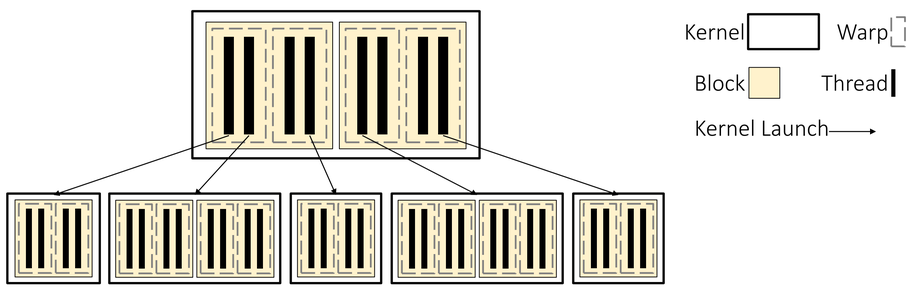
KLAP: Kernel Launch Aggregation and Promotion for Optimizing Dynamic Parallelism

Izzat El Hajj[1] (Illinois), Juan Gómez-Luna[2] (Córdoba), Cheng Li (Illinois), Li-Wen Chang (Illinois), Dejan Milojicic (HPE), Wen-mei Hwu (Illinois)
[1]elhajj2@Illinois.edu, [2]el1goluj@uco.es

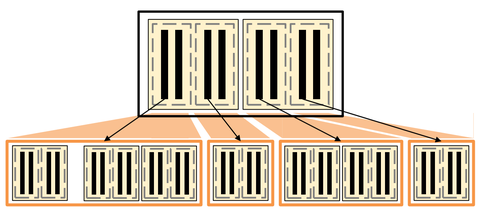
Kernel Launch Aggregation

Problem: Launching too many fine-grain kernels

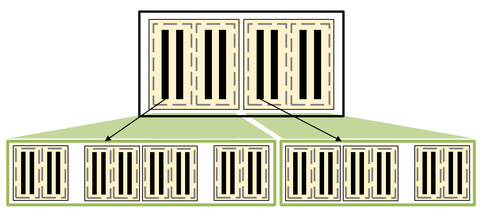
- Large kernel count incurs too much **launch overhead**
- Fine granularity of kernels **underutilizes GPU resources**



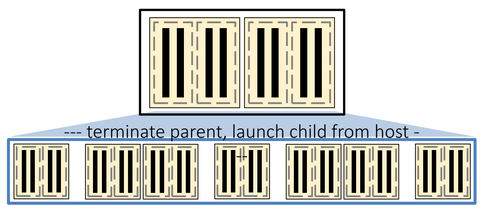
Solution: One thread launches a kernel on behalf of a group



Warp-Granularity Kernel Launch Aggregation



Block-Granularity Kernel Launch Aggregation



Kernel-Granularity Kernel Launch Aggregation

Code Transformation:

```
kernel<<<gD,bD>>>(arg1,arg2,arg3)
```

Original Kernel Call

```
__global__ void kernel(params) {
    kernel body
}
```

Original Kernel

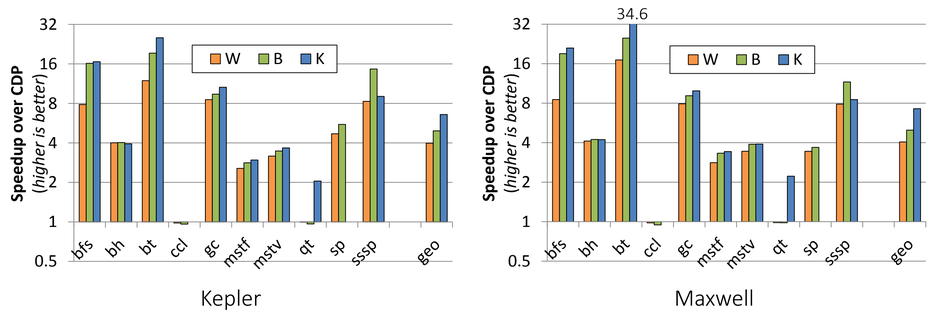
```
allocate arrays for args, gD, and bD
store args in arg arrays
store gD in gD array, and bD in bD array
new gD = sum of gD array across warp/block
new bD = max of bD array across warp/block
if(threadIdx == launcher thread in warp/block) {
    kernel_agg<<<new gD, new bD>>>
            (arg arrays, gD array, bD array)
}
```

