HPGMG

Nikolay Sakharnykh, 4/20/2016

NVIDIA

HPGMG

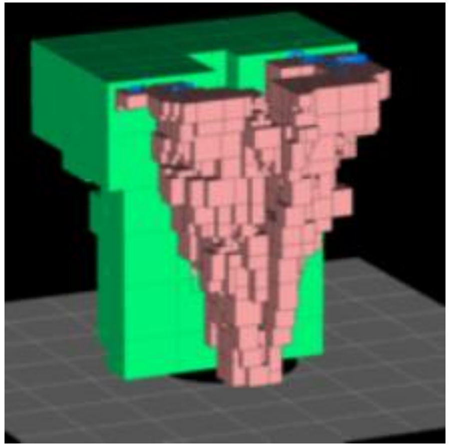
High-Performance Geometric Multi-Grid

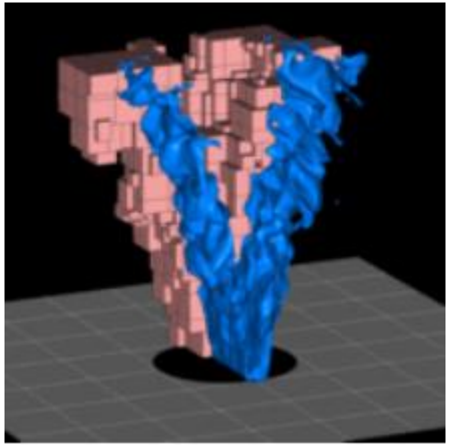
Finite-volume geometric multi-grid proxy

