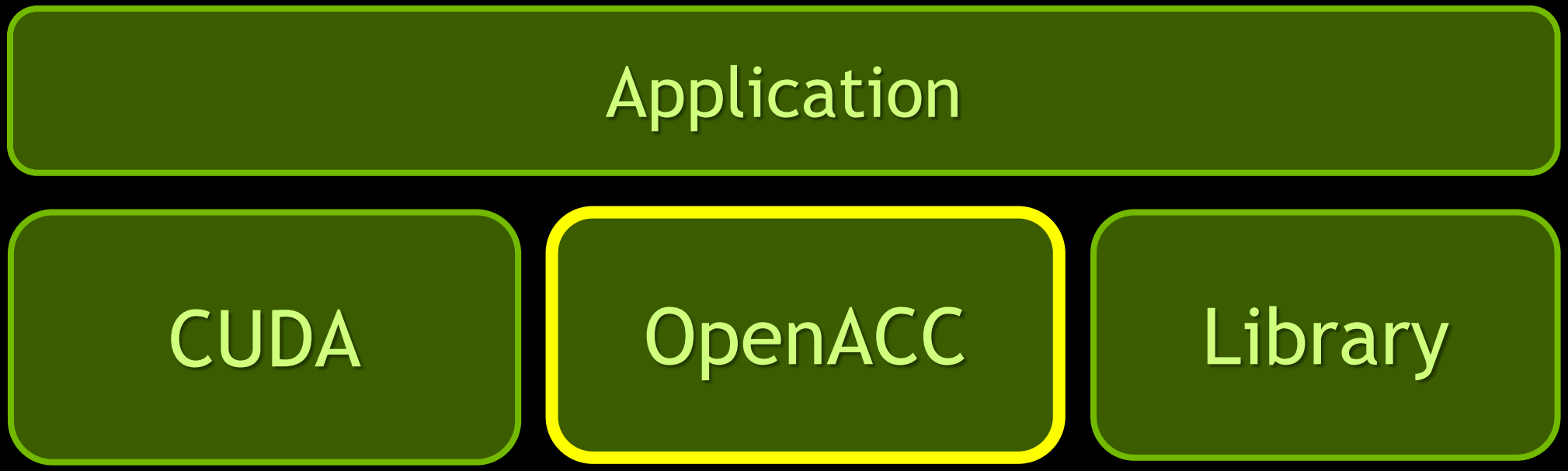
GPUカーネル

CUDA

```
__global__ void saxpy(int n, float a,
           float *x, float *y)
{
  int i = threadIdx.x + blodkDim.x * blockIdx;
  if (i < n)
    y[i] += a*x[i];
}

...
size_t size = sizeof(float) * N;
cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/256, 256 >>>(N, 3.0, d_x, d_y);
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
...
```

threadIdx: スレッドID
blockDim: ブロックサイズ
blockIdx: ブロックID

アプリをGPU対応する方法

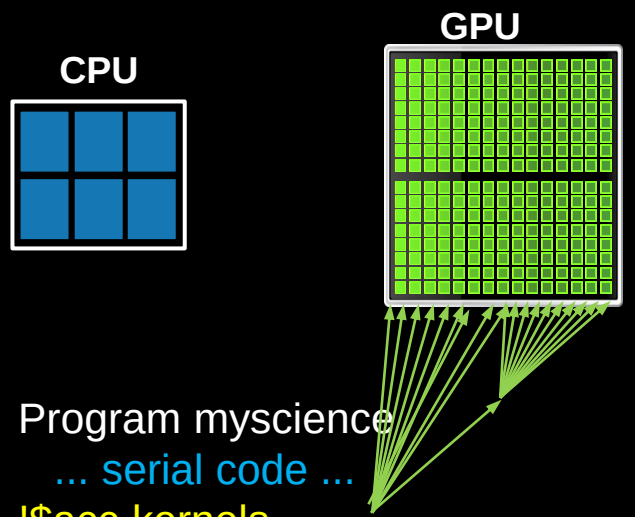


主要処理をCUDAで記述
高い自由度

既存コードにディレクティブを挿入
簡単に加速

GPU対応ライブラリにチェンジ
簡単に開始

OPENACC




```
Program myscience
   ... serial code ...
!$acc kernels
   do k = 1,n1
      do i = 1,n2
         ... parallel code ...
      enddo
    enddo
!$acc end kernels
   ... serial code …
End Program myscience
```

ヒント
の追加

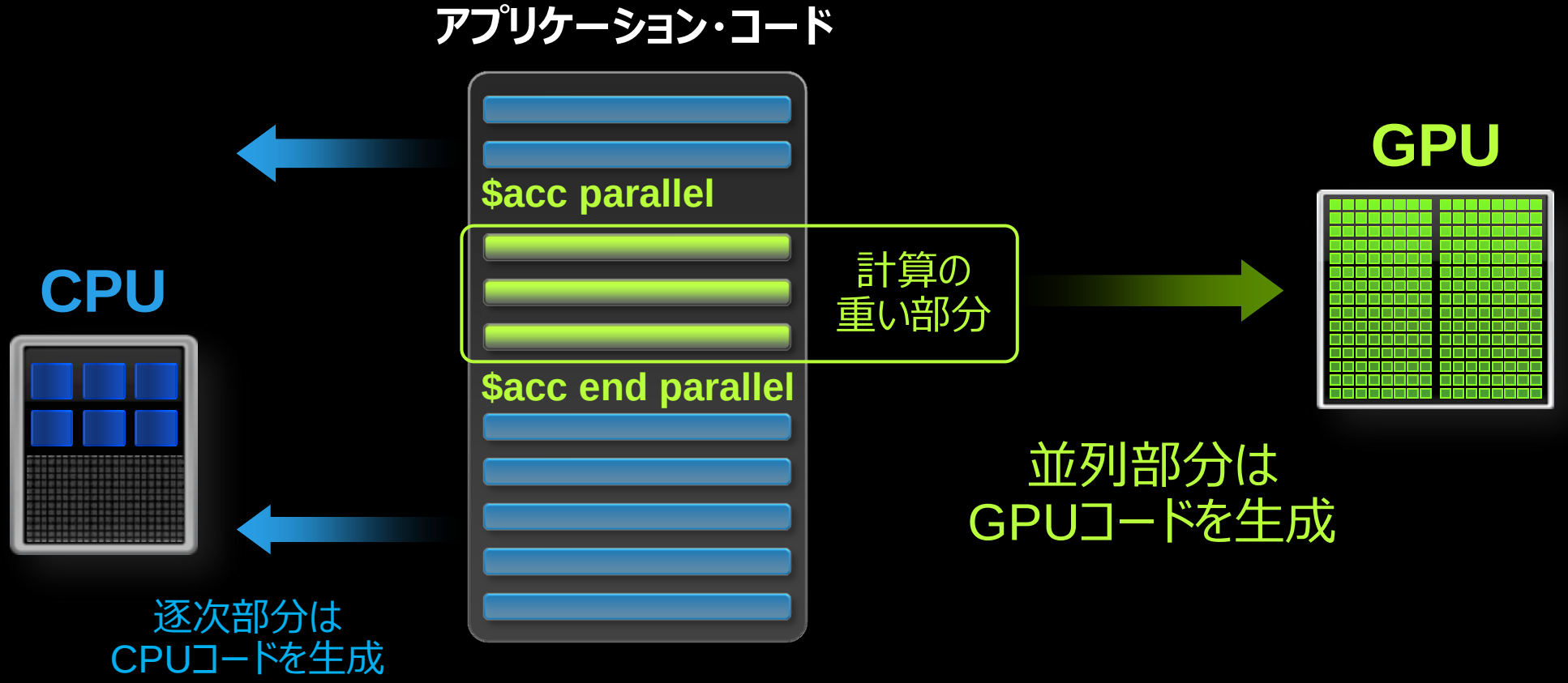
既存のC/Fortranコード

簡単: 既存のコードに
コンパイラへのヒントを追加

強力: そこそこの労力で、コンパイラが
コードを自動で並列化

オープン: 複数コンパイラベンダが、
複数アクセラレータをサポート
NVIDIA, AMD, Intel(予定)

