Directed Optimization On Stencil-based Computational Fluid Dynamics Application(s)

Islam Harb

08/21/2015

VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Agenda

- Motivation
- Research Challenges
- Contributions & Approach
- Results
- Conclusion
- Future Work

Motivation

- Computational Fluid Dynamics:
  - Physics aspects of fluid flow are represented by discretized algebraic forms/equations (e.g. Pressure, Density, Veloctiy …etc.).

- Usually, It's computational and data intensive.
  - High order numerical algorithms to study these physical aspects of high speed turbulent flows.
  - Requires long time to run (e.g. Converge and Find Solutions with very low error).

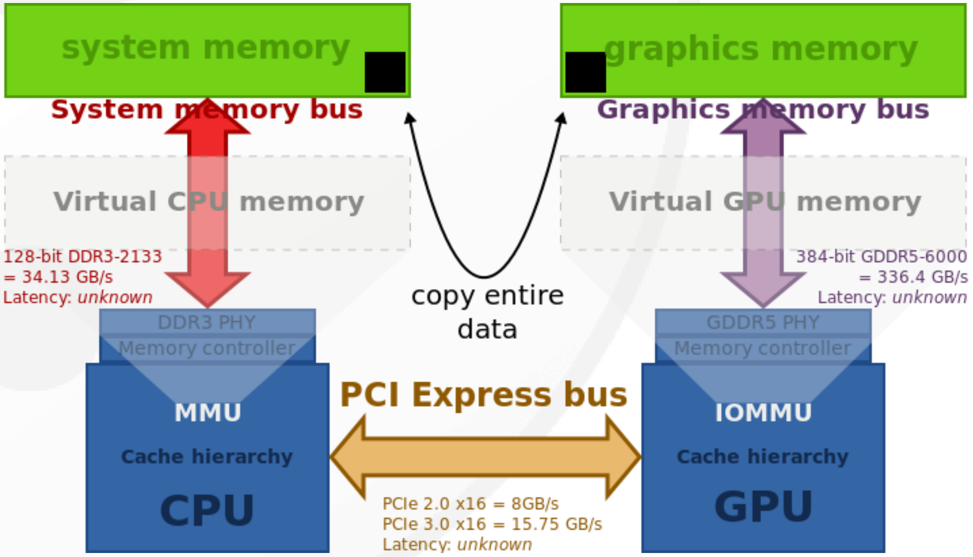
- Motivated by aerospace and mechanical engineering domains.

Motivation

- Our Main Focus:
  - The computer science view/aspects of such domain(s).
  - Optimizing these algorithms/Apps to achieve better performance.
  - Efficient parallelization of these algorithms/apps to run on multi-core platforms (e.g. NVIDIA/AMD GPUs, Intel Xeon Phi …etc.).
  - Exploring the programmability and performance aspects and trade-offs.
    - Programming models (e.g. OpenACC vs. CUDA/OpenCL)

Research Challenges

- Communications between the CPU and the GPU
  - Slow Data Transfer
  - Goals:
    1. Efficient data transfer techniques
    2. Minimize the data transfers.

