HIGH PERFORMANCE PARALLEL PROGRAMMING (CS61064)

Soumyajit Dey
CSE, IIT Kharagpur

Loop Fusion

- Classical compiler optimization in programming
- Improve performance by reducing off-chip memory traffic
  - reduction of cache miss
  - better control of multiple instruction
  - reduces branching condition
- Operates by fusing iterations of different loops when those iterations reference the same data

Loop Fusion


```
//Before Fusion
for (i = 0; i < 300; i++)
    a[i] = a[i] + 3;
for (i = 0; i < 300; i++)
    b[i] = b[i] + 4;

//After Fusion
for (i = 0; i < 300; i++)
{
    a[i] = a[i] + 3;
    b[i] = b[i] + 4;
}
```

Kernel Fusion

- An optimization technique applied to a group of GPU kernels to increase efficiency by decreasing execution time, power consumption
- GPU kernels can not be scheduled once launched in device
- Kernel fusion can rearrange and schedule the kernels from the host side
- Kernels using the same or different data array(s) can be replaced with a single kernel call
- The new kernel aggregates the code segments of the separate kernels

Advantages

Increase efficiency by -

- data reuse using on-chip memory improves performance
- reducing off-chip memory data traffic
- reducing global memory data transfers
- reducing kernel launch overhead
- utilising maximum threads in GPU

Limitations

Kernel fusion does not always result in performance improvement:

- architectural resources like the capacity of on-chip memory and registers are limited
- reduction of off-chip memory data traffic is feasible upto a certain limit
- overhead in identifying fusable kernels
- overhead in defining a scalable method to search for the optimal rearrangement of fusable kernels
- may introduce divergence

Kernel Fusion example

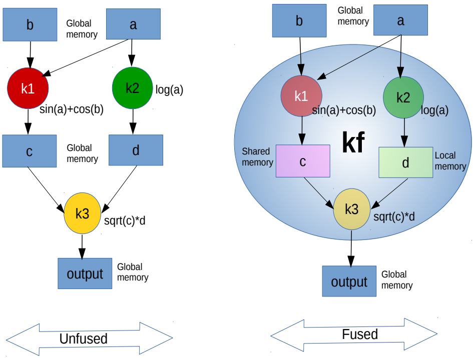


Figure: Kernel Fusion example

Code snippet for Kernel 1

```
__global__ void
process_kernel1(float *a, float *b, float
     *c, int datasize) {
int blockNum=blockIdx.z*(gridDim.x*
    gridDim.y)+blockIdx.y*gridDim.x+
    blockIdx.x;
int threadNum=threadIdx.z*(blockDim.x*
    blockDim.y)+threadIdx.y*blockDim.x+
    threadIdx.x;
int i=blockNum*(blockDim.x*blockDim.y*
    blockDim.z)+threadNum;
if (i<datasize)
     c[i]=sin(a[i])+cos(b[i]);
}
```

Code snippet for Kernel 2

```
__global__ void
process_kernel2(float *a, float *d, int
    datasize) {

    int blockNum=...
    int threadNum=...
    int i=...

    if (i<datasize)
        d[i]=log(a[i]);
}
```

Code snippet for Kernel 3

```
__global__ void
process_kernel3(float *c, float *d, float
    *output, int datasize){

    int blockNum=...
    int threadNum=...
    int i=...

    if (i<datasize)
        output[i]=sqrt(c[i])*d[i];
}
```

Code snippet for fused kernel

```
__global__ void
process_fused_kernel(float *a, float *b,
    float *output, int datasize) {

     __shared__ float c[blockDim.x *
        blockDim.y * blockDim.z];
    float d;
    int blockNum=...
    int threadNum=...
    int i=...
    if (i<datasize) {
        c[threadNum]=sin(a[i])+cos(b[i]);
        d=log(a[i];
        output[i]=sqrt(c[threadNum])*d;
    }
}
```