2nd and 4th order flux approximation

AMR and Low Mach Combustion codes

Top500 benchmarking

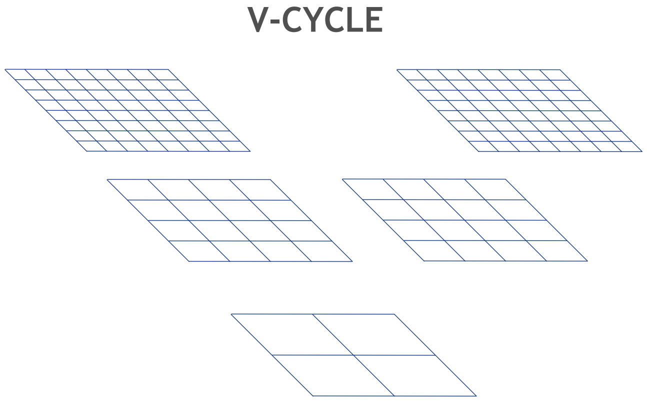


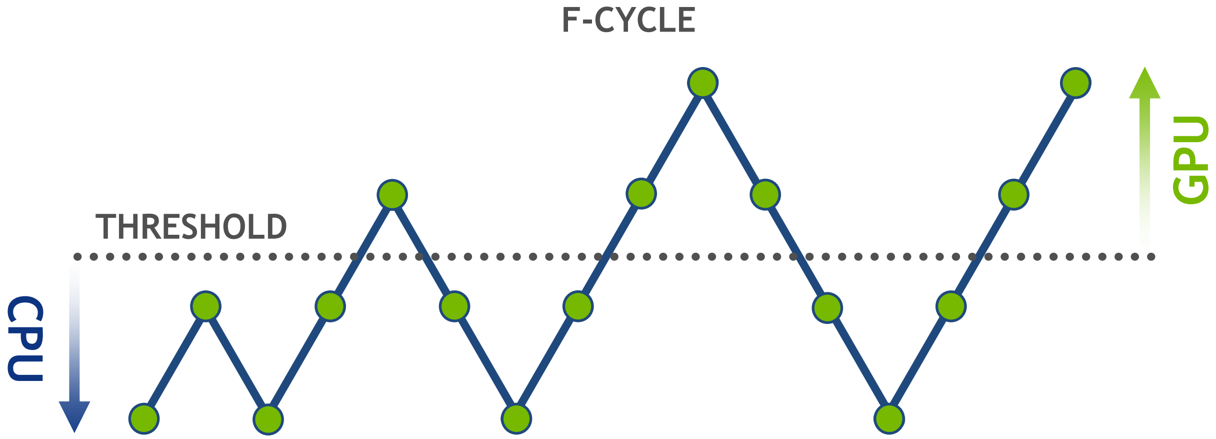


http://crd.lbl.gov/departments/computer-science/PAR/research/hpgmg/

HYBRID IMPLEMENTATION

Take advantage of both architectures



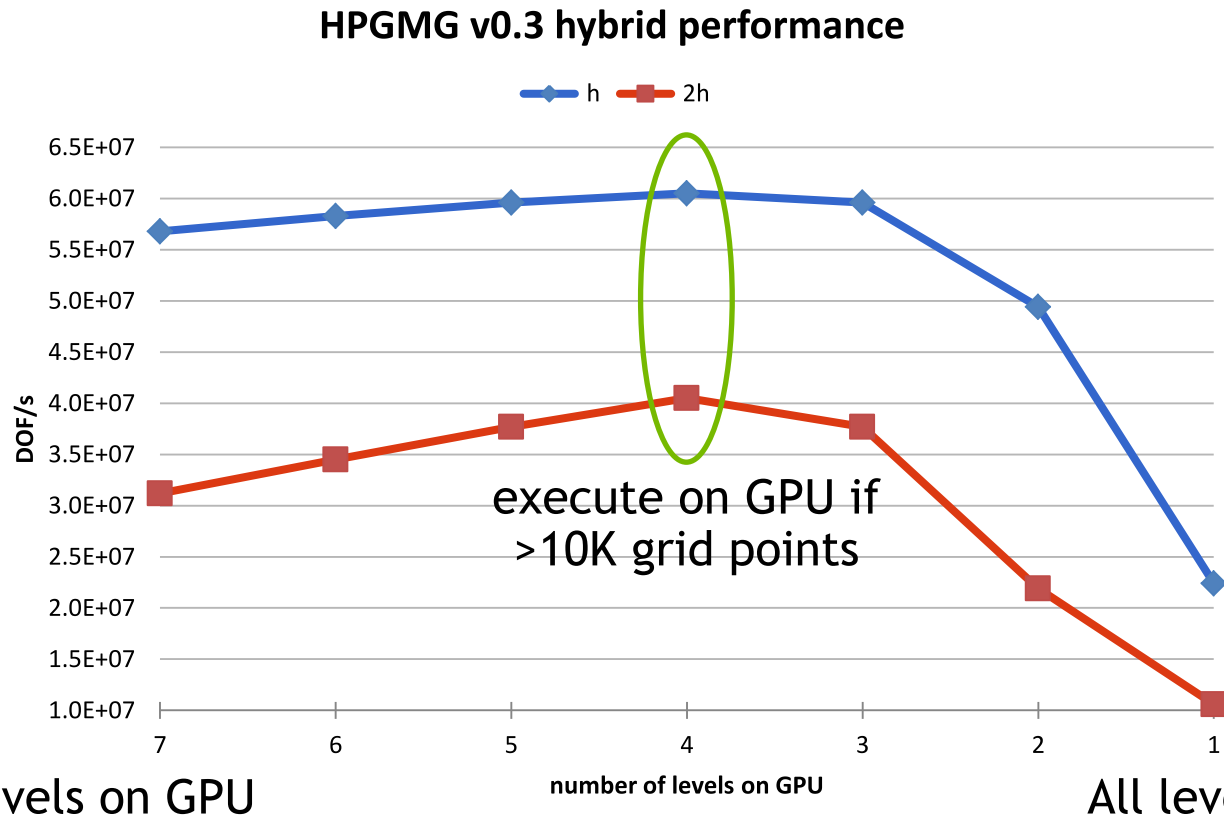


Fine levels are executed on throughput-optimized processors (GPU)

Coarse levels are executed on latency-optimized processors (CPU)

HYBRID IMPLEMENTATION

What is the optimal threshold?