実行モデル



SAXPY (Y=A*X+Y)

CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

CUDA

```
__global__ void saxpy(int n, float a,
           float *x, float *y)
{
  int i = threadIdx.x + blodkDim.x * blockIdx;
  if (i < n)
    y[i] += a*x[i];
}

...
size_t size = sizeof(float) * N;
cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/256, 256 >>>(N, 3.0, d_x, d_y);
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
...
```

SAXPY (Y=A*X+Y)

OpenMP

```
void saxpy(int n,
           float a,
           float *x,
           float *restrict y)
{
#pragma omp parallel for
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

OpenACC

```
void saxpy(int n,
           float a,
           float *x,
           float *restrict y)
{
#pragma acc parallel copy(y[:n]) copyin(x[:n])
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

SAXPY (Y=A*X+Y, FORTRAN)

OpenMP

```
subroutine saxpy(n, a, X, Y)
  real :: a, X(:), Y(:)
  integer :: n, i

  !$omp parallel do
  do i=1,n
    Y(i) = a*X(i)+Y(i)
  enddo
  !$omp end parallel do
end subroutine saxpy


...
call saxpy(N, 3.0, x, y)
...
```

OpenACC

```
subroutine saxpy(n, a, X, Y)
  real :: a, Y(:), Y(:)
  integer :: n, i

  !$acc parallel copy(Y(:)) copyin(X(:))
  do i=1,n
    Y(i) = a*X(i)+Y(i)
  enddo
  !$acc end parallel
end subroutine saxpy


...
call saxpy(N, 3.0, x, y)
...
```

OPENMPとの併用

OpenMP / OpenACC

```
void saxpy(int n, float a,
           float *x,
           float *restrict y)
{
#pragma acc parallel copy(y[:n]) copyin(x[:n])
#pragma omp parallel for
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

簡単にコンパイル

OpenMP / OpenACC

```
void saxpy(int n, float a,
           float *x,
           float *restrict y)
```

```
$ pgcc -Minfo -acc saxpy.c
saxpy:
     16, Generating present_or_copy(y[:n])
         Generating present_or_copyin(x[:n])
         Generating Tesla code
     19, Loop is parallelizable
         Accelerator kernel generated
         19, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

```
...
```

簡単に実行

OpenMP / OpenACC

```
void saxpy(int n, float a,
           float *x,
           float *restrict y)
...
```

```
$ nvprof ./a.out
==10302== NVPROF is profiling process 10302, command: ./a.out
==10302== Profiling application: ./a.out
==10302== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 62.95%  3.0358ms         2  1.5179ms  1.5172ms  1.5186ms  [CUDA memcpy HtoD]
 31.48%  1.5181ms         1  1.5181ms  1.5181ms  1.5181ms  [CUDA memcpy DtoH]
  5.56%  268.31us         1  268.31us  268.31us  268.31us  saxpy_19_gpu
```

簡単に高速

Automotive

Real-Time Object Detection

Global Manufacturer of Navigation Systems

40時間で5倍

Financial

Valuation of Stock Portfolios using Monte Carlo

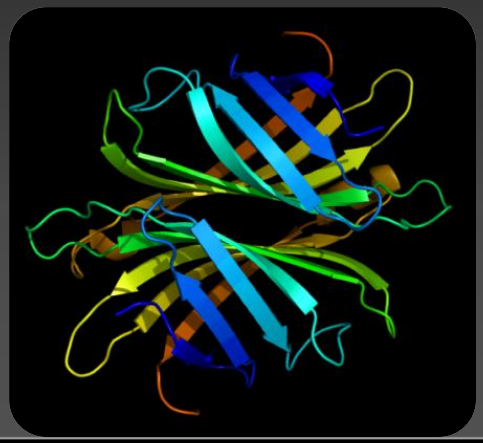
Global Technology Consulting Company

4時間で2倍

Life Science

Interaction of Solvents and Biomolecules

University of Texas at San Antonio



8時間で5倍

コンパイラとツール

2013年10月～

2013年12月～

2014年1月～

2015年(予定)