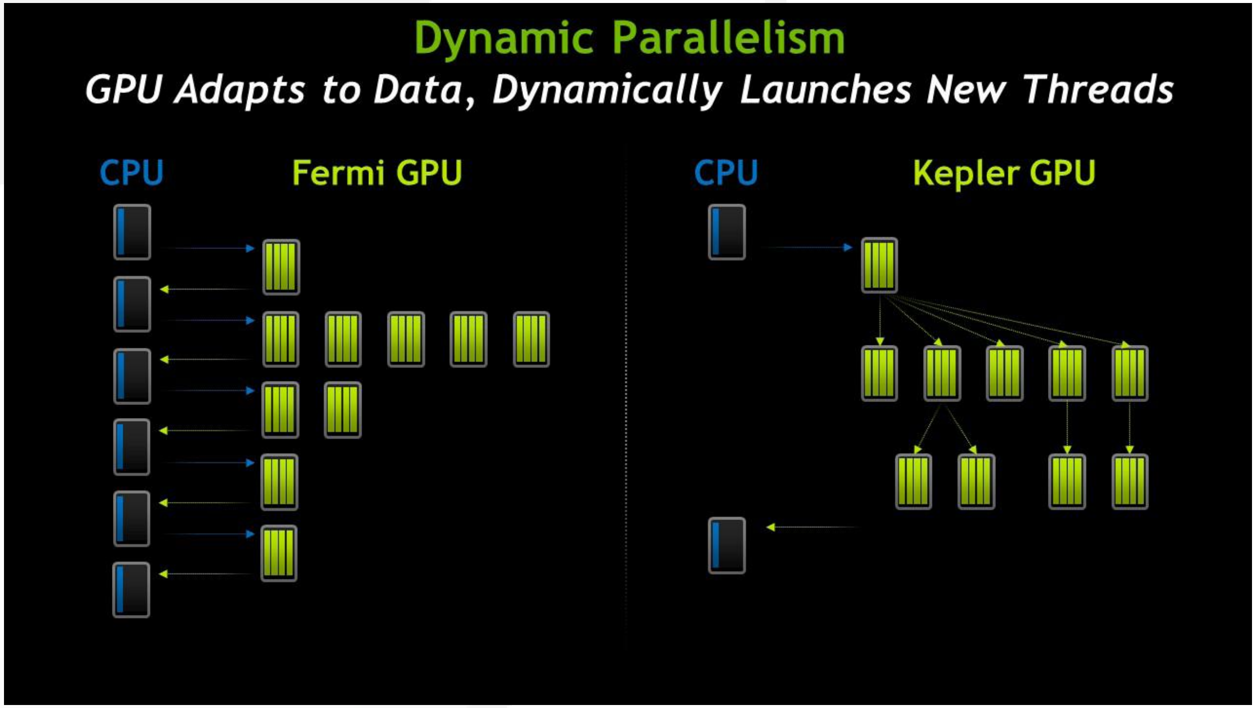


Research Challenges

- Efficient optimizations for multi-core platforms.
  - Understanding the architecture and resources limitations for each platform.
  - Shared Memory vs. Global Memory.
  - Avoid Register Pressure.
  - Identifying the optimum block-size.
  - Minimize Control Flow Divergence within Warp.

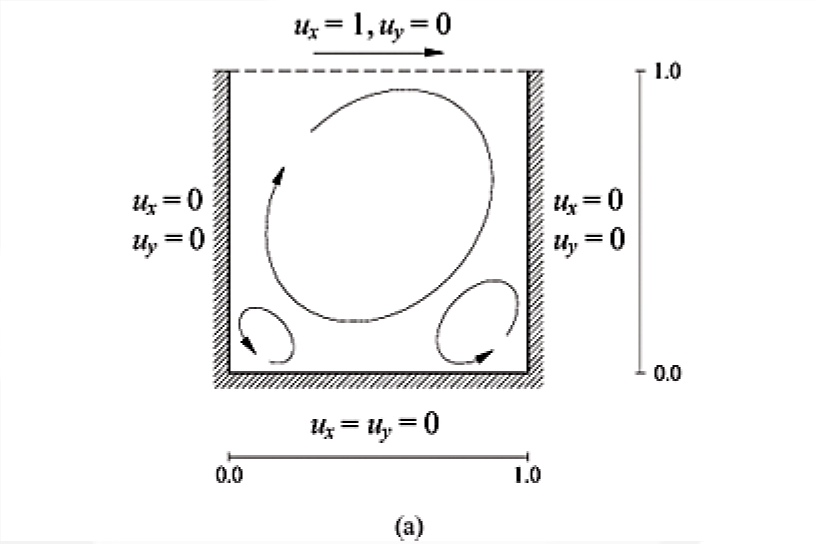
Research Challenges

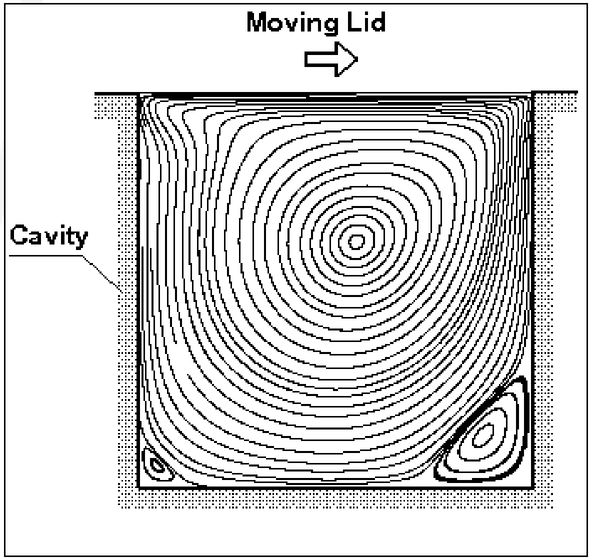
- Inter-Block Synchronization
  - Hybrid Model – CPU-based Synchronization
  - Dynamic Parallelism – GPU-based Synchronization.



