# Automated GPU Kernel Transformation as an Optimization Problem

Kristian Kadlubiak

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

ikadlubiak@fit.vutbr.cz

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

# An Introduction to the GPGPU


SM
Instruction Cache
Instruction Buffer
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Instruction Buffer
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
DP Unit
LD/ST
SFU
Texture / L1 Cache
Tex
Tex
Tex
Tex
64KB Shared Memory

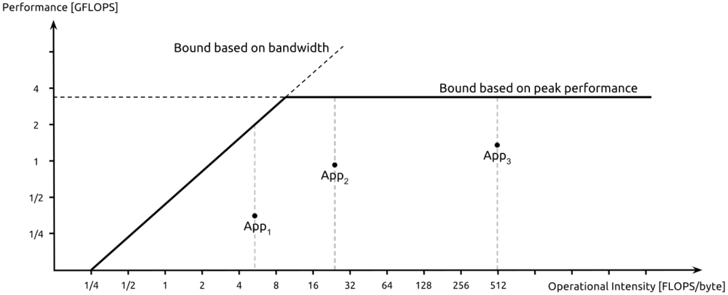
# The Performance of Different Memory Types

| Memory type | Latency [clocks] | Visibility | Amount |
|---|---|---|---|
| Register | aprox. 0 | thread | 128 B |
| Shared Memory | aprox. 50 | block | 64 KB (32 B) |
| Global Memory | aprox. 200 | global | 8 GB |

# Different Memory Types Usage Example

```
__global__ void MatrixAdd(A, B, C, stride)
{
  __shared__ sB[blockDim.y][blockDim.x];

  int globalIdX = getGlobalIdX();
  int globalIdY = getGlobalIdY();
  int localIdX  = getLocalIdX();
  int localIdY  = getLocalIdY();

  float rA;
  float rC;

  rA = A[globalIdY * stride + globalIdX];

  sB[localIdY][localIdX] = B[globalIdY * stride + globalIdX];

  rC = rA + sB[localIdY][localIdX];

  C[globalIdY * stride + globalIdX] = rC;

}
```

- Modeling theoretical peak performance in relation with the operational intensity
- Helpful in determination of a bottleneck



# The Roofline Model: Example

```
__global__ void vectorAdd(A, B, C)
{
  int globalIdX = getGlobalIdX();

  C[globalIdX] = A[globalIdX] + B[globalIdX];
}
```

- 1 FLOP per 8 bytes loaded from the global memory
- A memory bound problem
- 1/8 operational intensity with 320 GB/s memory throughput leads to 40 GFLOPS instead of 8228 GFLOPS (Nvidia GTX 1080)
- Considering addition of two vectors, each of size 1 GB, the computation would take 25 ms (40 GFLOPS) compared to 0.12 ms (8228 GFLOPS)

# The Kernel Fusion

```
__global__ void twoVectorAdd(A, B, C, D, E)
{
  int globalIdX = getGlobalIdX();

  float rA = A[globalIdX];

  D[globalIdX] = rA + B[globalIdX];
  E[globalIdX] = rA + C[globalIdX];
}
```

- 2 FLOP per 12 bytes loaded from global memory
- Still a memory bound problem
- 1/6 operational intensity with 320 GB/s memory throughput leads to 53 GFLOPS
- Considering addition of three vectors each of size 1 GB. The computation would take 37.5 ms. However two consecutive calls to the `vectorAdd()` would take 50 ms.
- By fusing two kernels into one we are able to cut the runtime by 25 %

# Fusion Candidates

- Two constructions are suitable for the fusion

```
kernel1<<<grid,block>>>(inA, outB);
kernel2<<<grid,block>>>(inA, outC);
```

- Aforementioned example

```
kernel1<<<grid,block>>>(inA, outB);
kernel2<<<grid,block>>>(inB, outC);
kernel3<<<grid,block>>>(inC, outD);
```

- Kernels creating ”chain” or ”pipeline”
- Data dependencies implies the order of execution