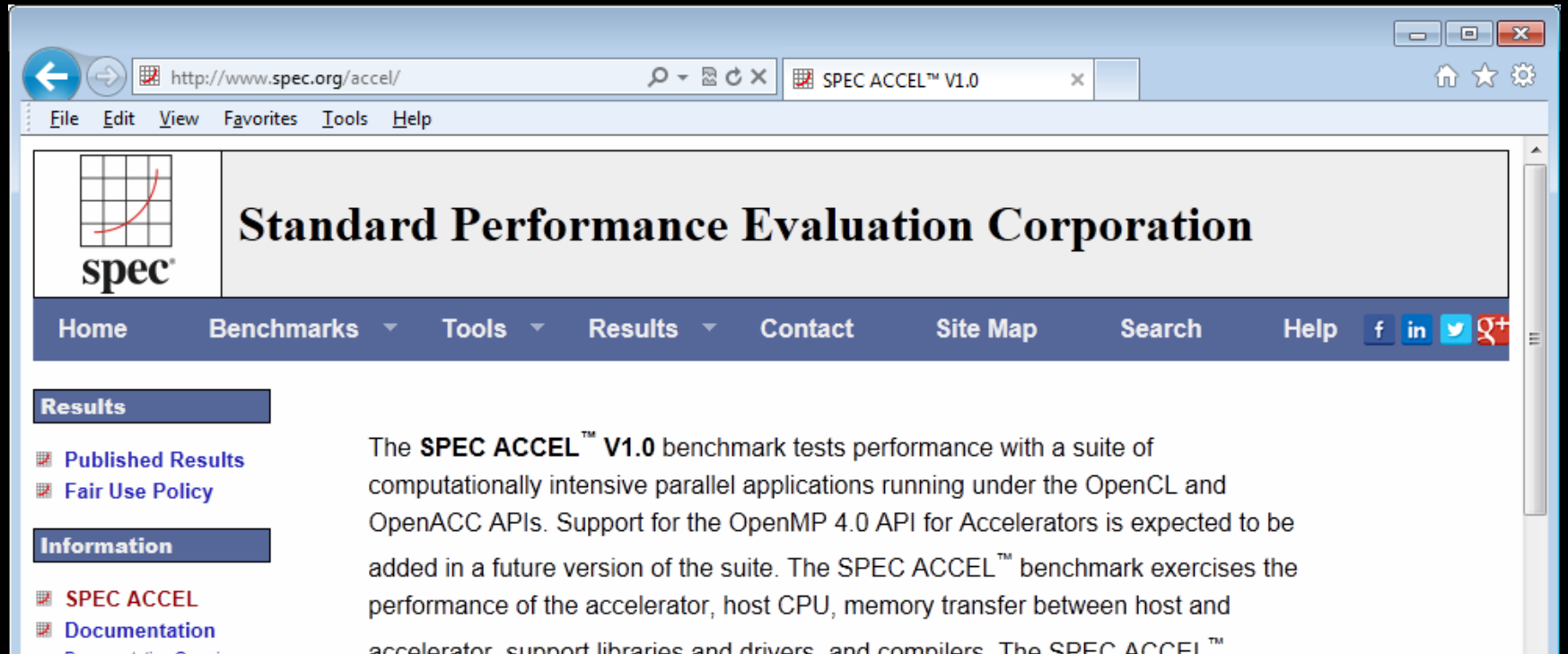
コンパイラ　　OpenACC 2.0対応

デバッグツール

SPEC ACCEL

- 15本のOpenACCベンチマーク

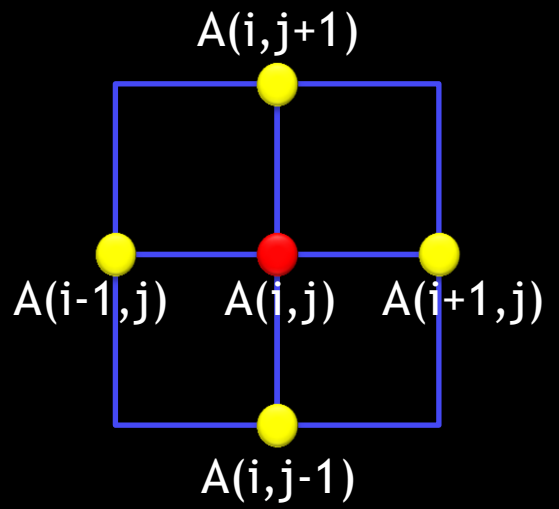
www.spec.org/accel



OPENACCでどこまで出来るの？

例: JACOBI ITERATION

```
while ( error > tol ) {
  error = 0.0;

  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]));
    }
  }

  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
}
```



並列領域 (KERNELS CONSTRUCT)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  #pragma acc kernels
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
}
```

- Parallels と Kernels
  - 並列領域を指示
- Parallels
  - 並列実行スタート
- Kernels
  - 複数のカーネル

並列領域 (KERNELS CONSTRUCT)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }
```

- Parallels と Kernels
  - 並列領域を指示
- Parallels
  - 並列走行の開始
- Kernels

```
$ pgcc -Minfo=acc -acc jacobi.c
jacobi:
     60, Loop carried scalar dependence for 'error' at line 64
         ...
         Accelerator scalar kernel generated
     61, Loop carried scalar dependence for 'error' at line 64
         ...
         Accelerator scalar kernel generated
```

リダクション (REDUCTION CLAUSE)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  #pragma acc kernels
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
}
```

- 演算の種類

+	和
*	積
Max	最大
Min	最小
|	ビット和
&	ビット積
^	XOR
||	論理和
&&	論理積

リダクション (REDUCTION CLAUSE)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
```

- 演算の種類

+	和
*	積
Max	最大

```
$ pgcc -Minfo=acc -acc jacobi.c
jacobi:
     59, Generating present_or_copyout(Anew[1:4094][1:4094])
         Generating present_or_copyin(A[:][:])
         Generating Tesla code
     61, Loop is parallelizable
     63, Loop is parallelizable
         Accelerator kernel generated
         61, #pragma acc loop gang /* blockIdx.y */
         63, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
             Max reduction generated for error
```

データ転送方法 (DATA CLAUSE)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
```

```
$ pgcc -Minfo=acc -acc jacobi.c
jacobi:
     59, Generating present_or_copyout(Anew[1:4094][1:4094])
         Generating present_or_copyin(A[:][:])
         Generating Tesla code
     61, Loop is parallelizable
     63, Loop is parallelizable
         Accelerator kernel generated
         61, #pragma acc loop gang /* blockIdx.y */
         63, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
             Max reduction generated for error
```

データ転送方法 (DATA CLAUSE)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels \
      pcopyout(Anew[1:N-2][1:M-2]) pcopyin(A[0:N][0:M])
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  #pragma acc kernels \
      pcopyout(A[1:N-2][1:M-2]) pcopyin(Anew[1:N-2][1:M-2])
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
```

- copyin (Host→GPU)
- copyout (Host←GPU)
- copy
- create
- present

- pcopyin
- pcopyout
- pcopy
- pcreate

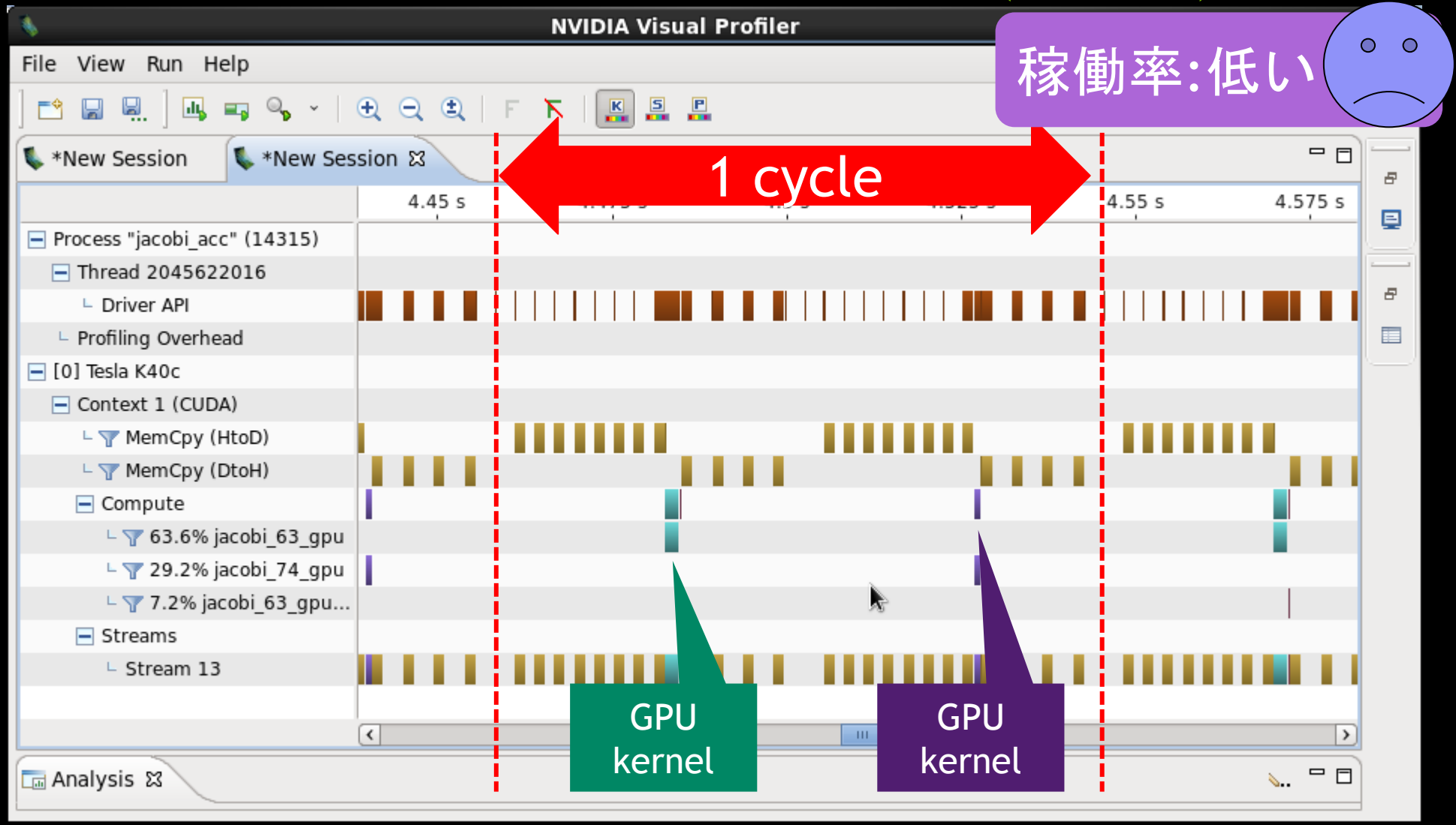
データ転送方法 (DATA CLAUSE)