execute on GPU if >10K grid points

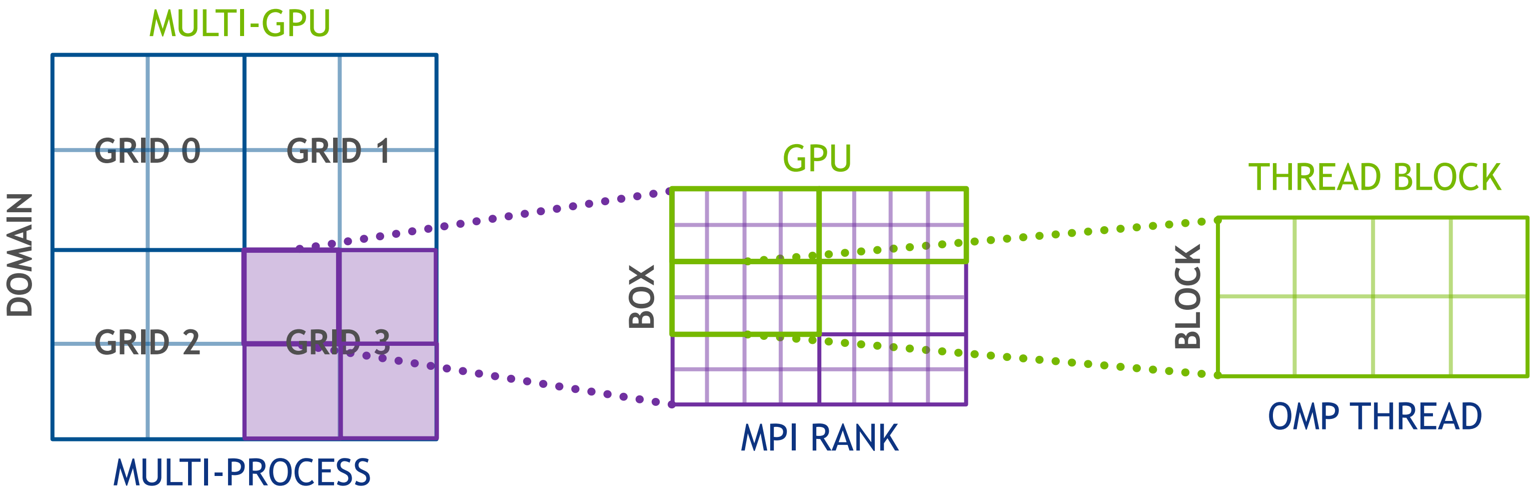
All levels on GPU

All levels on CPU

MEMORY MANAGEMENT

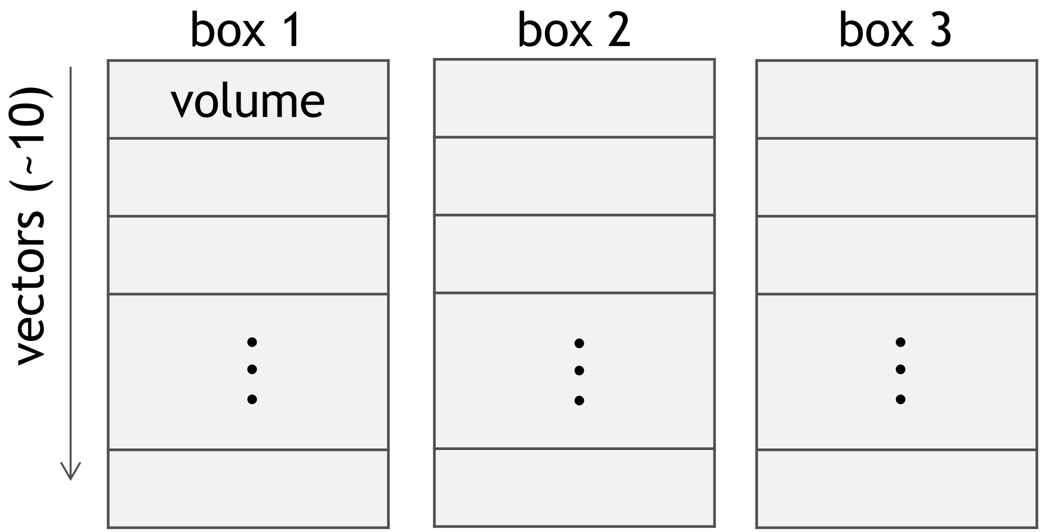
Data structures

HPGMG-FV entities naturally map to GPU hierarchy

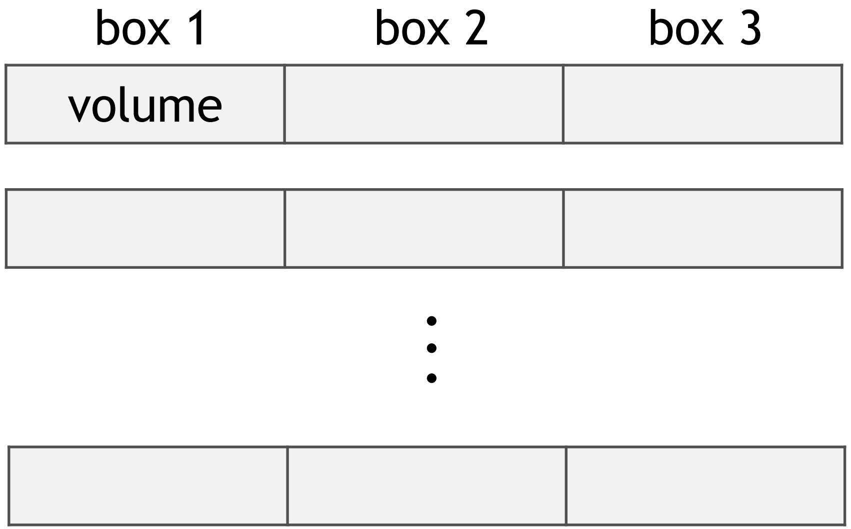


MEMORY MANAGEMENT

Data structures



Vector data within a level is disjoint
Requires **one copy per box**



Vector data within a level is contiguous
Requires **one copy per vector**

MEMORY MANAGEMENT

Using Unified Memory

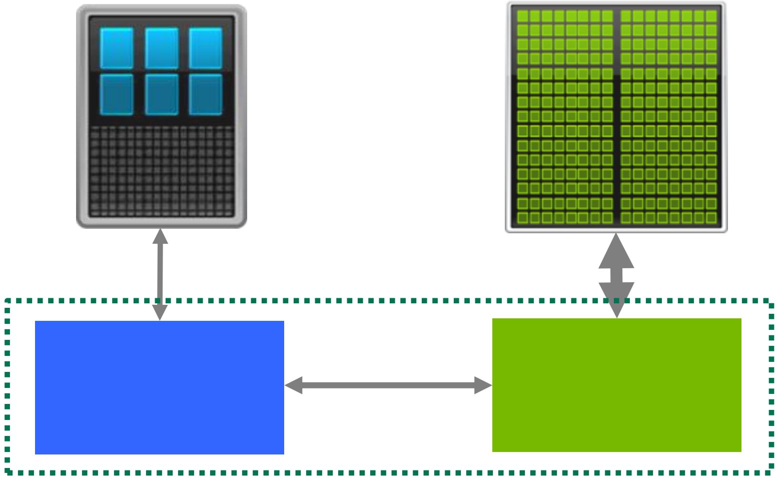
No changes to data structures

No explicit data movements

Single pointer for CPU and GPU data

