CCMT

# FPGA Acceleration of Fluid-Flow Kernels

Ryan Blanchard

Greg Stitt, Herman Lam

Center for Compressible Multiphase Turbulence (CCMT)
ECE Department, University of Florida

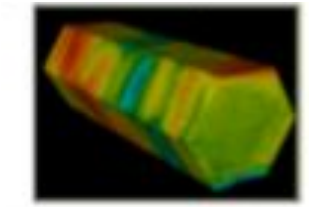
UF UNIVERSITY of FLORIDA


NNSA

# Overview

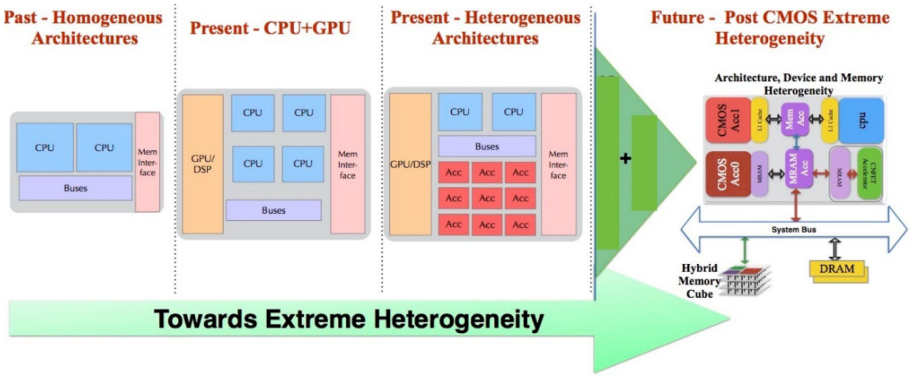
- Homogeneous systems inherently limited by processor performance
  - Hardware Accelerators offer performance and power benefits

- Case study: accelerate HPC physics app with FPGA
  - Develop custom HW pipeline for compute-bound kernel
  - Compare performance with CPU and GPU
  - FPGA brings versatile performance and attractive energy considerations