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels \
      pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  #pragma acc kernels \
      pcopy(A[:][:]) pcopyin(Anew[:][:])
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
```

- copyin (Host→GPU)
- copyout (Host←GPU)
- copy
- create
- present

- pcopyin
- pcopyout
- pcopy
- pcreate

データ転送がボトルネック (NVVP)



過剰なデータ転送

Host

```
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels \
      pcopy(Anew[:][:]) \
      pcopyin(A[:][:])
  {




  }

  #pragma acc kernels \
      pcopy(A[:][:]) \
      pcopyin(Anew[:][:])
  {


  }
}
```

copyin

copyout

copyin

copyout

GPU

```
#pragma acc loop reduction(max:error)
for (int j = 1; j < N-1; j++) {
  #pragma acc loop reduction(max:error)
  for (int i = 1; i < M-1; i++) {
    Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                  A[j-1][i] + A[j+1][i]) * 0.25;
    error = max(error, abs(Anew[j][i] - A[j][i]));
  }
}
```

```
for (int j = 1; j < N-1; j++) {
  for (int i = 1; i < M-1; i++) {
    A[j][i] = Anew[j][i];
  }
}
```

データ領域 (DATA CONSTRUCT)

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  #pragma acc kernels pcopy(A[:][:]) pcopyin(Anew[:][:])
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      A[j][i] = Anew[j][i];
    }
  }
}
```

- copyin (CPU→GPU)
- copyout (CPU←GPU)
- copy
- create
- present

- pcopyin
- pcopyout
- pcopy
- pcreate

適正なデータ転送

Host

```
#pragma acc data \
    pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels \
      pcopy(Anew[:][:]) \
      pcopyin(A[:][:])
  {









  }


  #pragma acc kernels \
      pcopy(A[:][:]) \
      pcopyin(Anew[:][:])
  {

  }
}
```

copyin

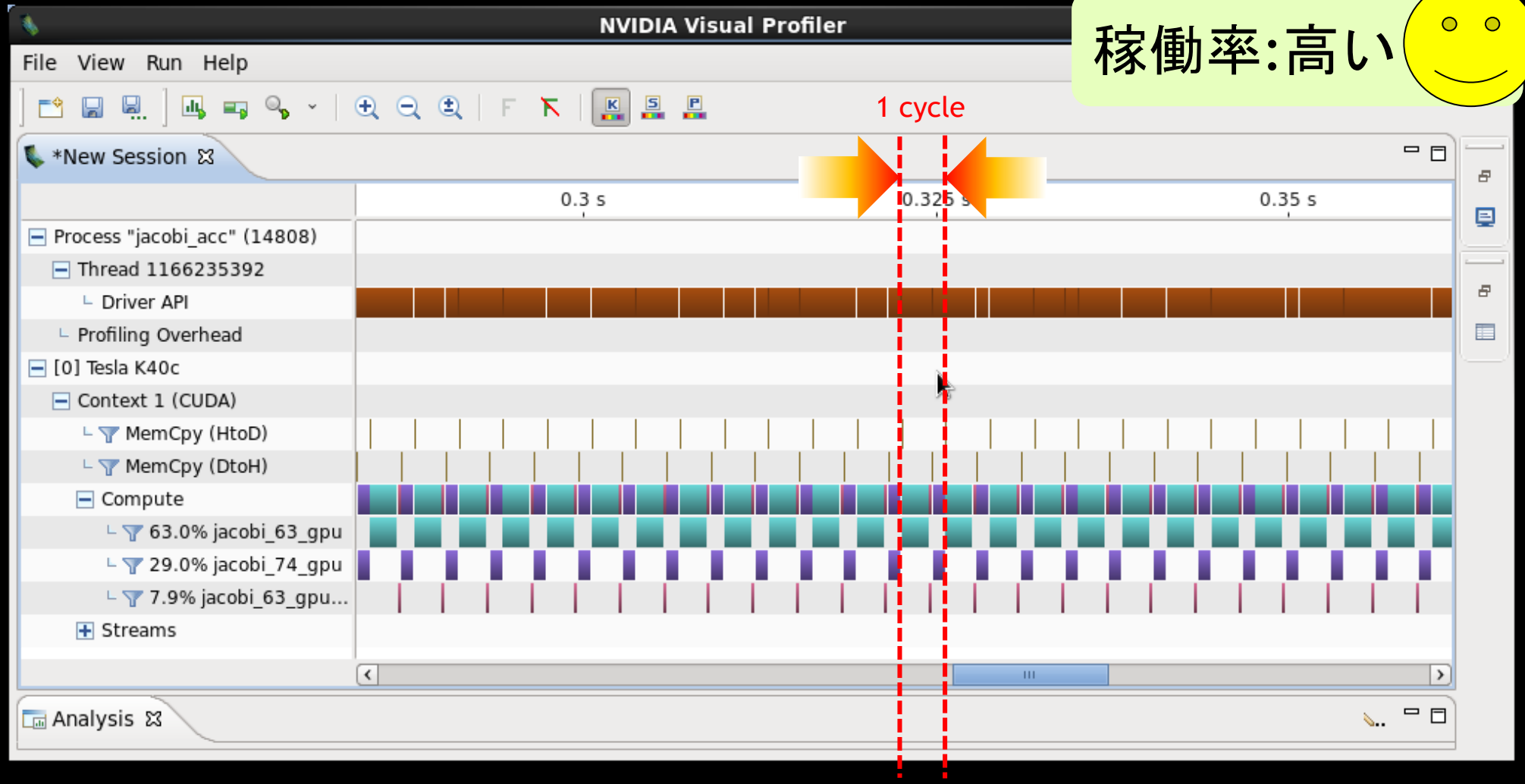
GPU

```
#pragma acc loop reduction(max:error)
for (int j = 1; j < N-1; j++) {
  #pragma acc loop reduction(max:error)
  for (int i = 1; i < M-1; i++) {
    Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                  A[j-1][i] + A[j+1][i]) * 0.25;
    error = max(error, abs(Anew[j][i] - A[j][i]));
  }
}
```

```
for (int j = 1; j < N-1; j++) {
  for (int i = 1; i < M-1; i++) {
    A[j][i] = Anew[j][i];
  }
}
```