# The Kernel Fusion

# Demonstration Example

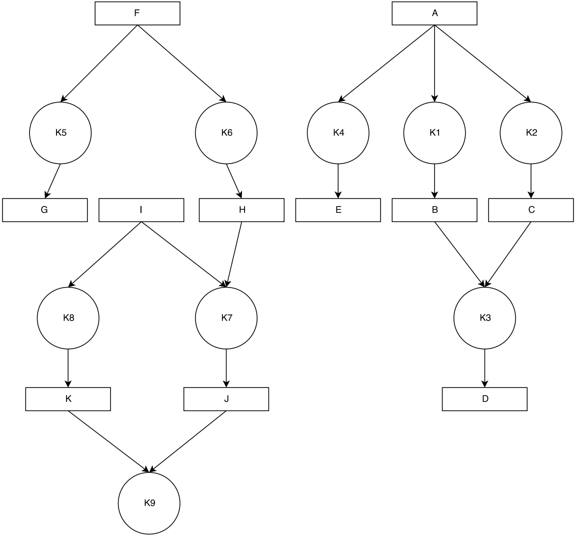
```
int main()
{
  //preprocessing

  kernel1<<<grid,block>>>(inA, outB);
  kernel2<<<grid,block>>>(inA, outC);
  kernel3<<<grid,block>>>(inB, inC, outD);
  kernel4<<<grid,block>>>(inA, outE);
  kernel5<<<grid,block>>>(inF, outG);
  kernel6<<<grid,block>>>(inF, outH);
  kernel7<<<grid,block>>>(inH, inI, outJ);
  kernel8<<<grid,block>>>(inI, outK);
  kernel9<<<grid,block>>>(inK, inJ, outL);

  //postprocessing
}
```

**The data dependency graph**

It is a DAG $G_{ddg}(V,E)$ where $K \subseteq V$ and $D \subseteq V$ represents kernels and data arrays respectively. $E$ is a set of edges composed of two types of edges:

- $(x,y) \in E; x \in D, y \in K$ and $x$ is input array of the kernel $y$
- $(y,x) \in E; x \in D, y \in K$ and $x$ is output array of the kernel $y$

# The Data Dependency Graph of Example

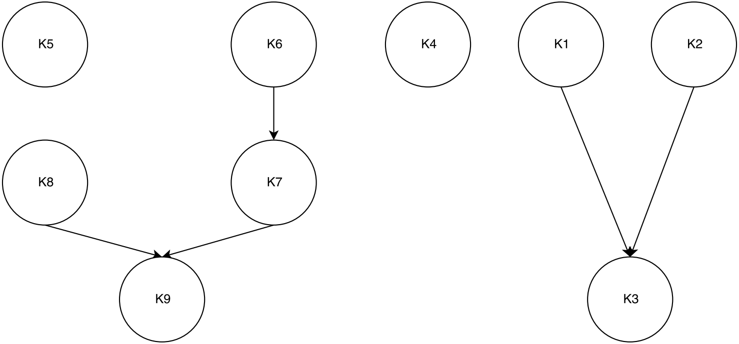


**The Order-of-execution graph**

It is a DAG $G_{ooe}(K,O)$ where $K$ represents kernels and $O$ is a set of edges defined as follows:

- $\forall x,z \in K, y \in D; (x,y) \in E \vee (y,z) \in E \iff (x,z) \in O$

# The Order-of-Execution Graph of Example



**General definition of an combinatorial optimization problem**

The goal is to find $y \in f(x)$, such that

$$m(x,y) = g\{m(x,y')|y' \in f(x)\}$$

where $x \in I$ and $I$ is a set of instances, $f(x)$ is a set of feasible solutions. Function $m$ is a measure of $y$ which for every tuple $(x,y); x \in I, y \in f(x)$ returns positive integer and $g$ is goal function, which is either $max$ or $min$.

**The definition of combinatorial optimization problem in context of kernel fusion**

Consider $K$ a set of $n$ kernels.
The goal is to find $K_1, K_2, \ldots, K_m \subseteq K$

- $K_i \cap K_j = \emptyset; i \neq j; i,j \in \{0, 1, \ldots, m\}$
- $\bigcup_{i=0}^{m} K_i = K$

such that $\sum_{i=0}^{m} T_p(K_i)$ where $T_p : \mathcal{P}(K) \to \mathbb{R}$ is minimized.

## The definition of an optimization problem with constrains

Consider $K$ set of original kernels $|K| = n$ and $F$ set of new kernels $|F| = m$

The goal is to minimize $\sum_{j=1}^{m} T_p(F_j)$ which is subject to:

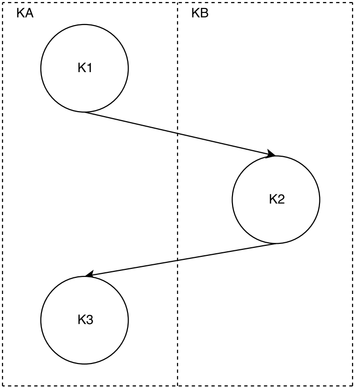
- $\sum_{i \in F_k} T_m(K_i) > T_p(F_k), \forall F_k \in F$
- $x_{ij} \in \{0,1\}, \forall i \in \{1,\ldots,n\} \forall j \in \{1,\ldots,m\}$
- $\sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1,\ldots,n\}$
- $x_{qr} = 1, \forall q \in K_{a \to b}, x_{ar} = 1, x_{br} = 1$
- $\forall F_x \in F, \forall K_i \in F_x, \exists K_j \in F_x, DegKin(K_i, K_j) > 0$
- $SHMEM(F_j) \leq SHMEM_{max}, \forall j \in \{1,\ldots,m\}$
- $REG(F_j) \leq REG_{max}, \forall j \in \{1,\ldots,m\}$