Reactor Assembly

- Projected performance to notional FPGA-accelerated HPC system

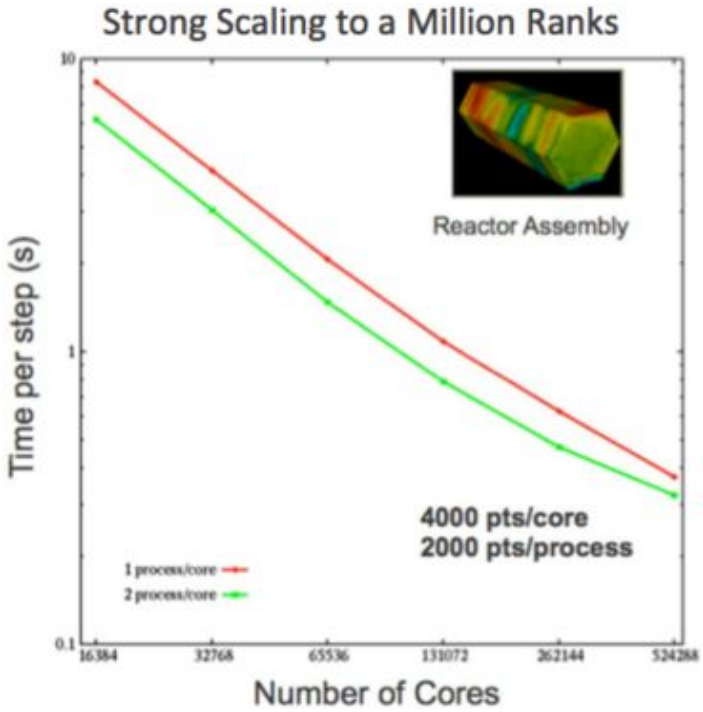


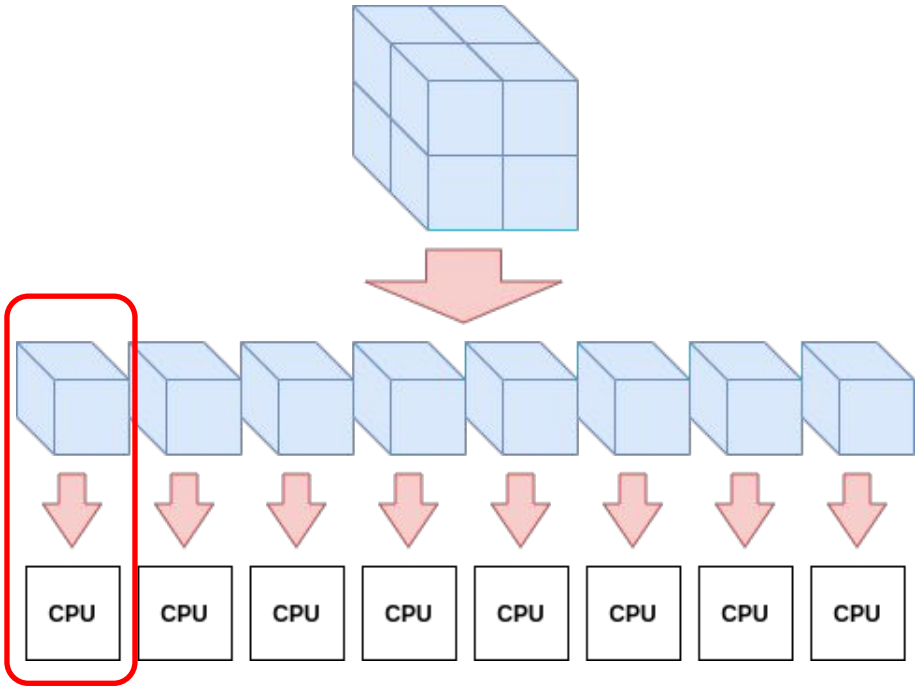



* https://orau.gov/exheterogeneity2018/2018-Extreme- Heterogeneity-BRN-report-final.pdf

# Motivation: Application

- Nek5000: physics HPC app renowned for high scalability*
- Coarse-grained parallelism obtained by dividing large volume into many sub-spaces
- Can we go beyond CPU scaling?
  - Extract fine-grained parallelism inherent in sub-space computations