Use **`cudaMallocManaged`** for allocations

Developer View With Unified Memory



Unified Memory

UNIFIED MEMORY

Simplified GPU programming

Minimal modifications to the original code:

(1) **`malloc`** replaced with **`cudaMallocManaged`** for levels accessed by GPU

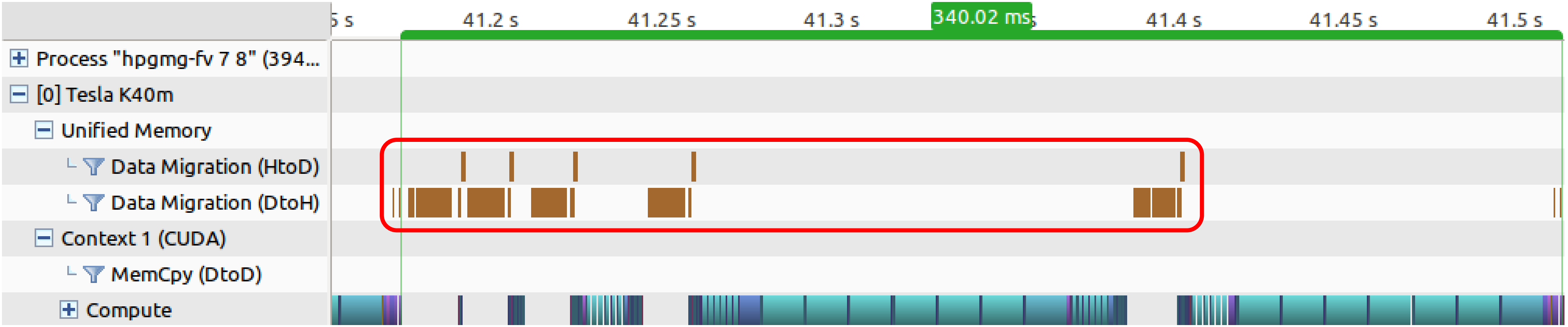
(2) Invoke CUDA kernel if level size is greater than threshold (or use directives)

```
void  smooth(level_type *level,...){
...
if(level->use_cuda) {
   // run on GPU
   cuda_cheby_smooth(level,...);
}
else {
   // run on CPU
   #pragma omp parallel for
   for(block = 0; block < num_blocks; block++)
      ...
}}
```

UNIFIED MEMORY

What about performance?

