コード移動に基づくGPUカーネル融合

福原淳司　滝本宗宏
東京理科大学　理工学研究科　情報科学専攻

背景

- GPUプログラミングの普及
- GPUのオフチップメモリは低速

GPUプログラムを高速化する際のボトルネックとなる

カーネル融合(Kernel Fusion)

2つ以上のカーネルをまとめて、1つのカーネルにする手法

- カーネル起動オーバーヘッドの削減
- 他の最適化の効果をより大きくする

```
int main(){
  ...
  kernel1<<<grid, block>>>(d,a,b);
  kernel2<<<grid, block>>>(d2,a2,d);
  ...
}
```

```
int main(){
  ...
  fused<<<grid, block>>>(d,a,b,d2,a2);
  ...
}
```

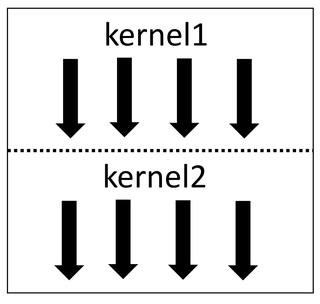
• 垂直融合(Vertical Fusion)

カーネルを縦に融合する手法。
各スレッドはkernel1を実行した後にkernel2を実行する。

```
void kernel1(...){
  ...
}

void kernel2(...){
  ...
}
```

```
void fused(...){
  kernel1();
  kernel2();
}
```

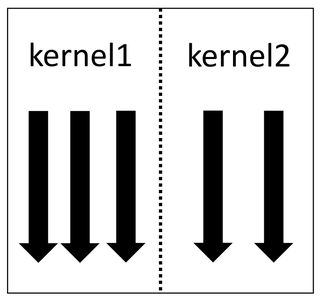


• 水平融合(Horizontal Fusion)

カーネルを横に融合する手法。
水平融合には以下の2種類がある。

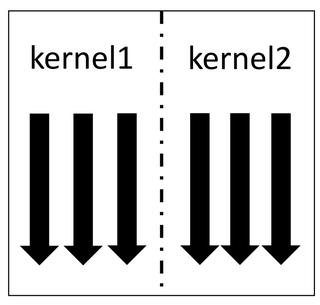
- Inner Block Fusion
  - 1つのブロック内で水平融合する
  - スレッドレベルの並列性が向上
  - 1スレッドが使える共有メモリが減る
  - ブロック内同期があると難しい

```
void fused(...){
  if(threadId < N){
    kernel1();
  } else {
    kernel2();
  }
}
```



- Inter Block Fusion
  - ブロックレベルで水平融合する
  - ブロックレベルの並列性が向上

```
void fused(...){
  if(blockId < N){
    kernel1();
  } else {
    kernel2();
  }
}
```



問題点と目的

既存手法はカーネル間のデータ依存を考慮して融合しており、**プログラムの制御構造は考慮できていない**。

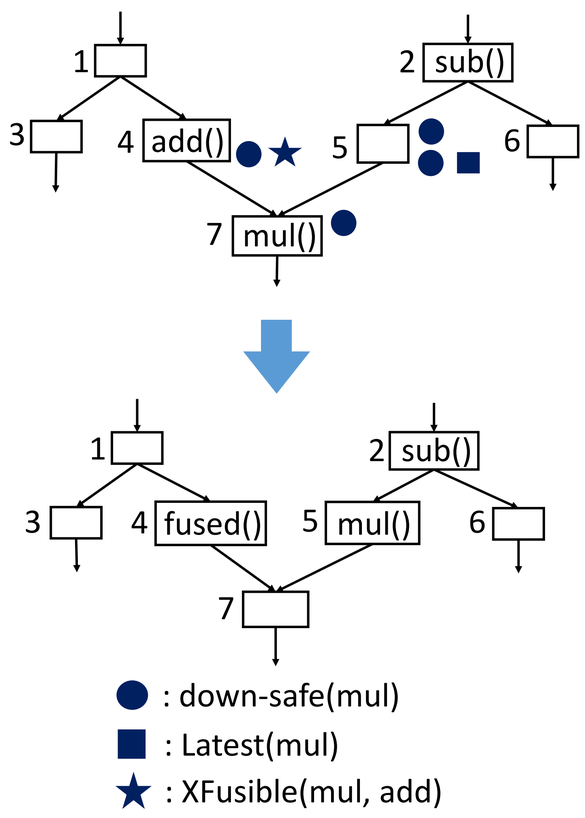
制御構造を考慮したカーネル融合を実現し、より多くのカーネルを融合する

```
int main(){
  if(cond){
    kernel1<<<grid, block>>>(...);
  }
  kernel2<<<grid, block>>>(...);
}
```