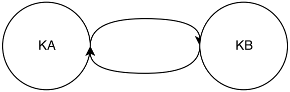
## Explanation

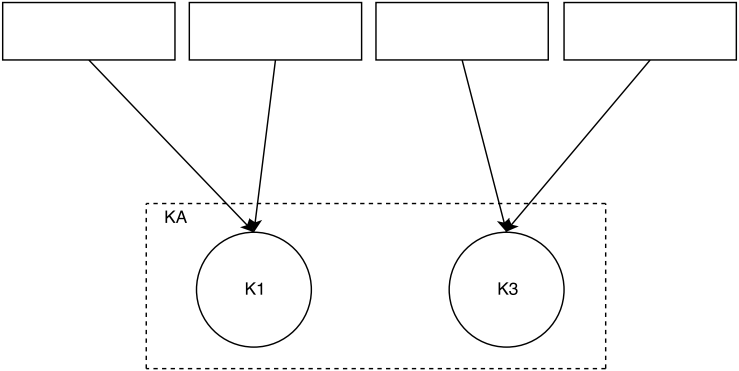
Where:

- $T_m(K_i)$ is measured execution time of the kernel $K_i \in K$
- $T_p(F_j)$ is execution time projection of new fused kernel $F_j \in F$
- $x_{ij} = 1$ when $K_i \in K$ is fused into $F_j$
- $K_{a \to b}$ is set of all kernels in path in $G_{ooe}$ from kernel $K_a$ to $K_b$
- $DegKin(K_i, K_j)$ is number of common immediate ancestors in $G_{ddg}$ for $K_i$ and $K_j$
- $DegKin(K_i, K_j)$ is $n - 1$, when there is path in $G_{ooe}$ consisting of $n$ nodes between $K_i$ and $K_j$
- $DegKin(K_i, K_j)$ is $0$ otherwise
- $SHMEM(F_j)$ is amount of shared memory required by new fused kernel $F_j$
- $REG(F_j)$ is number of registers required per thread by new fused kernel $F_j$

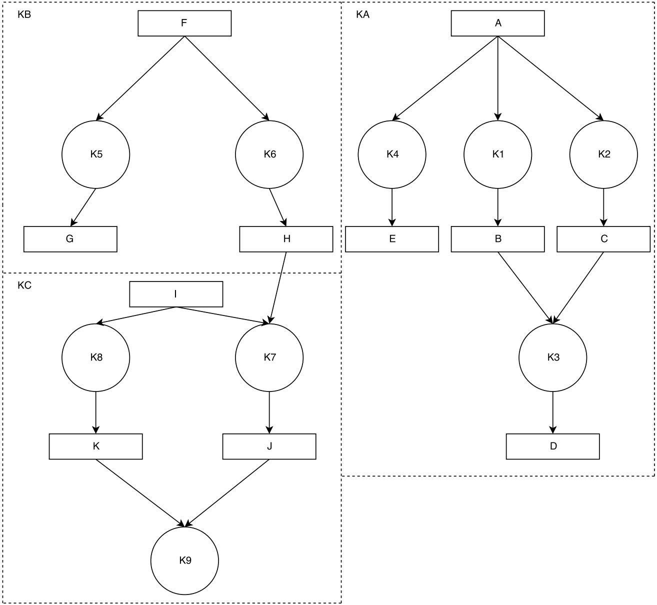
# The Line Requirement Explained




KA
K1
K3

# Feasible Solution



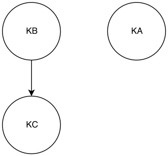
FIT

### The Order-of-execution graph of solution

It is a DAG $G_{ooefs}(F, O_s)$ where $F$ represents new fused kernels and $O_s$ is a set of edges defined as follows:

- $\forall F_x, F_y \in F; F_x \neq F_y; \exists K_i \in F_x \exists K_j \in F_y; (K_i, K_j) \in O \iff (F_x, F_y) \in O_s$

# The Order-of-Execution Graph of Solution: Example




```
int main()
{
  //preprocessing

  kernelA<<<grid,block>>>(inA, outD);
  kernelB<<<grid,block>>>(inF, outG, outH);
  kernelC<<<grid,block>>>(inH, inI, outL);

  //postprocessing
}
```