Types

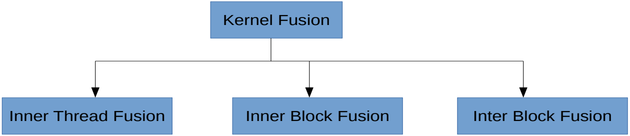


Figure: Kernel Fusion types

Inner Thread Fusion

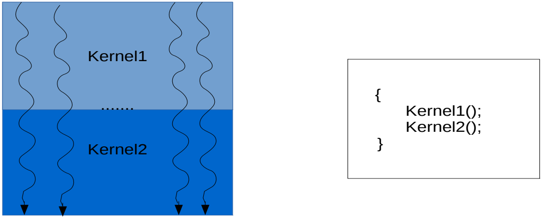


Figure: Inner Thread Fusion

Inner Thread Fusion

- Combines computation of two kernel into single thread
- Suitable for both dependent and independent kernels if dataspace size is same
- Let, $S_{th,i}$ represents the size of threads in a thread block $S_{bk,i}$ represent the size of blocks in kernel $i(i = 1, 2)$
- For fused kernel -
  - $S_{th} = \max(S_{th,1}, S_{th,2})$
  - $S_{bk} = \max(S_{bk,1}, S_{bk,2})$
- Not suitable if -
  - Kernels not having same thread/block space
  - Results in unbalanced workloads between threads

Inner Thread Fusion Example

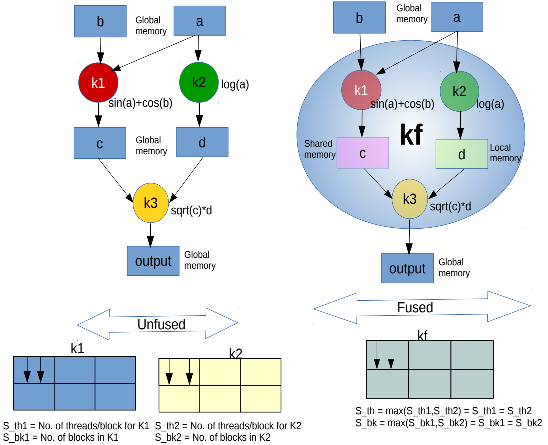


Figure: Inner Thread Fusion of dependent kernels with same dataspace size and thread/block size

Inner Thread Fusion Example

Fusion of dependent kernels with same dataspace size and thread/block size

```
//Unfused kernels
k1(a, b, c, n):
    i = global threadId
    c[i]=sin(a[i])+cos(b[i])
k2(a, d, n) :
    i = global threadId
    d[i]=log(a[i])
//Fused kernel
kf(a, b, out , n):
    local c,d
    i = global threadId
    if(i<n)
        c=sin(a[i])+cos(b[i]);
        d=log(a[i];
        output[i]=sqrt(c)*d);
```

Inner Thread Fusion Example

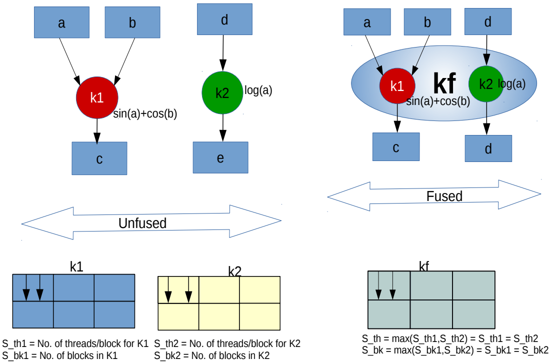


Figure: Inner Thread Fusion of independent kernels with same dataspace size and thread/block size

Inner Thread Fusion Example

Fusion of independent kernels with same dataspace size and thread/block size

```
//Unfused kernels
k1(a, b, c, n):
    i = global threadId
    c[i]=sin(a[i])+cos(b[i])
k2(d, e, n) :
    i = global threadId
    e[i]=log(d[i])

//Fused kernel
kf(a, b, c, d, e, n):
i = global threadId
if(i<n)
    c[i]=sin(a[i])+cos(b[i])
    e[i]=log(d[i])
```

Inner Thread Fusion Example

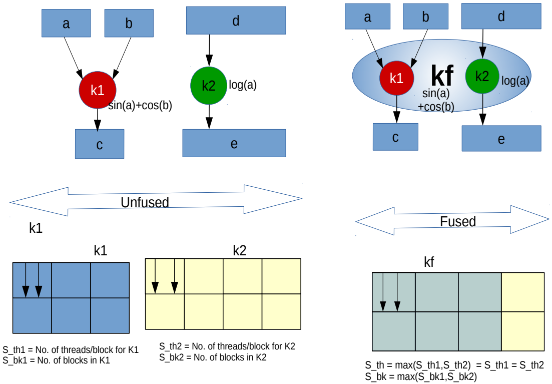


Figure: Inner Thread Fusion of independent kernels with different data size but same thread/block size