Contributions & Approach

- Case Study Application
  - Lid-Driven Cavity (LDC)
    - Fluid contained in a square domain with boundary conditions on all sides.
    - Three stationary sides
    - One moving side (with velocity tangent to the side).





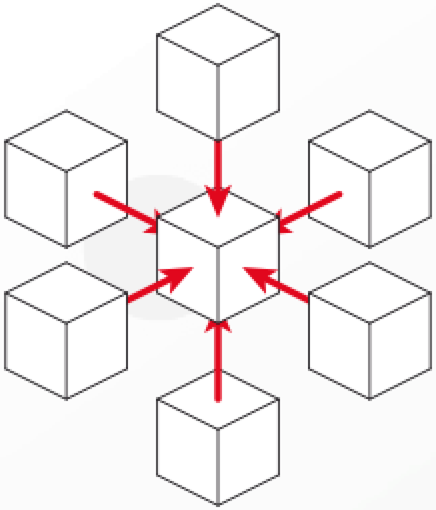
Contributions & Approach

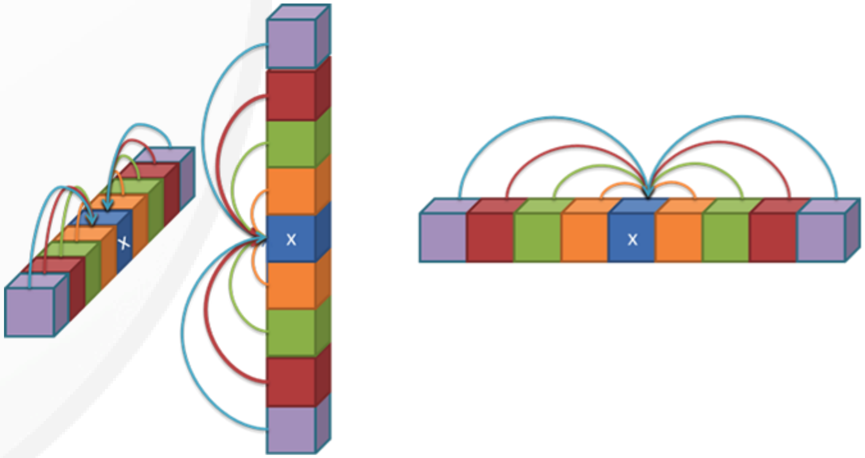
- Acceleration of the LDC
  - OpenACC Implementation
  - CUDA Implementation.
  - Source-to-Source Translation
    - CUDA to OpenCL --- Tool: CU2CL
- Examining the programmability vs. the Performance of the three programming models.
  - CUDA/OpenCL expected to perform better.
  - OpenACC easier to program (i.e. minimal to no changes to the sequential code)
  - OpenCL/OpenACC portable across different Platforms (e.g. CPU, NVIDIA GPU and AMD GPU)..

Contributions & Approach (Shared Memory)

- Shared Memory Optimization

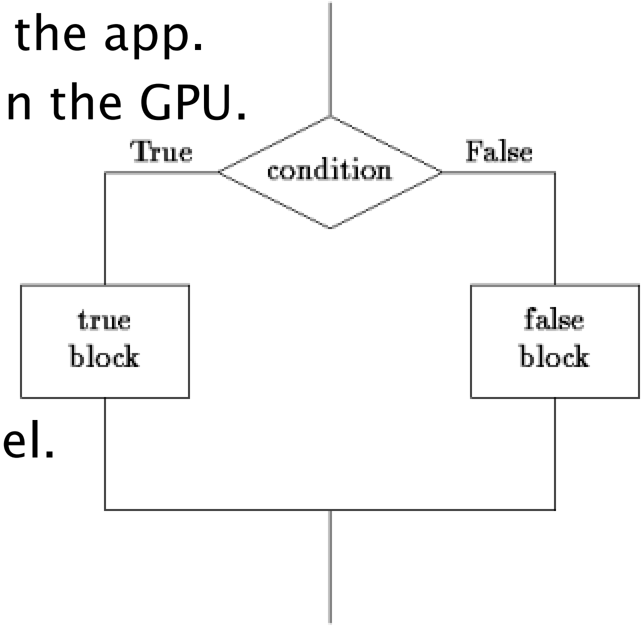
- LDC is stencil-based application.
  - Each interior cell calculations involve the neighboring cells values.
  - Data reuse benefits from accessing less expensive shared memory instead of the global memory.





Contributions & Approach (Branching Elimination)

- Control Flow Diversion/Branching
  - **Diversion:** difference execution paths in the app.
  - Different execution paths are serialized in the GPU.
  - Impact the performance negatively.