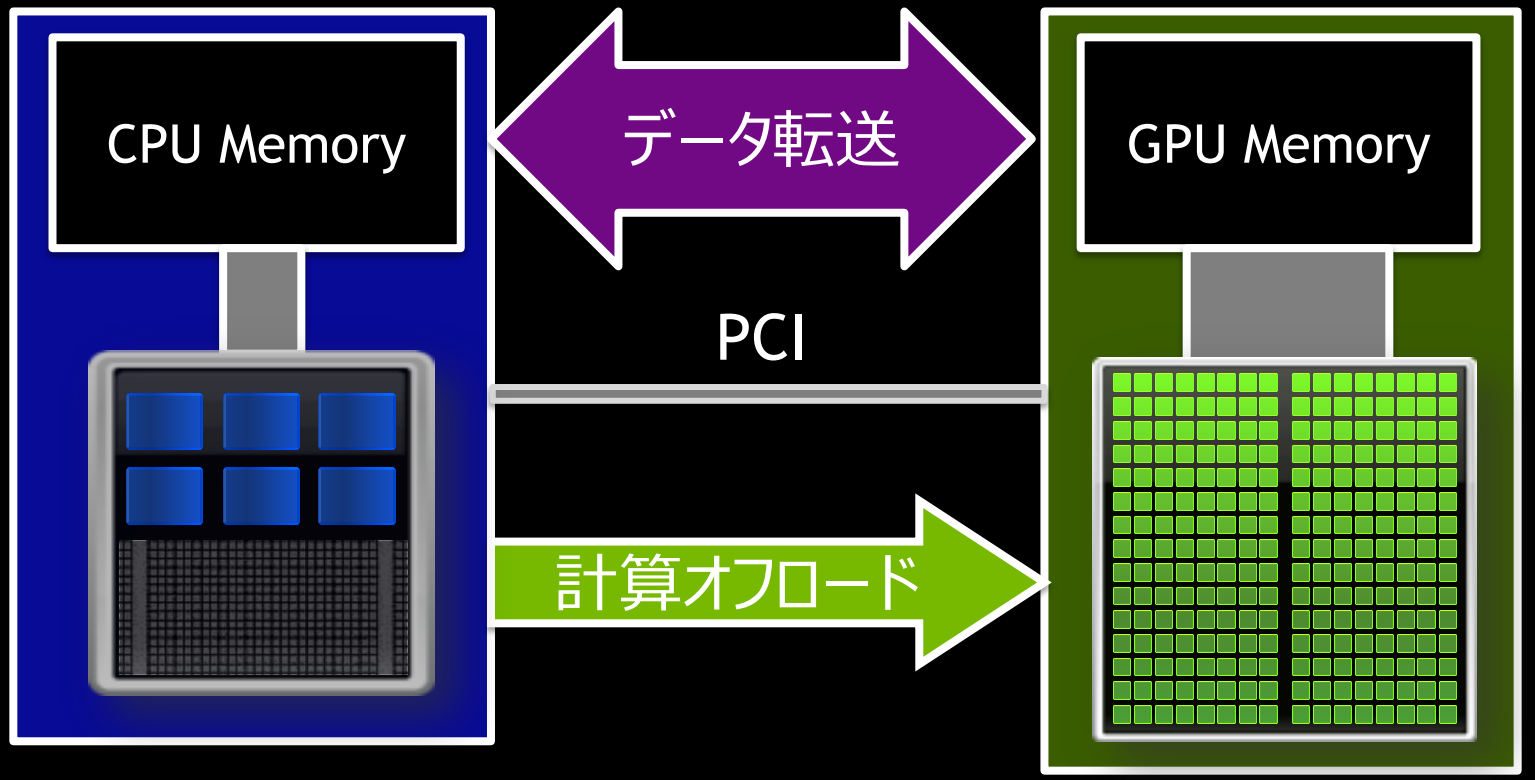
copyout

データ転送の削減 (NVVP)



稼働率:高い

2つの処理



計算オフロード、データ転送、両方を考慮する必要がある

カーネルチューニング

カーネルチューニング (LOOP CONSTRUCT)

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  ...

}
```

カーネルチューニング (LOOP CONSTRUCT)

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error)
```

- Gang
- Worker
- Vector … SIMD幅
- Independent

```
$ pgcc -Minfo=acc -acc jacobi.c
jacobi:
     59, Generating present_or_copyout(Anew[1:4094][1:4094])
         Generating present_or_copyin(A[:][:])
         Generating Tesla code
     61, Loop is parallelizable
     63, Loop is parallelizable
         Accelerator kernel generated
         61, #pragma acc loop gang /* blockIdx.y */
         63, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
             Max reduction generated for error
```

カーネルチューニング (LOOP CONSTRUCT)

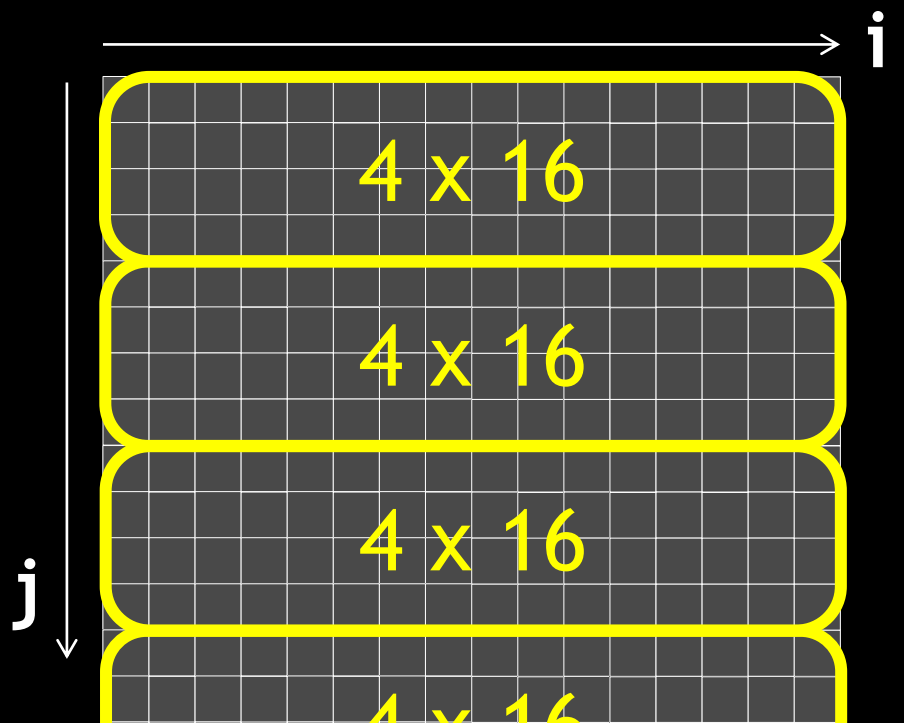
```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error) gang vector(1)
  for (int j = 1; j < N-1; j++) {
    #pragma acc loop reduction(max:error) gang vector(128)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  ...

}
```

- Gang
- Worker
- Vector … SIMD幅

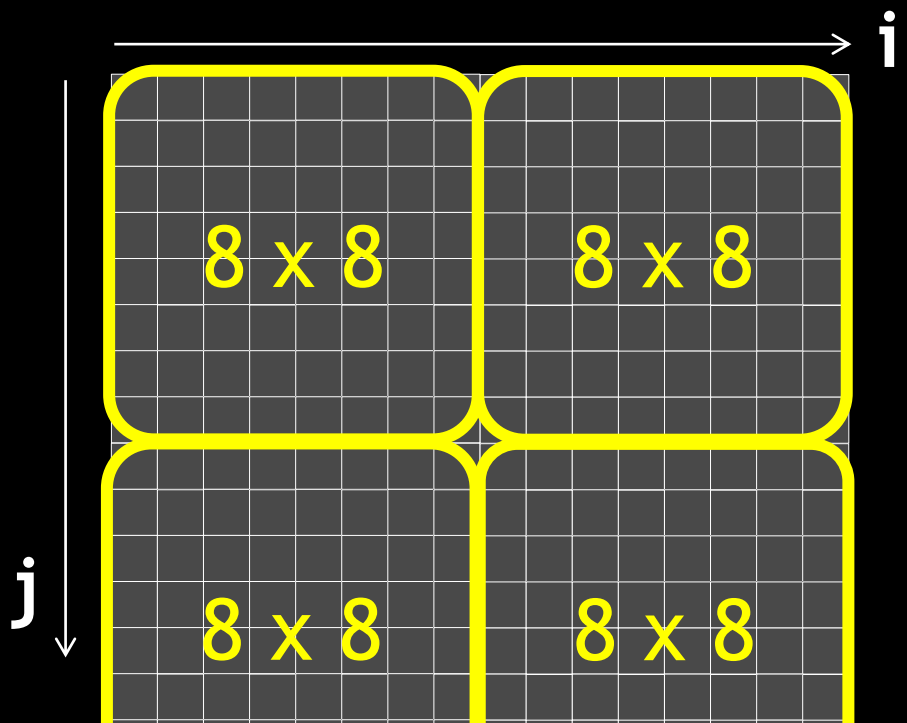
- Collapse
- Independent
- Seq
- Cache
- Tile