Transformed Kernel Call
(block-granularity aggregation example)

```
__global__ void kernel_agg(param arrays, gD array, bD array) {
    calculate index of parent thread
    load params from param arrays
    load actual gridDim/blockDim from gD/bD arrays
    calculate actual blockIdx
    if(threadIdx < actual blockDim) {
        kernel body   (with kernel launches transformed and with
                       using actual gridDim/blockDim/blockIdx)
    }
}
```

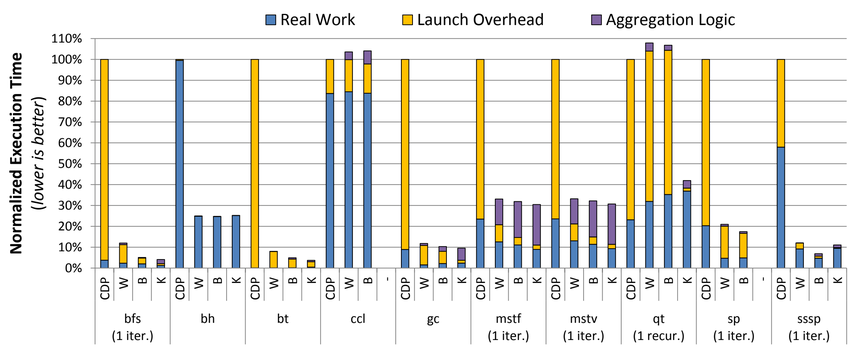
Transformed Kernel
(block-granularity aggregation example)

Results:



Kepler Maxwell

Increasing aggregation granularity improves performance
(geomean speedup of **6.58x** for K-aggregation on Kepler)

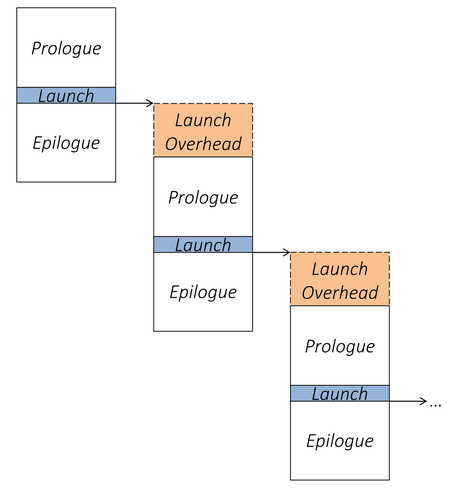


Performance improvement comes from **reduced launch overhead** and **better resource utilization**

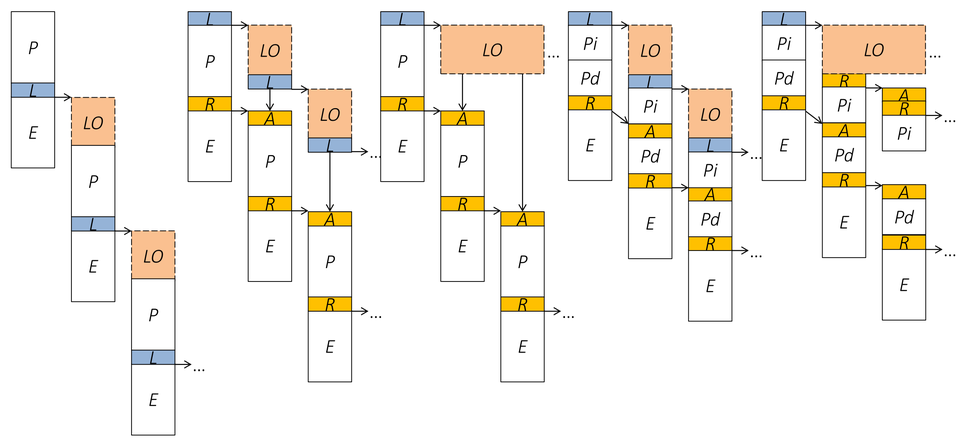
Kernel Launch Promotion

Problem: Deep call stacks

- Launch overhead **dominates the critical path**
- Call stack has **limited depth**



Solution: One thread launches a kernel on behalf of a group



Original Kernel | Promotion (P) | Promotion with Aggregation (PA) | Promotion with Overlap (PO) | Promotion with Aggregation and Overlap (PAO)

Code Transformation:

```
__global__ void kernel(params_avail, params_post) {
    prologue
    if(launcher thread) {
        kernel<<<1,nThreads>>>(args_avail, args_post)
    }
    epilogue
}
```