- Proposed Solution
  - Kernel Fission based on the control flow.
  - Each Flow is handled by a separate kernel.
- In the Lid-Driven Cavity
  - Interior Cells – Performed by Kernel
  - Each boundary cells (e.g. Upper/Lower Rows, most right/left Columns and Corner Cells) are performed by separate Kernels.

Contributions & Approach (Registers Usage)

- Registers are one of the critical resources of the GPU.
- Pros:
  - Very Fast Memory Access.
- Cons:
  - Limited Number per thread based on Architecture. (e.g. Kepler allows upto 255 register per thread).
  - Using big number of registers per thread leads to less active concurrent/parallel warps/threads. (i.e. Register Pressure issue).
- Analysis using performance tools (e.g. CodeXL, NVIDIA Visualizer).

Contributions & Approach (Registers Usage)

- Register Pressure Solutions
  - Register Pressure: Usage of large number of registers per thread that leads to a contention.
  - Compiler-based
    - Capping the register per thread limit to specific number (e.g. For NVIDIA, -maxrregcount=<value>).
    - Not always improves the performance.
  - Algorithmic-based
    - Kernel Fission in order to reduce the workload per thread.
    - Not always feasible – depends on the instructions dependency.
    - May add redundant computation overhead.

Contributions & Approach (GPU/CPU Communication)

- Data Transfer between GPU and CPU
  - If data is small enough to reside in the GPU global memory.
    - Single copy in at the beginning – Before computation
    - Single copy out at the end -- After computation
  - Otherwise (Data can’t reside in the GPU Memory)
    - Out of my scope so far.
    - Data partitioning: Pipeline data transfer with computation (if possible).
    - Others.

Contributions & Approach (GPU/CPU Communication)

- Data Transfer between GPU and CPU
  - Data/Updates need to be exchanged between the CPU and the GPU.
  - If this data will be processed by the CPU using some arithmetic operations.
    - Then almost some of the programming models allow "reduction" operations (e.g. OpenACC)
    - Other programming models, "reduction" techniques can be manually implemented (e.g. CUDA and OpenCL).

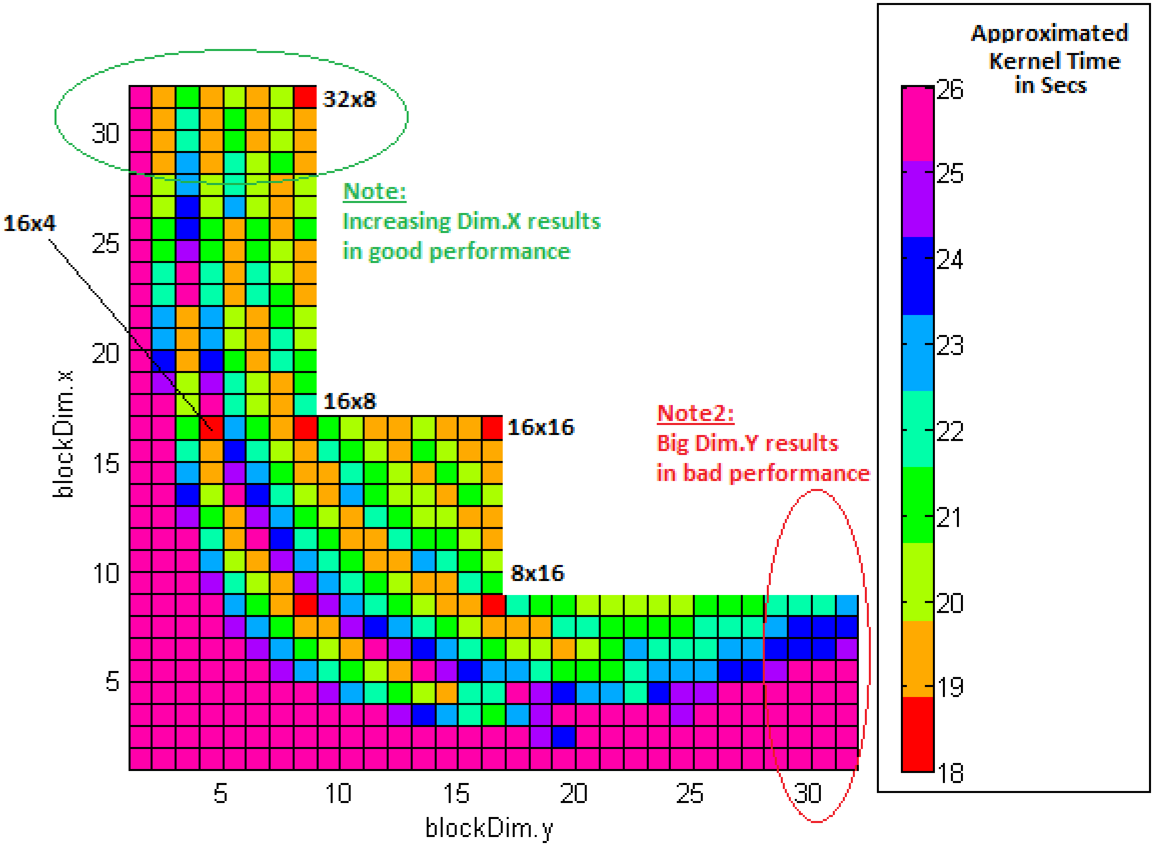
Contributions & Approach (Synchronization)