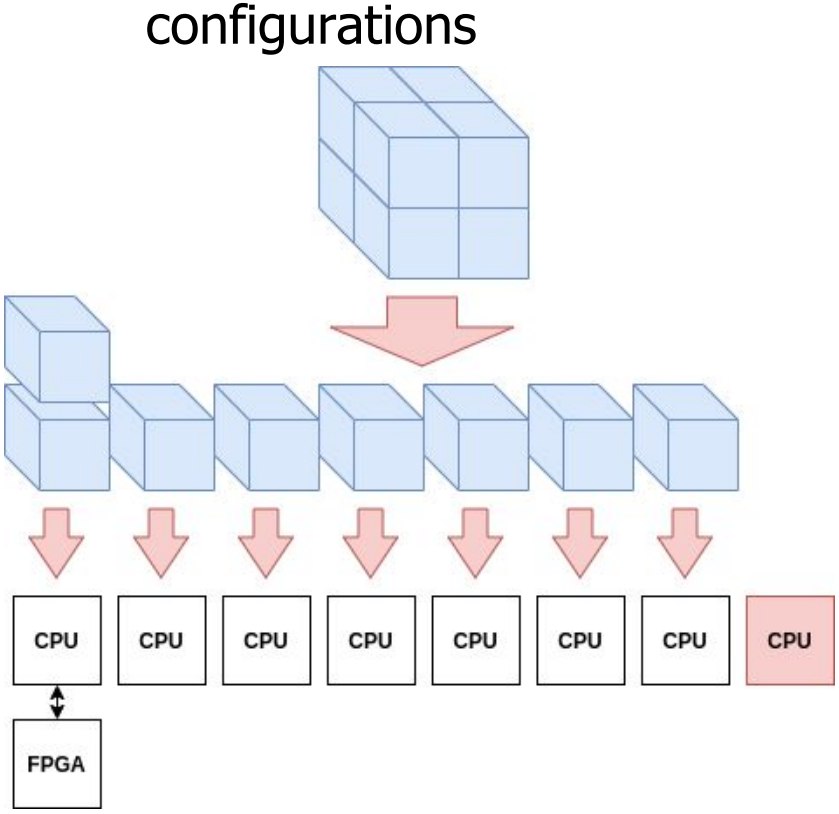


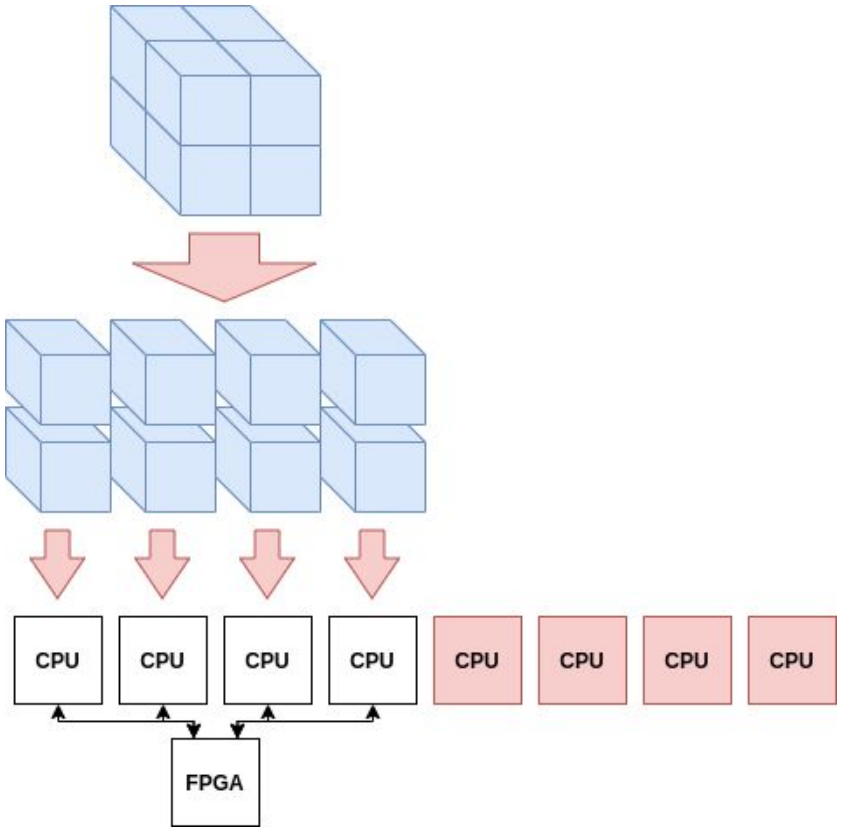




*https://www.mcs.anl.gov/~oanam/old_page/index.php/Scaling

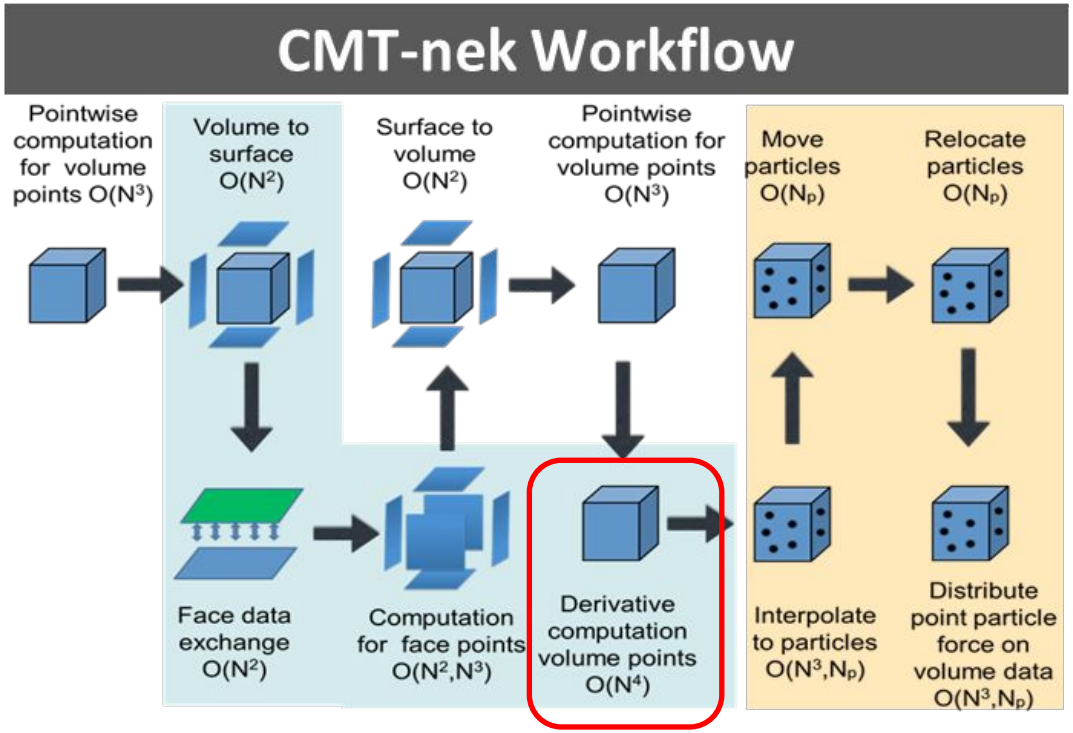
# Motivation: FPGA Accelerator

- We show FPGA can accelerate CPU core's performance
- We project how this performance can complement multicores in future heterogeneous systems
  - Design Space Exploration (DSE) of notional accelerator configurations



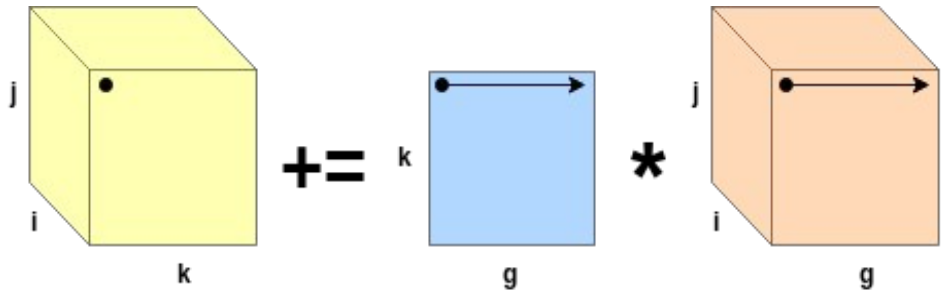
# Target Compute Kernel




```
/* For each timestep: */
for ( t = 0; t < TIMESTEPS; t++ ) {

  if (rank == PROBED_RANK) tA = now();

  /* For each of the three 'stages': */
  for (r = 0; r < RK; r++) {

    /* ------------------------- Compute (A) -------------------------

    /* For each element owned by this rank: */
    for ( e = 0; e < ELEMENTS_PER_PROCESS; e++ ) {

      /* For each block in the element: */
      for ( b = 0; b < PHYSICAL_PARAMS; b++ ) {

        /* Generate Ur, Us, and Ut. */
        operation_conv(elements_Q[e]->B[b], RX, Rx, Hy, Hz, Ur, Us, Ut);

        /* Perform the three derivative computations (R, S, T). */
        operation_dr(kernel, Ur, Vr);
        operation_ds(kernel, Us, Vs);
        operation_dt(kernel, Ut, Vt);

        /* Add Vr, Vs, and Vt to make R. */
        operation_sum( Vr, Vs, Vt, elements_R[e]->B[b] );
```