Original Kernel

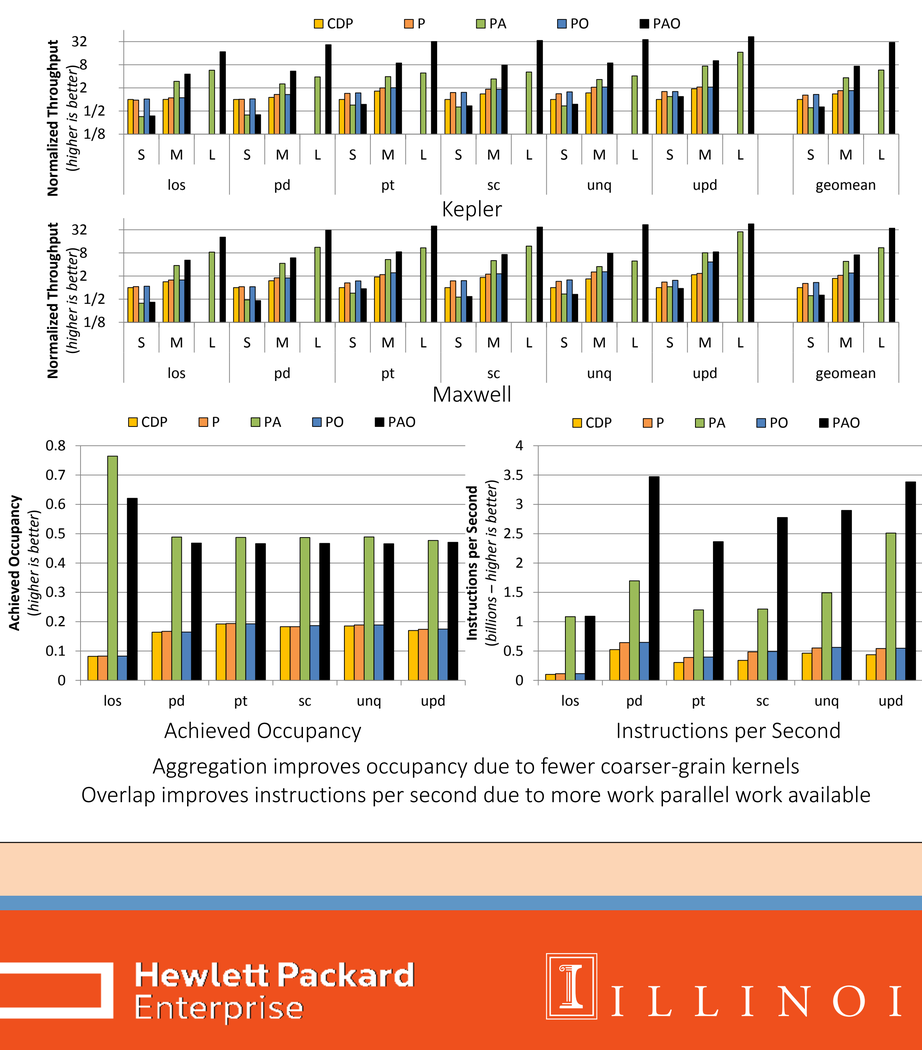
```
__global__ void kernel(params_avail, params_post) {
    if(launcher thread) {
        allocate postponed arg buffers
        allocate child flag
        kernel_from_kernel<<<1,nThreads>>>
            (args_avail, postponed arg buffers, child flag)
    }
    prologue
    if(launcher thread) {
        store args_post in postponed arg buffers
        memory fence
        set child flag to release child
    }
    epilogue
}
```

Transformed Kernel (called from host)

```
args_avail  : arguments available at the beginning of the kernel
args_post   : arguments whose values are "postponed" because they
              are not available at the beginning of the kernel
params_avail : parameters corresponding to available arguments
params_post  : parameters corresponding to postponed arguments
__global__ void kernel_from_kernel(params_avail,
                                   postponed param buffers, flag) {
    if(launcher thread) {
        allocate postponed arg buffers
        allocate child flag
        kernel_from_kernel<<<1,nThreads>>>
            (args_avail, postponed arg buffers, child flag)
    }
    wait to acquire flag
    load params_post from postponed param buffers
    prologue
    if(launcher thread) {
        store args_post in postponed arg buffers
        memory fence
        set flag to release child
    }
    epilogue
}
```

Transformed Kernel (called from kernel)

Results:



Achieved Occupancy | Instructions per Second

Aggregation improves occupancy due to fewer coarser-grain kernels
Overlap improves instructions per second due to more work parallel work available

ECE ILLINOIS | UNIVERSIDAD DE CÓRDOBA | Hewlett Packard Enterprise | ILLINOIS