実行条件設定 (VECTOR CLAUSE)

```
#pragma acc loop gang vector(4)
for (j = 0; j < 16; j++) {
    #pragma accloop gang vector(16)
    for (i = 0; i < 16; i++) {
        ...
```




```
#pragma acc loop gang vector(8)
for (j = 1; j < 16; j++) {
    #pragma accloop gang vector(8)
    for (i = 0; i < 16; i++) {
        ...
```



カーネルチューニング (LOOP CONSTRUCT)

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error) \
      collapse(2) gang vector(128)
  for (int j = 1; j < N-1; j++) {
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }
  ...
}
```

- Gang
- Worker
- Vector … SIMD幅

- Collapse
- Independent
- Seq
- Cache
- Tile
- …

カーネルチューニング (LOOP CONSTRUCT)

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  error = 0.0;

  #pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
  #pragma acc loop reduction(max:error) independent
  for (int jj = 1; jj < NN-1; jj++) {
    int j = list_j[jj];
    #pragma acc loop reduction(max:error)
    for (int i = 1; i < M-1; i++) {
      Anew[j][i] = (A[j][i+1] + A[j][i-1] +
                    A[j-1][i] + A[j+1][i]) * 0.25;
      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }
  ...
}
```

- Gang
- Worker
- Vector … SIMD幅

- Collapse
- Independent
- Seq
- Cache
- Tile
- …

カーネルチューニング (LOOP CONSTRUCT)

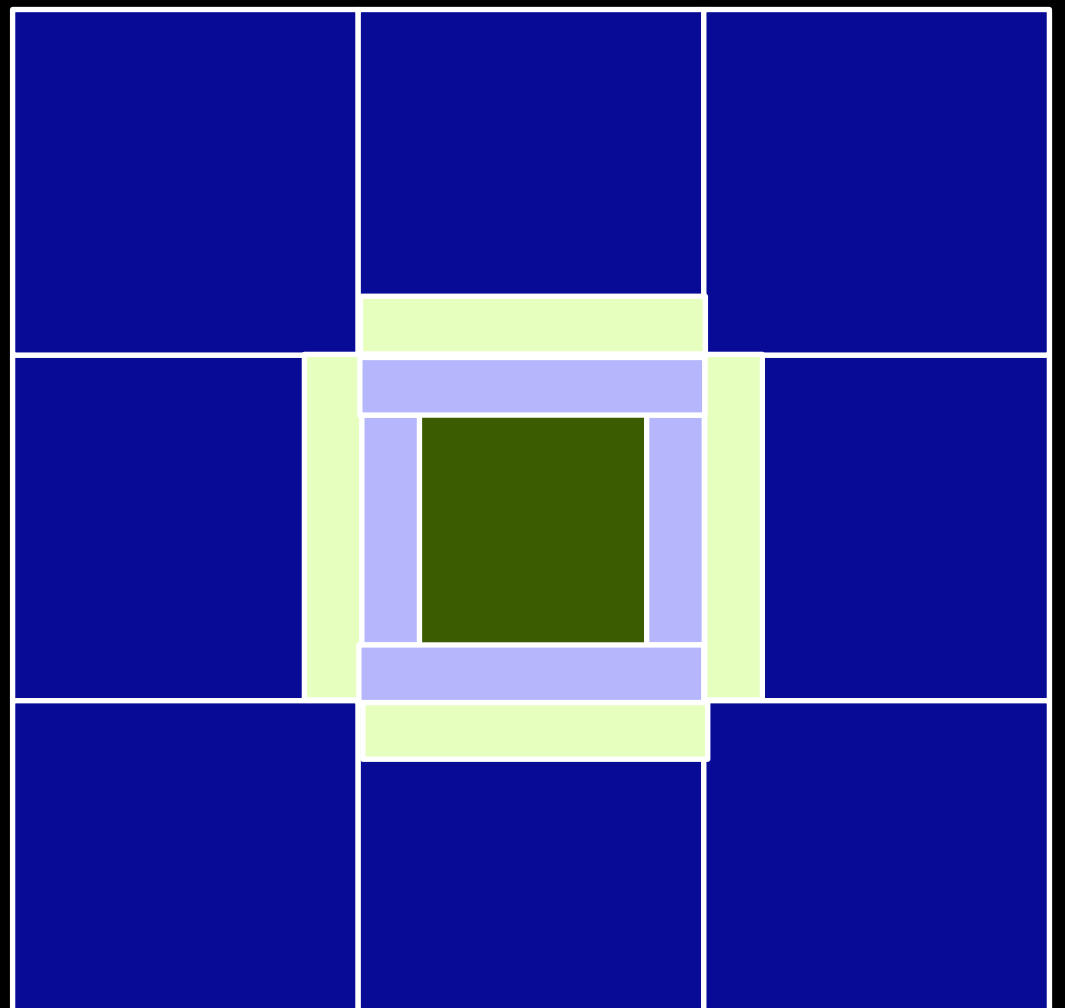
```
#pragma acc kernels pcopy(Anew[:][:]) pcopyin(A[:][:])
#pragma acc loop seq
for (int k = 3; k < NK-3; k++) {
  #pragma acc loop
  for (int j = 0; j < NJ; j++) {
    #pragma acc loop
    for (int i = 0; i < NI; i++) {
      Anew[k][j][i] = func(
          A[k-1][j][i], A[k-2][j][i], A[k-3][j][i],
          A[k+1][j][i], A[k+2][j][i], A[k+3][j][i], ...
      );
    }
  }
}
```

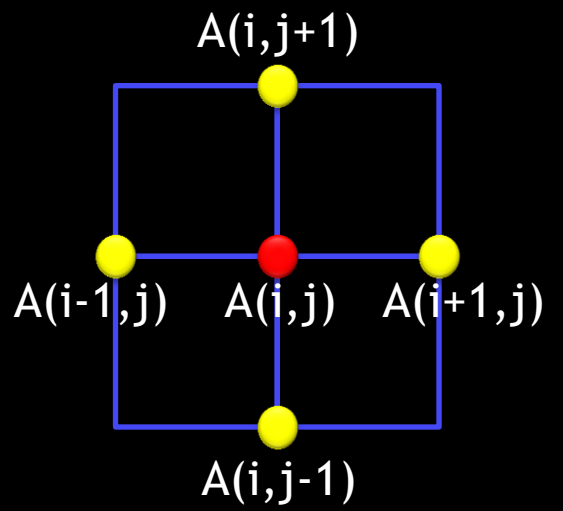
- Gang
- Worker
- Vector … SIMD幅

- Collapse
- Independent
- Seq
- Cache
- Tile
- …

MPIとは簡単に併用できるの?