NVVP timeline for HPGMG

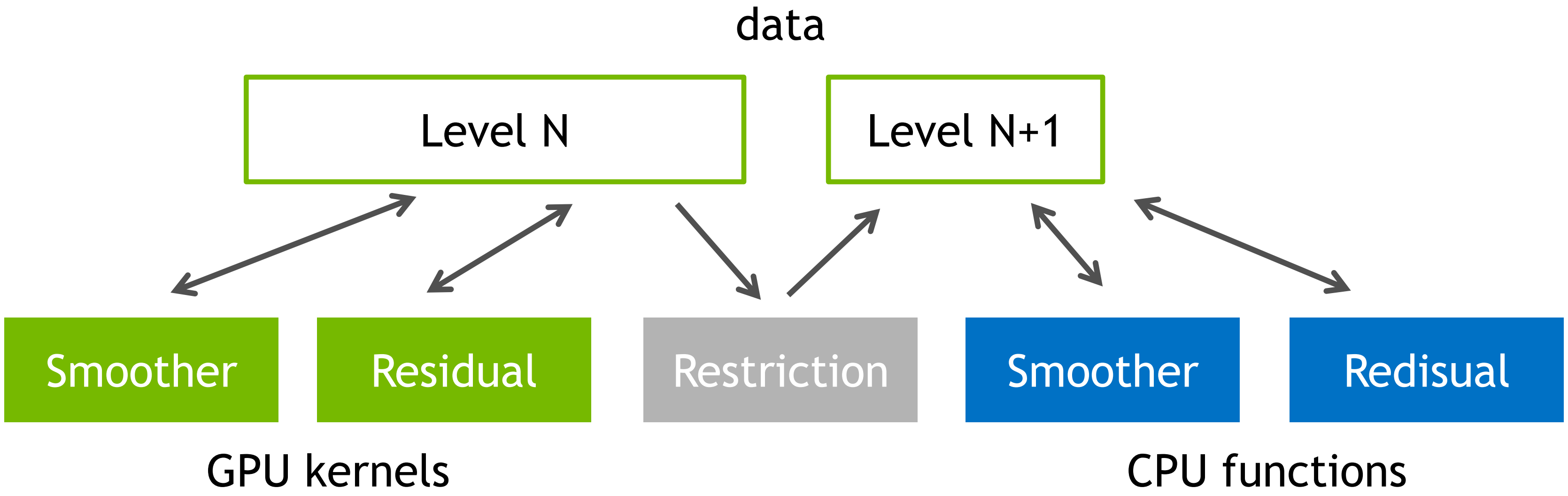


level 2 level 1 level 0

Problem: excessive faults and migrations at CPU-GPU crossover points

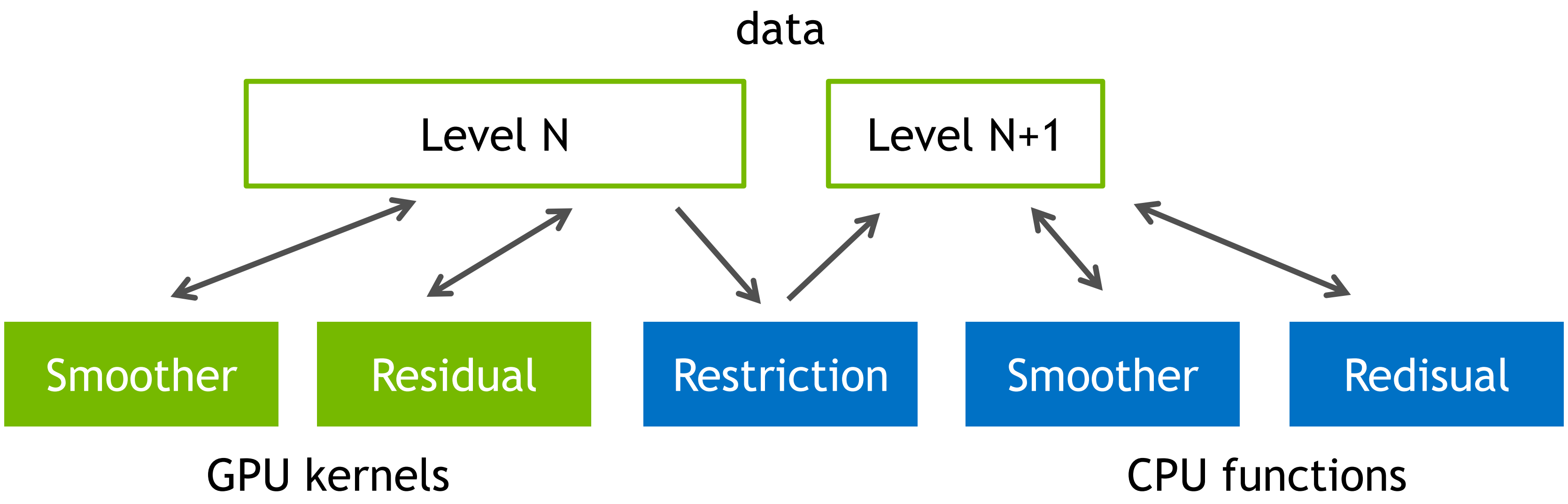
UNIFIED MEMORY

Eliminating page migrations and faults



UNIFIED MEMORY