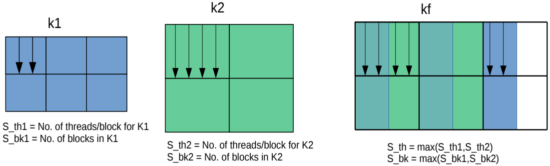
Inner Thread Fusion

Fusion of independent kernels with different data size but same thread/block size:

```
//Fused kernel
//Let n2>n1
kf(a, b, c, d, e, n1, n2):
i = global threadId
if(i<n1)
    c[i]=sin(a[i])+cos(b[i])
    e[i]=log(d[i])
else if(i<n2)
    e[i]=log(d[i])
```

Inner Thread Fusion Limitation

Fusion of independent kernels with different data size and thread/block size:
NOT SUITABLE: Due to unbalanced workloads between threads



Inner Block Fusion

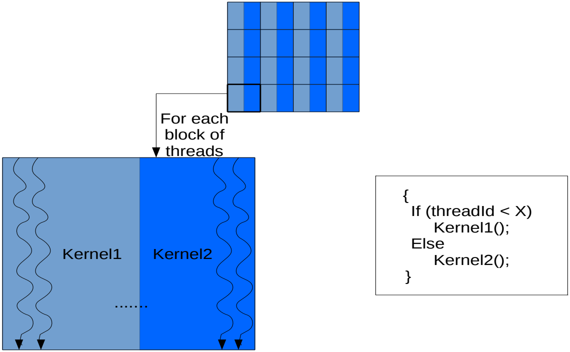


Figure: Inner Thread block Fusion

Inner Block Fusion

- Distribute computation of two different kernel among threads in single block
- For independent kernels with small block size(threads/block)
- Let, $S_{th,i}$ represents the size of threads in a thread block $S_{bk,i}$ represent the size of blocks in kernel $i(i = 1, 2)$
- For fused kernel -
  - $S_{th} = \text{sum}(S_{th,1}, S_{th,2})$
  - $S_{bk} = \max(S_{bk,1}, S_{bk,2})$
- Not suitable if -
  - S_{th} of fused kernel exceed upper bound of threads/block size
  - kernels with synchronization statement

Inner Block Fusion Example

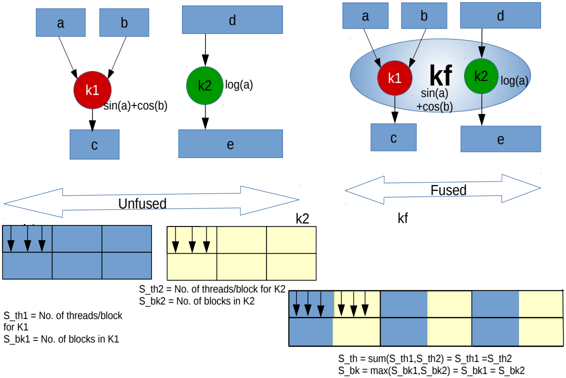


Figure: Inner Block Fusion of independent kernels with same dataspace size

Inner Block Fusion Example

Fusion of independent kernels with same dataspace size

```
//Unfused kernels
k1(a, b, c, n):
    i = global threadId
    c[i]=sin(a[i])+cos(b[i])

k2(d, e, n) :
    i = global threadId
    e[i]=log(d[i])

//Fused kernel
//S_th1 = No. of threads/block for K1
//S_th2 = No. of threads/block for K2
//S_th = sum(S_th1,S_th2)
```

Inner Block Fusion Example

```
kf(a, b, c, d, e, n):
    b = blockId
    t = threadId

    if(t<S_th1)
        c[t]=sin(a[t])+cos(b[t])
    else if(t<S_th)
        e[t-S_th1]=log(d[t-S_th1])
```