- Inter-Block Synchronization
  - It is an overhead in the accelerators world.
- Several Techniques/Models can be used for sync, Our Focus:
  - Hybrid Mode : CPU handles all the kernel launches.
  - Dynamic Parallelism Mode: GPU is in charge.
  - So far, Hybrid mode outperforms the Dynamic parallelism.
  - On the other hand, Dynamic Parallelism maybe used to save power consumption.

Results

Block Size Exploration

- OpenCL running on AMD Radeon HD 7970

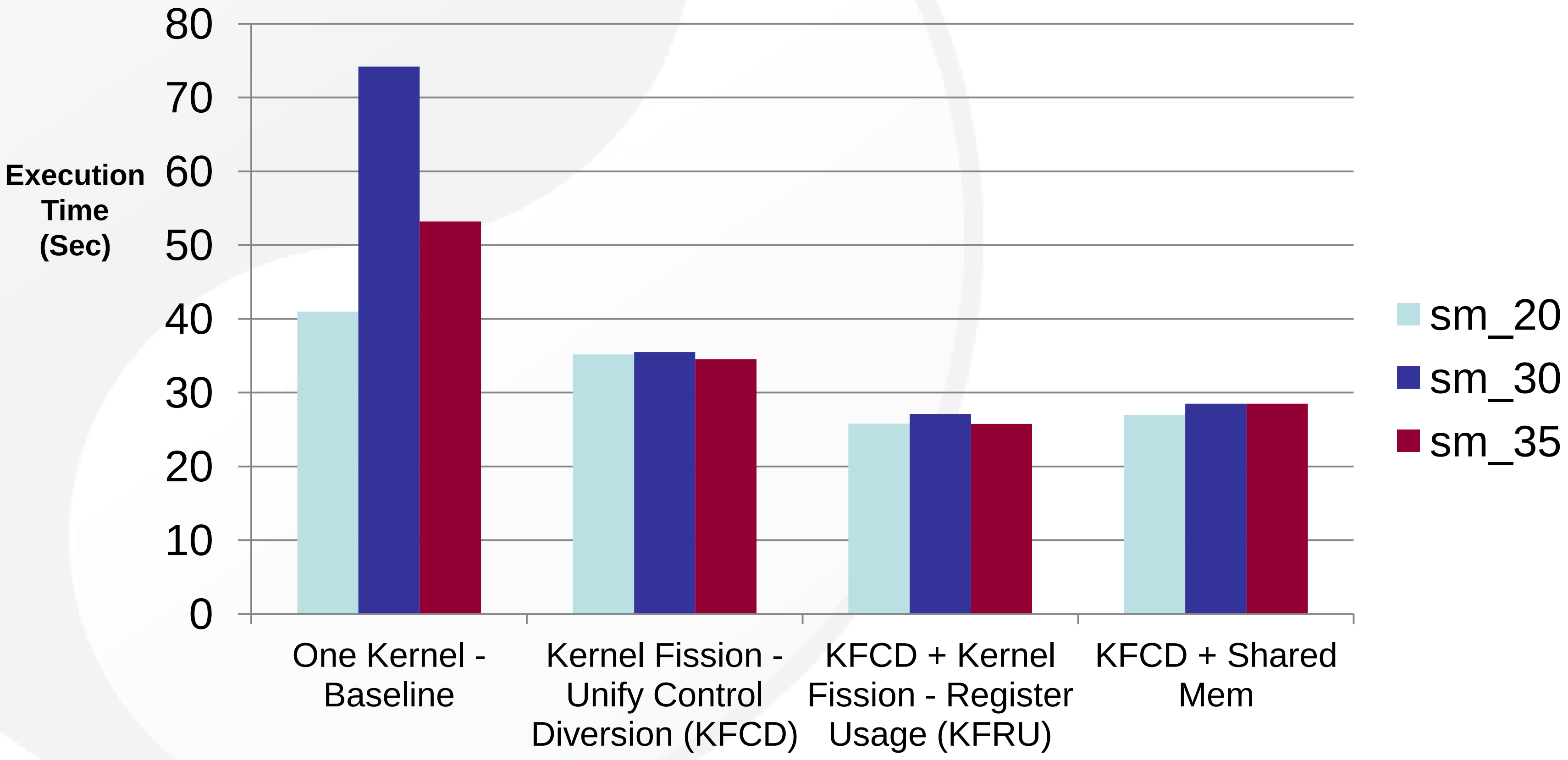