MPI並列 (HALO EXCHANGE)





- ブロック分割
- 各プロセスは1ブロック担当
- 境界部(halo)のデータ交換

MPI JACOBI ITERATION

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew)
  update_A( A, Anew );
}
```

MPI JACOBI ITERATION

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {

  pack_data_at_boundary( send_buf, A, ... );

  exchange_data_by_MPI( recv_buf, send_buf, ... );


  unpack_data_to_halo( A, recv_buf, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew)
  update_A( A, Anew );
}
```

GPU
1.送信データの梱包

MPI
2.データの交換

GPU
3.受信データの開梱

MPI JACOBI ITERATION

```
#pragma acc data pcopy(A) create(Anew)
while ( error > tol ) {
  #pragma acc kernels pcopyin(A) pcopyout(send_buf)
  pack_data_at_boundary( send_buf, A, ... );

  exchange_data_by_MPI( recv_buf, send_buf, ... );

  #pragma acc kernels pcopy(A) pcopyin(recv_buf)
  unpack_data_to_halo( A, recv_buf, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew)
  update_A( A, Anew );
}
```

GPU

1. GPU上でデータを送信バッファに梱包し、Hostに転送

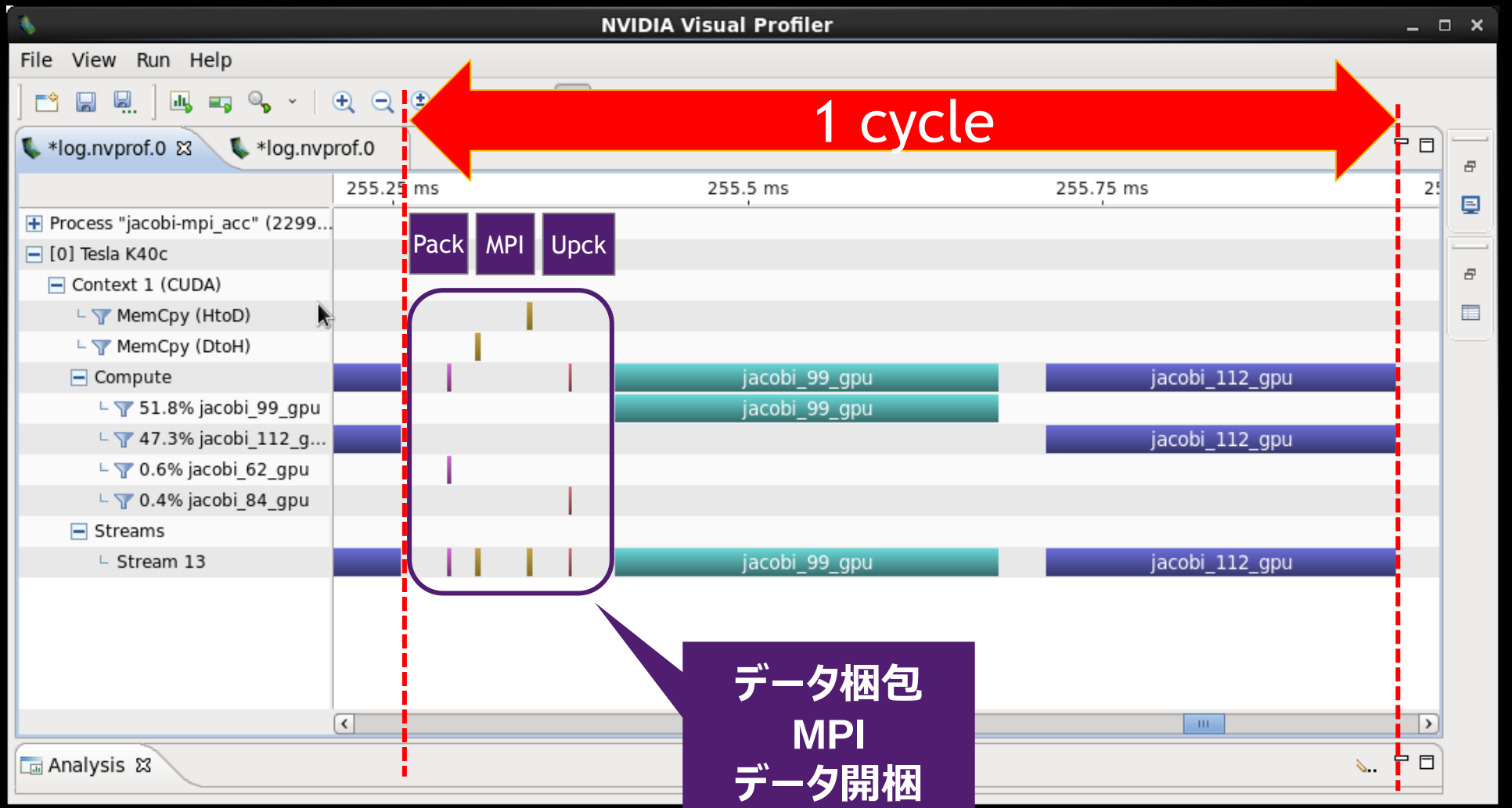
MPI

2. 隣接プロセスとデータ交換

GPU

3. GPUに転送、GPU上で受信バッファのデータを開梱

MPI JACOBI ITERATION (NVVP)



オーバーラップ (ASYNC/WAIT CLAUSE)

```
while ( error > tol ) {
  #pragma acc kernels pcopyin(A) pcopyout(send_buf)
  pack_data_at_boundary( send_buf, A, ... );

  exchange_data_by_MPI( recv_buf, send_buf, ... );

  #pragma acc kernels pcopy(A) pcopyin(recv_buf)
  unpack_data_to_halo( A, recv_buf, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew)
  update_A( A, Anew );
}
```

オーバーラップ (ASYNC/WAIT CLAUSE)

```
while ( error > tol ) {
  #pragma acc kernels pcopyin(A) pcopyout(send_buf)
  pack_data_at_boundary( send_buf, A, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A_inside( Anew, A, ... );

  exchange_data_by_MPI( recv_buf, send_buf, ... );

  #pragma acc kernels pcopy(A) pcopyin(recv_buf)
  unpack_data_to_halo( A, recv_buf, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A)
  calc_new_A_at_boundary( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew)
  update_A( A, Anew );
}
```