提案手法

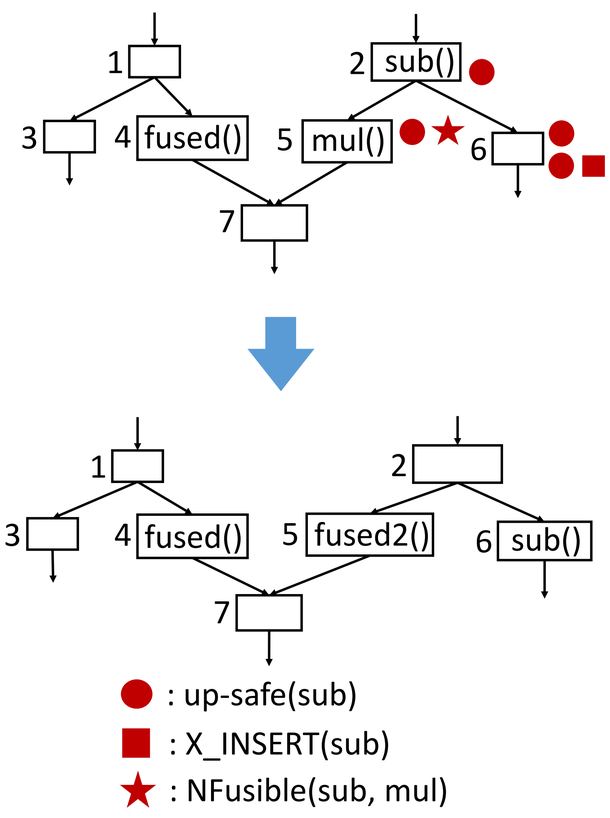
- 従来のカーネル融合と部分冗長除去法を組み合わせる
- 従来手法よりも多くのカーネルを融合できる
  - カーネル融合の効果をより大きくできる

1. Hoisting Phase



- カーネル呼出し文を巻き上げることで融合できるカーネルがないか探す
- 従来のPREで使われるデータフロー方程式を応用
- 節4でaddとmulが融合可能、節5にmulの呼び出し文の補償コードを挿入

2. Sinking Phase



- カーネル呼出し文を遅らせることで融合できるカーネルがないか探す
- 節2でsubとmulが融合可能、節6にsubの呼び出し文の補償コードを挿入
- 融合できなくなるまでフェーズ1と2を繰り返す

実験

本手法を簡単なプログラムに適用し、適用前後で実行速度を比較した。
約1.3倍の実行速度の改善が得られ、本手法の有効性を確認した。

```
__global__ void vector_add(float* d, float* a, float* b){
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  d[tid] = a[tid] + b[tid];
}
```

```
__global__ void vector_mad(float* d, float* a,
                           float* b, float* c){
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  d[tid] = a[tid] * b[tid] + c[tid];
}
```

実験環境
- OS: Ubuntu 18.04 LTS
- CPU: Intel Core i9-9900K
- GPU: NVIDIA TITAN RTX
- CUDA 11.1
- Clang/LLVM 11,1

ホスト側
```
add<<<grid, block>>>(d, a, b)
mad<<<grid, block>>>(d2, a2, b2, d)
```

カーネル融合

```
__global__ void fused_kernel(float* d, float* a, float* b,
                             float* c, float* e, float* f){
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  t = a[tid] + b[tid];      // 結果をレジスタに保持
  d[tid] = t;
  grid.sync();              // スレッドブロック間同期
  c[tid] = e[tid] * f[tid] + t;   // 保持してた結果の使用
}
```

```
fused_kernel<<<grid, block>>>
(d, a, b, d2, a2, b2)
```

○ Improvement
- グローバルメモリからのロード命令の削減
- カーネル起動命令の削減

× Degradation
- スレッドブロック間同期命令のオーバーヘッド

今後の展望

- 複雑なプログラムでは、ロード命令の削減と他の最適化の適用がより効果的になると考えられる。
- 本手法の効果をベンチマークプログラムで確認する。