Inner Block Fusion Example

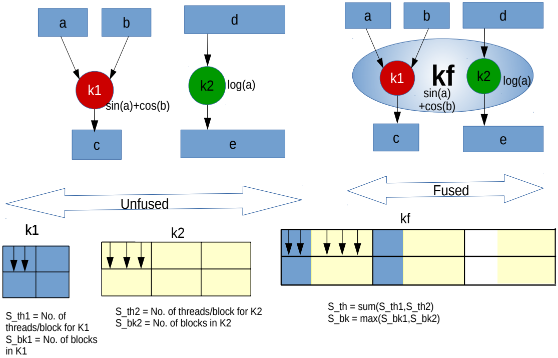


Figure: Inner Block Fusion of independent kernels with different dataspace size

Inner Block Fusion Example

Fusion of independent kernels with different dataspace size

```
//Unfused kernels
k1(a, b, c, n1):
    i = global threadId
    c[i]=sin(a[i])+cos(b[i])

k2(d, e, n2) :
    i = global threadId
    e[i]=log(d[i])

//Fused kernel
//S_th = sum(S_th1,S_th2)
//S_bk = max(S_bk1,S_bk2)
//Let S_th2 > S_th1
//Let S_bk2 > S_bk1
```

Inner Block Fusion Example


```
kf(a, b, c, d, e, n1, n2, X):
    b = blockId
    t = threadId
    if(b<S_bk)
        if(t<S_th1 AND b<S_bk1)
            c[t]=sin(a[t])+cos(b[t])
        else if(t<S_th)
            e[t-S_th1]=log(d[t-S_th1])
```

Inner Block Fusion Limitation

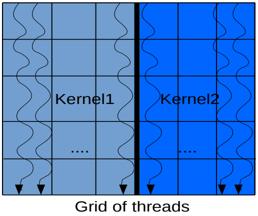
Upper bounds for threads/block is architecture dependent

- S_{th} of fused kernel must not exceed this upper bound

CUDA does not support synchronization for partial threads in a block

- kernels with sync() statement like reduction kernel not applicable for this type of fushion

Inter block Fusion




```
{
  If (blockId < X)
      Kernel1();
  Else
      Kernel2();
}
```

Figure: Inter Thread block Fusion

Inter block Fusion

- Distribute computation of two different kernel among different blocks
- For independent kernels with similar computation time
- Let, $S_{th,i}$ represents the size of threads in a thread block $S_{bk,i}$ represent the size of blocks in kernel $i (i = 1, 2)$
- For fused kernel -
  - $S_{th} = max(S_{th,1}, S_{th,2})$
  - $S_{bk} = sum(S_{bk,1}, S_{bk,2})$
- Not suitable if -
  - Workload of different thread block differs a lot

Inter Block Fusion Example

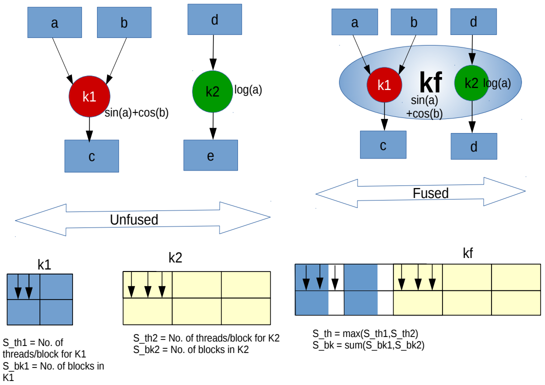


Figure: Inter Block Fusion of independent kernels with different dataspace size

Inter Block Fusion Example

Fusion of independent kernels with different dataspace size

```
//Unfused kernels
k1(a, b, c, n1):
    i = global threadId
    c[i]=sin(a[i])+cos(b[i])
k2(d, e, n2) :
    i = global threadId
    e[i]=log(d[i])

//Fused kernel
//S_th = max(S_th1,S_th2)
//S_bk = sum(S_bk1,S_bk2)
//Let S_th2 > S_th1
//Let S_bk2 > S_bk1
```

Inter Block Fusion Example

```
kf(a, b, c, d, e, n1, n2):
b = blockId
t = threadId
i = global threadId

if(b<S_bk1)
    if(t<S_th1)
        c[t]=sin(a[t])+cos(b[t])
else if(b<S_bk)
    if(t<S_th2)
        e[t]=log(d[t])
```

Inter Block Fusion Limitation

Workload of different thread block differs a lot. For example below two kernels are not suitable for this type of fusion.

```
k1(a, b, c, n1):
    i = global threadId
    c[i]=sqrt(sin(a[i])+cos(b[i]))
k2(d, e, f, n2) :
    i = global threadId
    f[i]=d[i]+e[i]
```