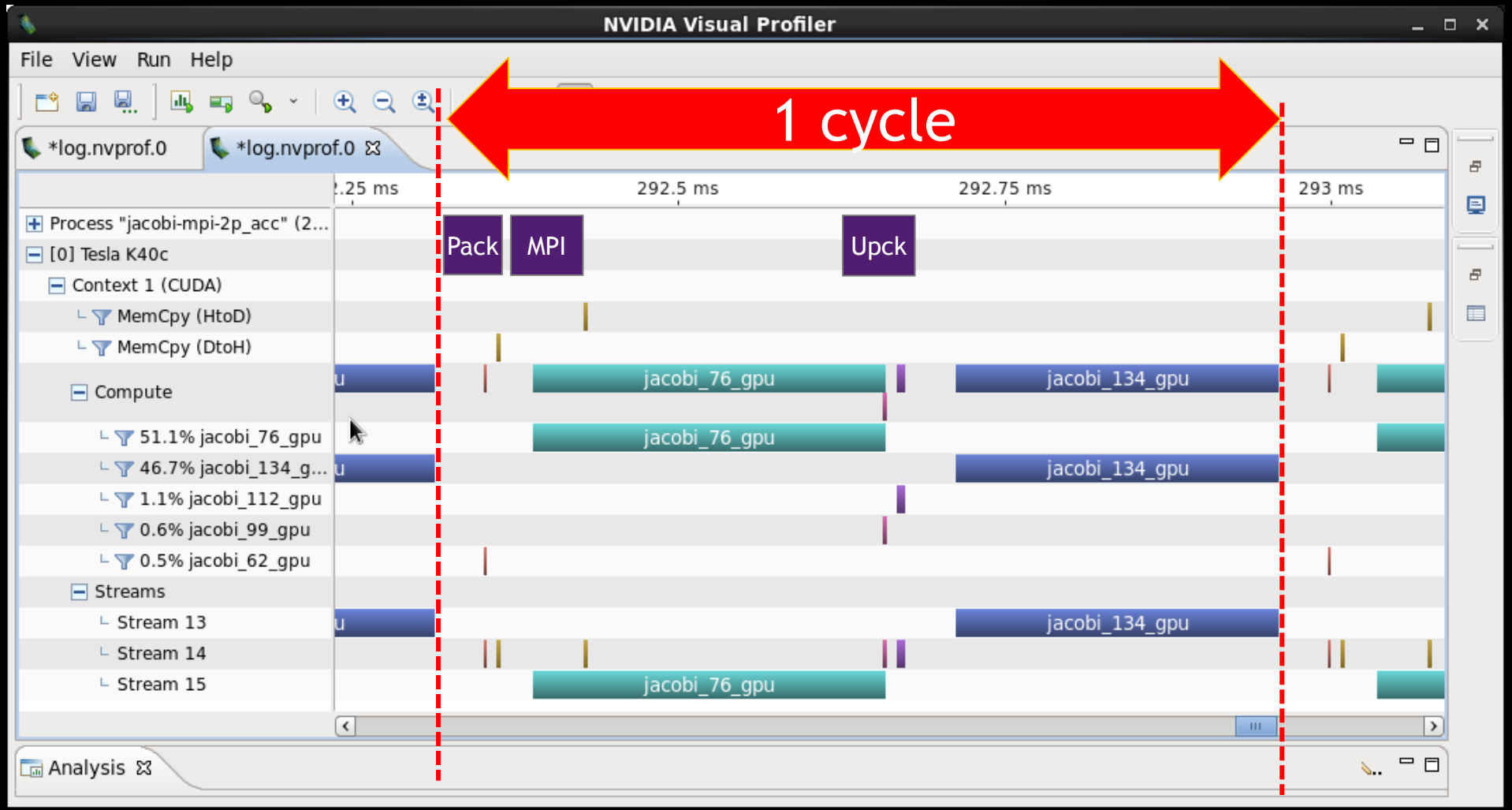
内部

境界部

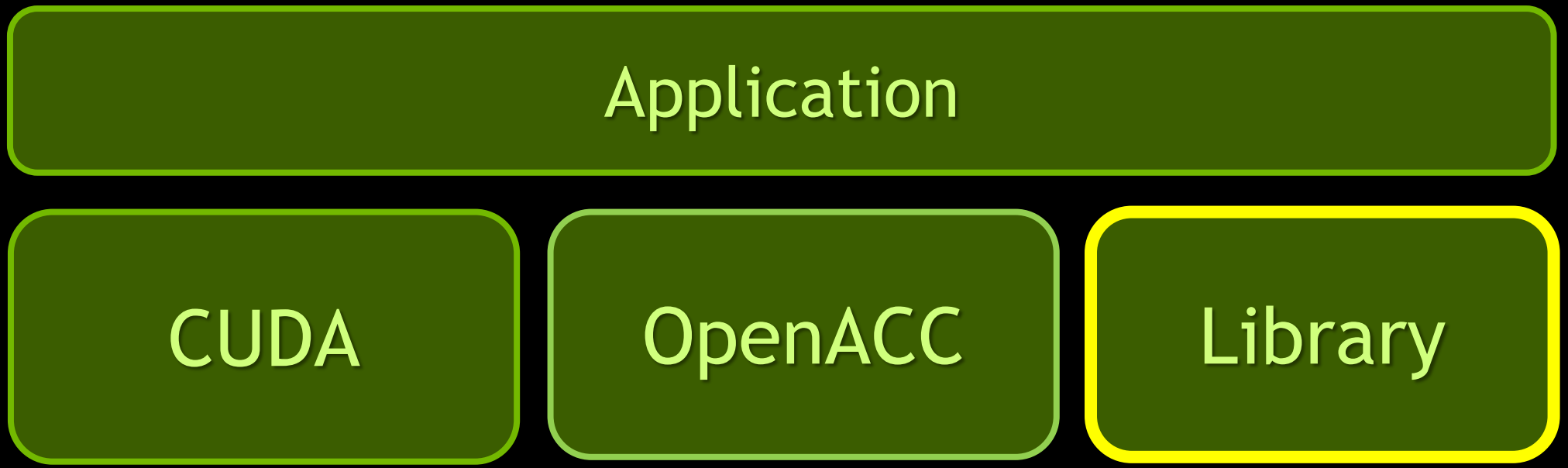
オーバーラップ (ASYNC/WAIT CLAUSE)

```
while ( error > tol ) {
  #pragma acc kernels pcopyin(A) pcopyout(send_buf) async(2)
  pack_data_at_boundary( send_buf, A, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A) async(1)
  calc_new_A_inside( Anew, A, ... );

  #pragma acc wait(2)
  exchange_data_by_MPI( recv_buf, send_buf, ... );

  #pragma acc kernels pcopy(A) pcopyin(recv_buf) async(2)
  unpack_data_to_halo( A, recv_buf, ... );

  #pragma acc kernels pcopy(Anew) pcopyin(A) async(2)
  calc_new_A_at_boundary( Anew, A, ... );

  #pragma acc kernels pcopy(A) pcopyin(Anew) wait(1,2)
  update_A( A, Anew );
}
```

オーバーラップ(NVVP)



アプリをGPU対応する方法



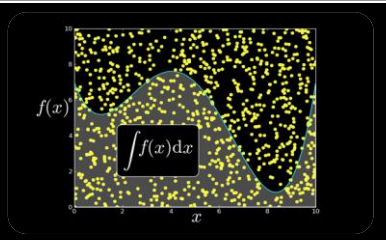
主要処理をCUDAで記述
高い自由度

既存コードにディレクティブを挿入
簡単に加速

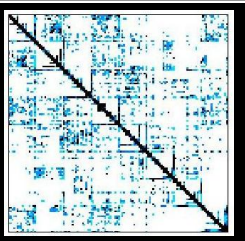
GPU対応ライブラリにチェンジ
簡単に開始

GPU対応のライブラリ (一部)

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

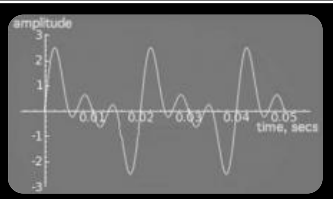
CULA | tools

GPU Accelerated
Linear Algebra

GPU VSIPL

Vector Signal
Image Processing

NVIDIA AmgX



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

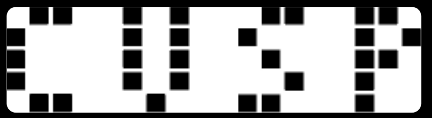
IMSL Library

MAGMA

ICL UT

Matrix Algebra on
GPU and Multicore

open source initiative



Sparse Linear
Algebra

open source initiative

Thrust

C++ STL Features
for CUDA

open source initiative

GPU TECHNOLOGY CONFERENCE

CUDA計算ライブラリ

高性能な計算ライブラリを提供

- cuFFT　Fast Fourier Transformsライブラリ
- cuBLAS　BLASライブラリ
- cuSPARSE　疎行列ライブラリ
- cuRAND　乱数生成ライブラリ
- NPP　画像処理 Performance Primitives
- Thrust　並列アルゴリズム C++ STL
- math.h　C99 浮動小数点ライブラリ

CUDAツールキットに標準搭載、フリー

developer.nvidia.com/cuda-downloads

まとめ

- GPUコンピューティングとCUDAの概要
- OpenACCの現状
  - 簡単: 既存コードへのディレクティブ追加
  - 強力: 少ない労力でGPU利用可能
  - オープン: 採用事例の増加