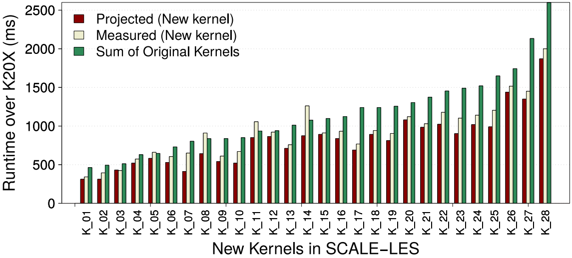
# Application

- It is possible to automate almost entire process
- Output of such process is a template of new kernels
- Generation of graphs is straightforward
- A modified version of GA can be used to solve combinatorial optimization problem

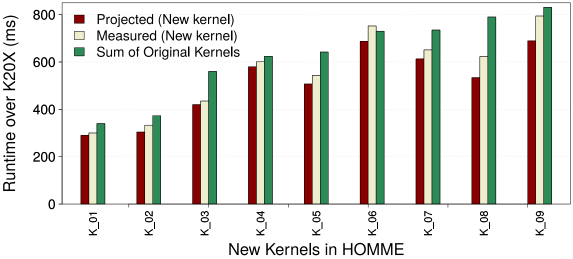
## The Kernel fusion algorithm

1. Gather metadata of original kernels $K_i, i = 1, \ldots, n$
2. Create the dependency graph
3. Create the order-of-execution graph
4. $G_0 \leftarrow$ generate M feasible solutions as an initial population
5. For all $M_i \in G_i$
   - Estimate runtime of $M_i$
6. $G_t^{Se} \leftarrow$ select $N \leq M$ individuals from $G_{t-1}$ according to selection method
7. $G_t^{Se} \leftarrow$ apply crossover and mutation
8. $G_t \leftarrow$ replace $N$ individuals with $G_t^{Se}$ according to selection policy
9. If termination criteria are not met go to step 5
10. Use values of the best solution as an template

# Results

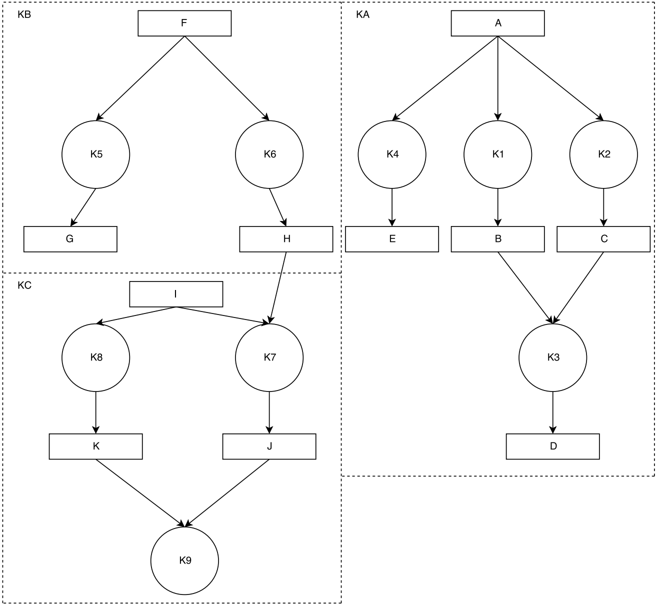


Projected (New kernel)
Measured (New kernel)
Sum of Original Kernels
Runtime over K20X (ms)
2500
2000
1500
1000
500
0
K_01 K_02 K_03 K_04 K_05 K_06 K_07 K_08 K_09 K_10 K_11 K_12 K_13 K_14 K_15 K_16 K_17 K_18 K_19 K_20 K_21 K_22 K_23 K_24 K_25 K_26 K_27 K_28
New Kernels in SCALE−LES

Projected (New kernel)
Measured (New kernel)
Sum of Original Kernels
Runtime over K20X (ms)
800
600
400
200
0
K_01 K_02 K_03 K_04 K_05 K_06 K_07 K_08 K_09
New Kernels in HOMME

# Feasible Solution



- Y. Lin G. Wang and W. Yi. “Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU”. In: Physical and Social Computing (CPSCom) 11 (2010), pp. 344–350.
- M. Wahib and N. Maruyama. “Scalable Kernel Fusion for Memory-Bound GPU Applications”. In: SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. 2014, pp. 191–202.
- Wikipedia: Optimization Problem. https://en.wikipedia.org/wiki/Optimization_problem. Accessed: 2017-11-30.