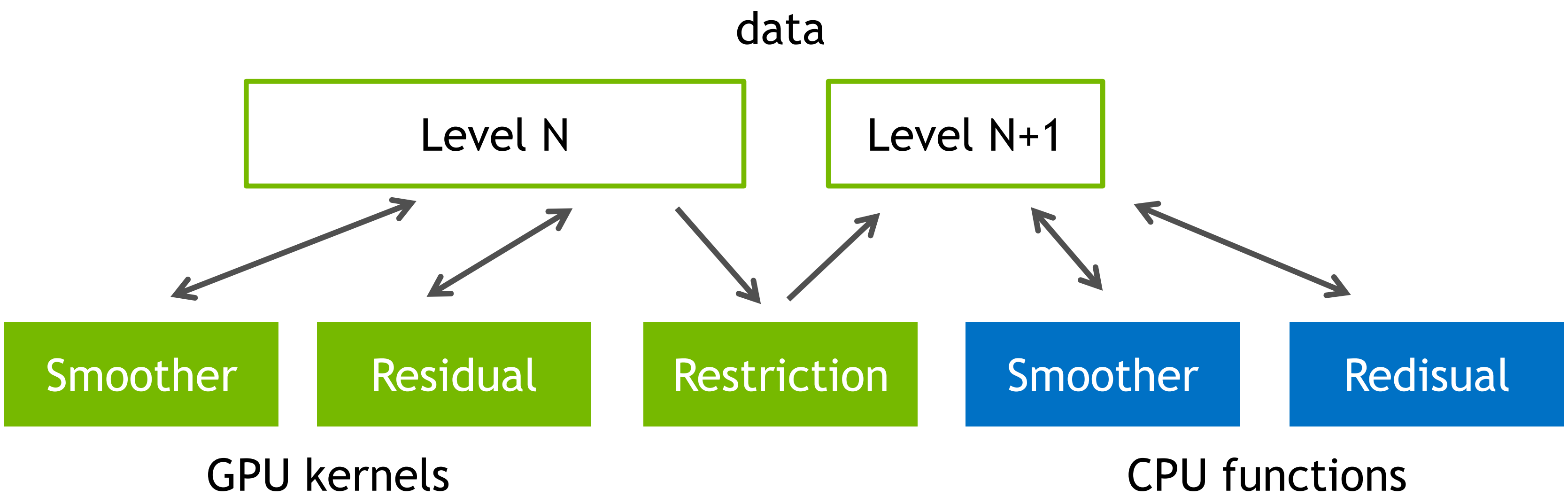
Eliminating page migrations and faults



Level N (large) is shared between CPU and GPU

UNIFIED MEMORY

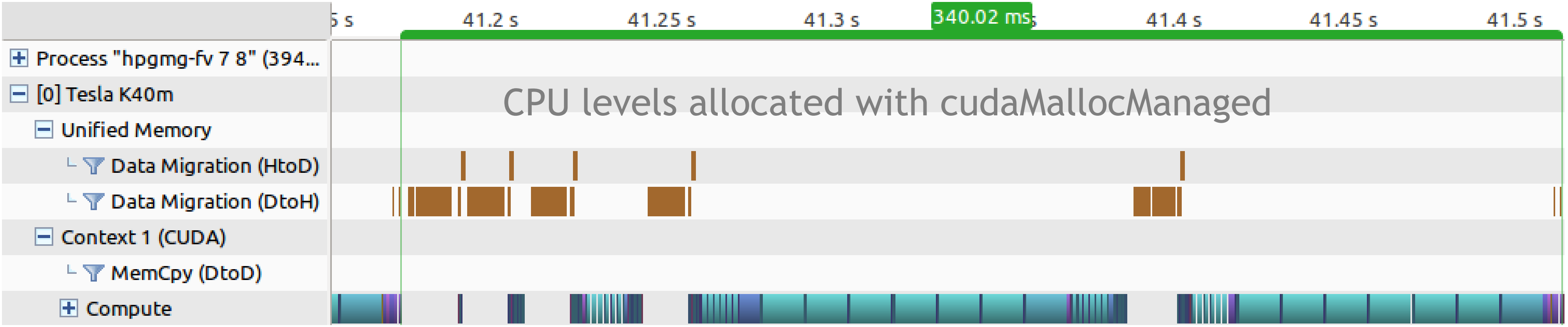
Eliminating page migrations and faults



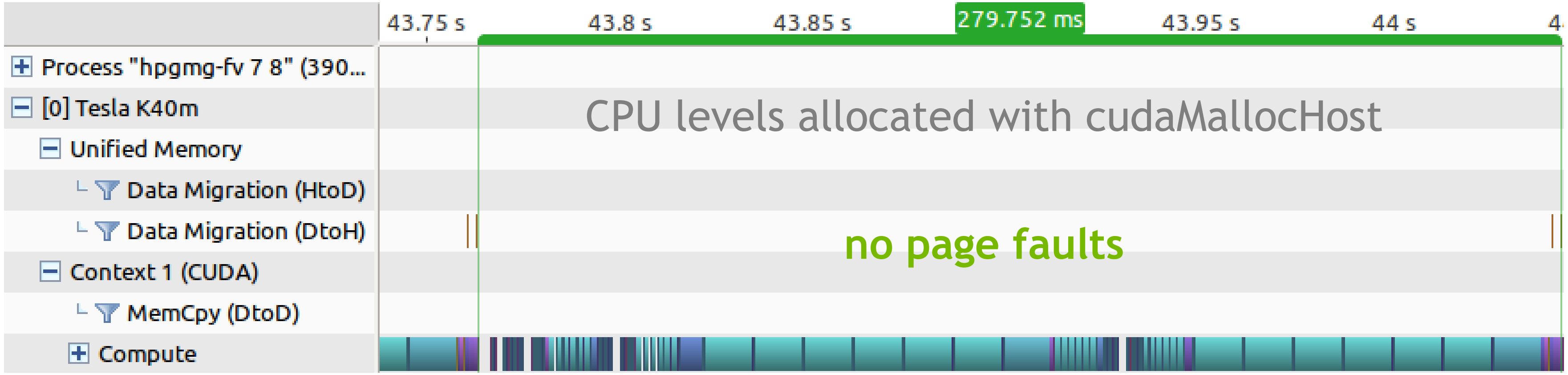
Level N+1 (small) is shared between CPU and GPU

