HPGMG
Nikolay Sakharnykh, 4/20/2016
HPGMG
High-Performance Geometric Multi-Grid

Finite-volume geometric multi-grid proxy
2\textsuperscript{nd} and 4\textsuperscript{th} 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?

HPGMG v0.3 hybrid performance

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
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)

```c
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?

Problem: excessive faults and migrations at CPU-GPU crossover points
UNIFIED MEMORY
Eliminating page migrations and faults

Level N

Level N+1

data

Smoother
Residual
Restriction

Smoother
Redisual

GPU kernels

CPU functions
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
is shared between CPU and GPU

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

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

Smooth
Residual
Restriction

GPU kernels

Smooth
Residual

CPU functions
UNIFIED MEMORY

CPU levels allocated with cudaMallocManaged

no page faults

+20% speed-up!
STENCILS ON GPU
Optimizations summary on Kepler

4th-order GSRB smoother performance

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

<table>
<thead>
<tr>
<th>MPI</th>
<th>SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyKernel(BOUNDARY-TO-BUFFER)</td>
<td>CopyKernel(ALL-TO-ALL)</td>
</tr>
<tr>
<td>cudaDeviceSync</td>
<td>shmem_barrier_all</td>
</tr>
<tr>
<td>MPI_Irecv + MPI_Isend</td>
<td></td>
</tr>
<tr>
<td>CopyKernel(INTERNAL-TO-INTERNAL)</td>
<td></td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td></td>
</tr>
<tr>
<td>CopyKernel(BUFFER-TO-BOUNDARY)</td>
<td></td>
</tr>
</tbody>
</table>

### Performance Comparison

![Graph comparing Time (ms) for different grid sizes (128^3, 64^3, 32^3)]

- **MPI**
- **NVSHMEM**