Note:
Increasing Dim.X results in good performance

Note2:
Big Dim.Y results in bad performance

Execution Time on K20c

- LDC Execution Time over multiple architectural generations on NVIDIA GPUs.



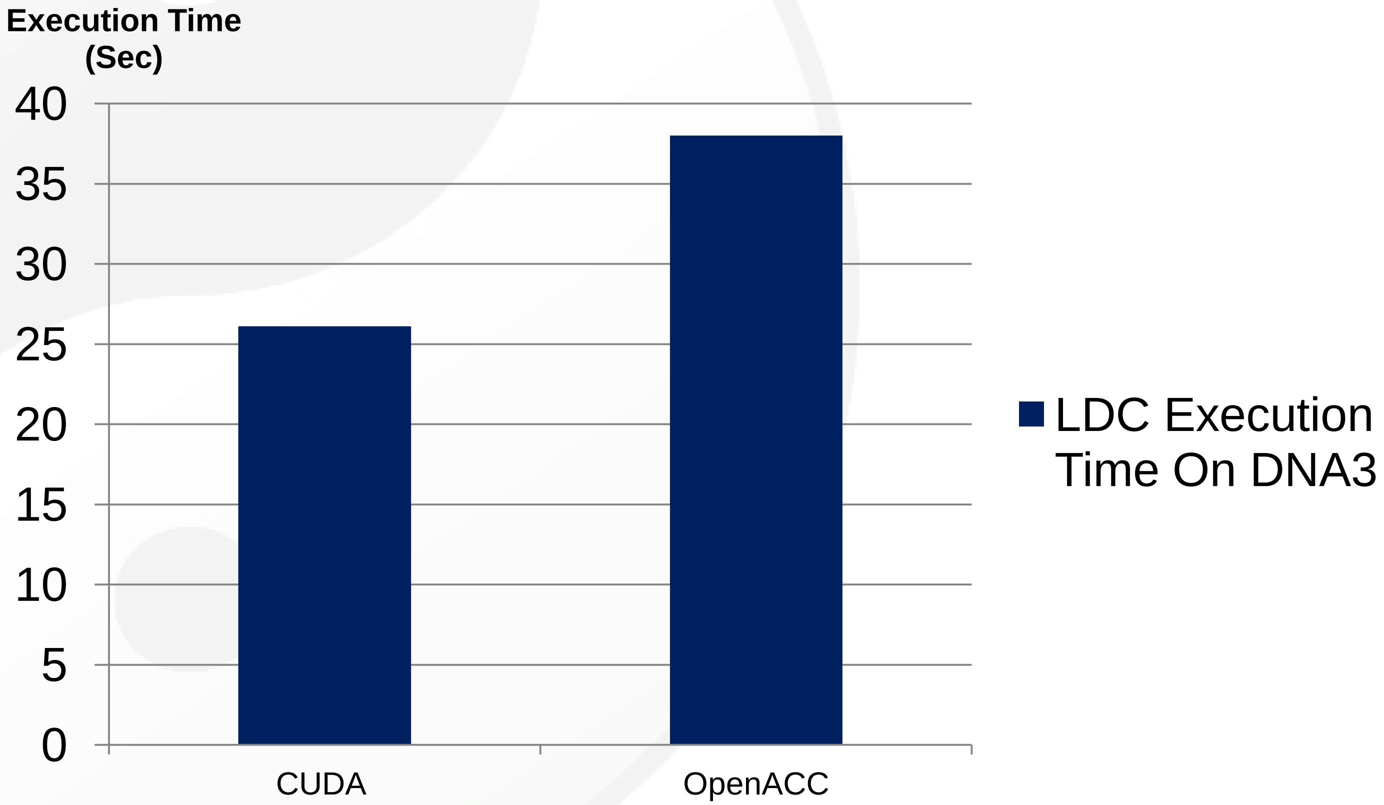
Programmability vs. Performance

- High level programming models (e.g. OpenACC)
  - Easy to use.
  - Less control over the architecture resources.
  - Most Likely Portable across Platforms (e.g. CPU, AMD/NVIDIA GPUs).
  - Lower Performance.