Solution: allocate the first CPU level with cudaMallocHost (zero-copy memory)

UNIFIED MEMORY



CPU levels allocated with cudaMallocManaged



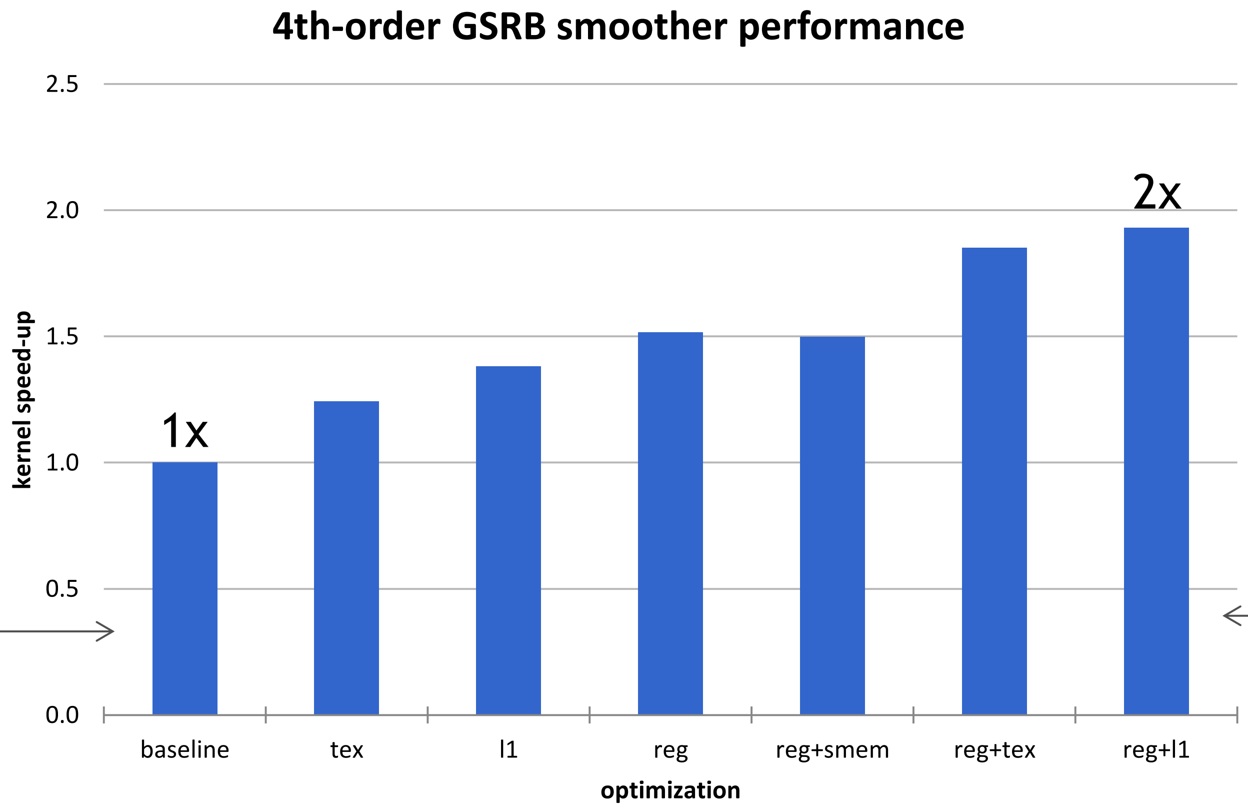
CPU levels allocated with cudaMallocHost

no page faults

+20% speed-up!

STENCILS ON GPU

Optimizations summary on Kepler



Minimal code changes

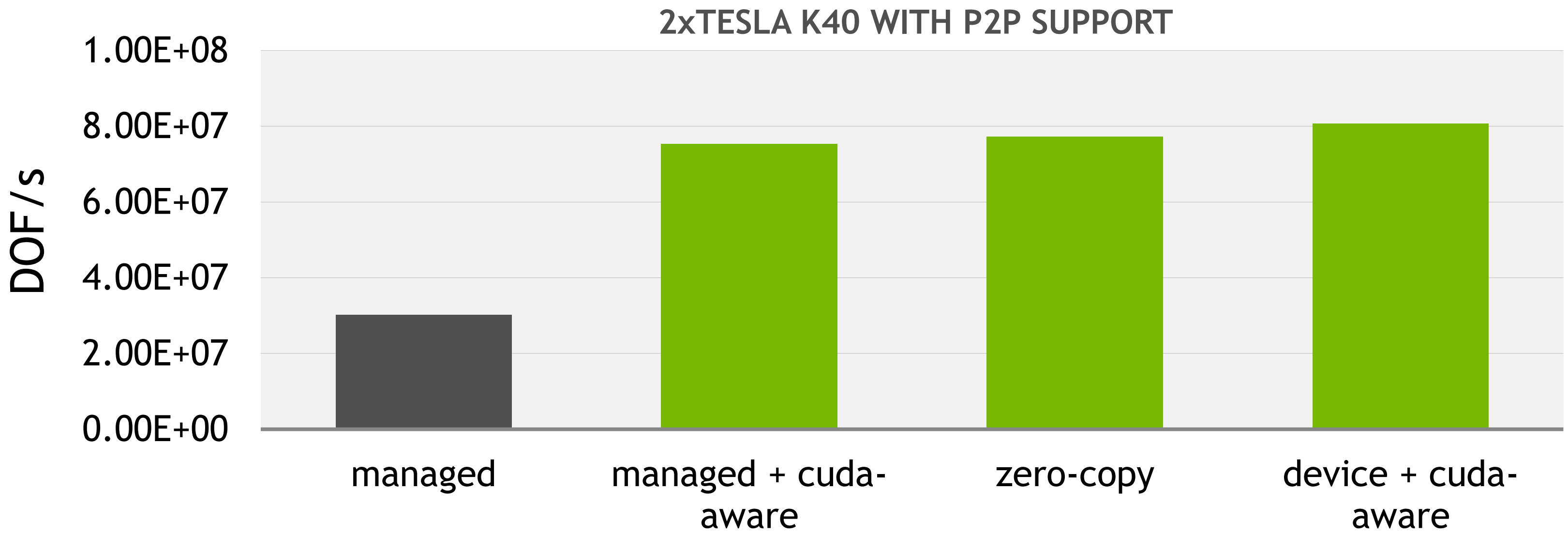
Moving data to on-chip memory

MULTI-GPU

CUDA-aware MPI

MPI buffers can be allocated with cudaMalloc, cudaMallocHost, cudaMallocManaged

CUDA-aware MPI can stage managed buffers through system/device memory



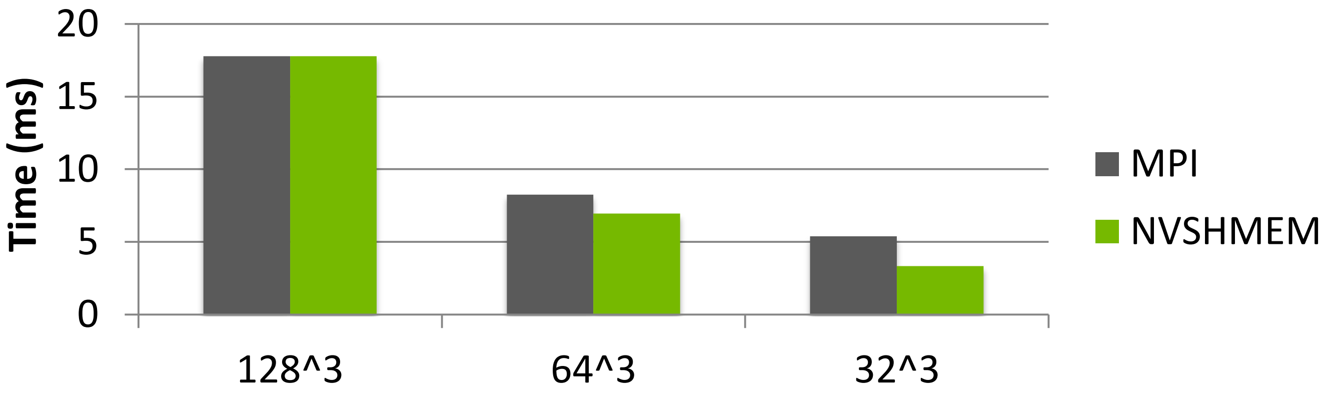
MULTI-GPU

MPI vs SHMEM

Boundary exchange code

MPI	SHMEM

CopyKernel(BOUNDARY-TO-BUFFER)
cudaDeviceSync
MPI_Irecv + MPI_Isend
CopyKernel(INTERNAL-TO-INTERNAL)
MPI_Waitall
CopyKernel(BUFFER-TO-BOUNDARY)
	CopyKernel(ALL-TO-ALL)
	shmem_barrier_all



NVIDIA