- Low level programming models (e.g. CUDA, OpenCL)
  - Difficult to use.
  - More control over the architecture resources.
  - Less/Not Portable across Platforms (e.g. CPU, AMD/NVIDIA GPUs).
  - Higher Performance with careful optimizations.

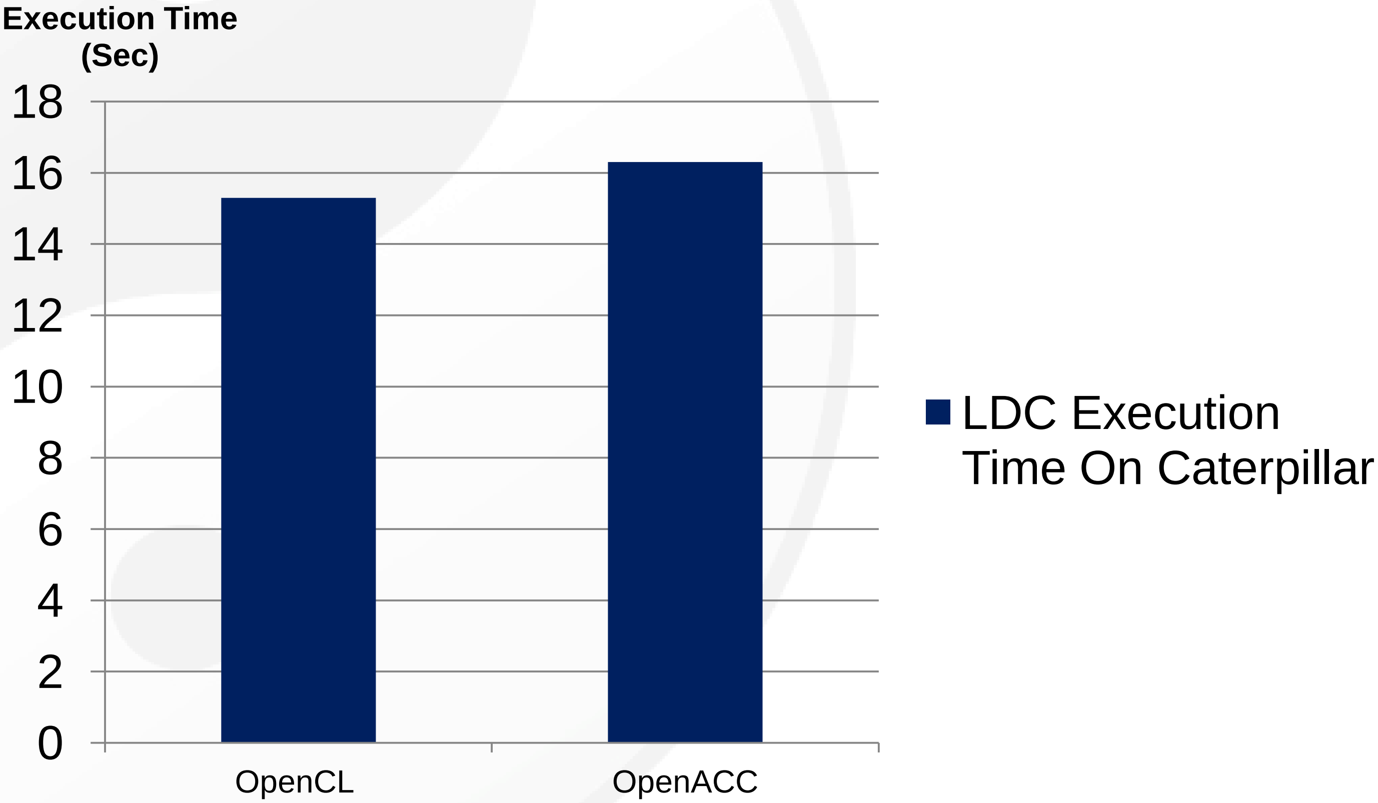
OpenACC vs. CUDA on K20c

- CUDA outperforms the OpenACC with ~1.3x



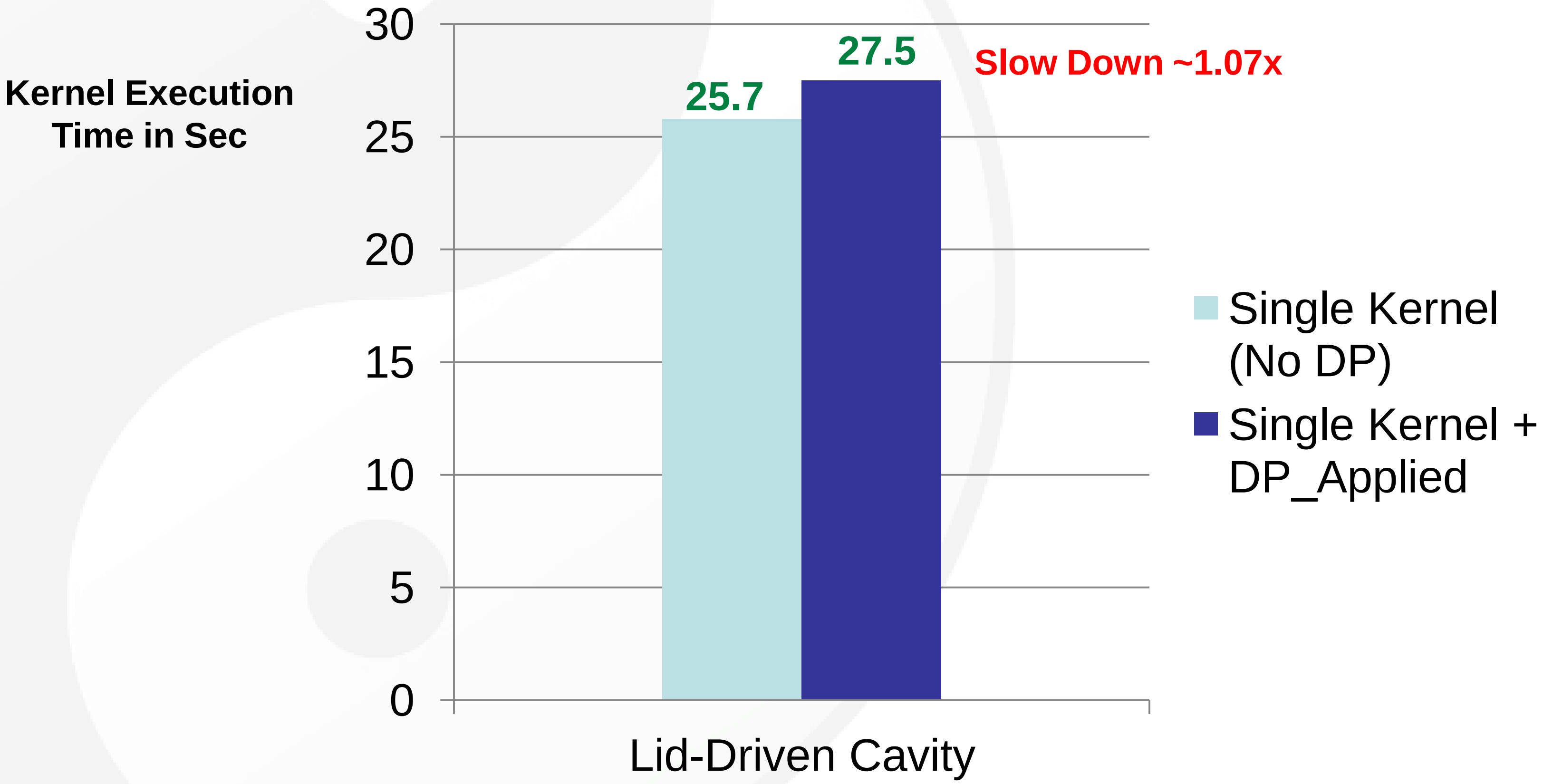
OpenACC vs. OpenCL on HD 7970

- OpenCL outperforms the OpenACC with ~1.07x



Dynamic Parallelism (DP) on K20c

- Kernel Fission – Register Usage



Conclusion

- Recap
  - Computational Fluid Dynamics (CFDs) is a driving force in the R&D and the manufacturing of many industrial processes.
  - Stencil patterns are heavily used in CFDs.
  - Directed optimizations for stencils are needed.
    a. Shared Memory
    b. Data Transfer between GPU and CPU
    c. Inter-Block synchronization.
    d. Programming models explorations (Programmability vs. Performance)
    e. Registers Usage and Control flow branching
    f. Block Size Exploration.

Future Work

- Dynamic Parallelism Exploration
  - Performance vs. Power Consumption.
- Further Performance Tuning for other CFD/Stencil applications. Related to expanding the benchmark.
- Further optimization, analysis and insights on the OpenCL/CUDA Lid-Driven Cavity Code.