- Derivative computations are compute bottleneck with $O(N^4)$ complexity
  - Three derivatives in a row, one for each 3D direction of flow
  - Opportunity for acceleration with custom FPGA circuit

# Target Compute Kernel: CPU

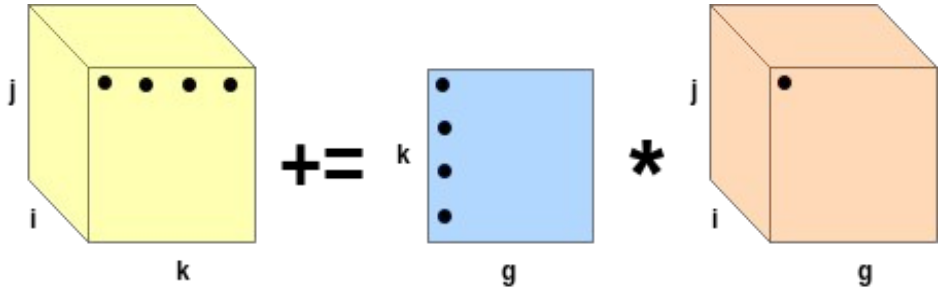


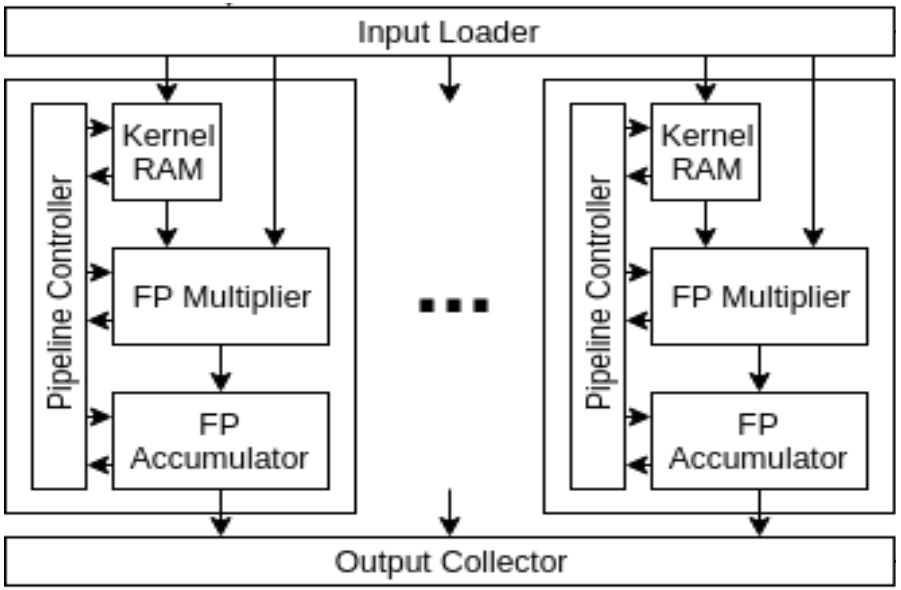

**Algorithm 1** Partial Derivative Compute Kernel

```
1:  for derivative = dr, ds, dt do
2:      for i = 1, ..., N do
3:          for j = 1, ..., N do
4:              for k = 1, ..., N do
5:                  for g = 1, ..., N do
6:                      if dr then
7:                          C_r[i][j][k] += A[i][g] * B[g][j][k]
8:                      end if
9:                      if ds then
10:                         C_s[i][j][k] += A[j][g] * B[i][g][k]
11:                     end if
12:                     if dt then
13:                         C_t[i][j][k] += A[k][g] * B[i][j][g]
14:                     end if
15:                 end for
16:             end for
17:         end for
18:     end for
19: end for
```

- Primary computation: 3D x 2D matrix multiply
  - $N^3$ outputs requiring N MACs each => $O(N^4)$ complexity
  - Memory intensity: each $N^3$ input accessed N times

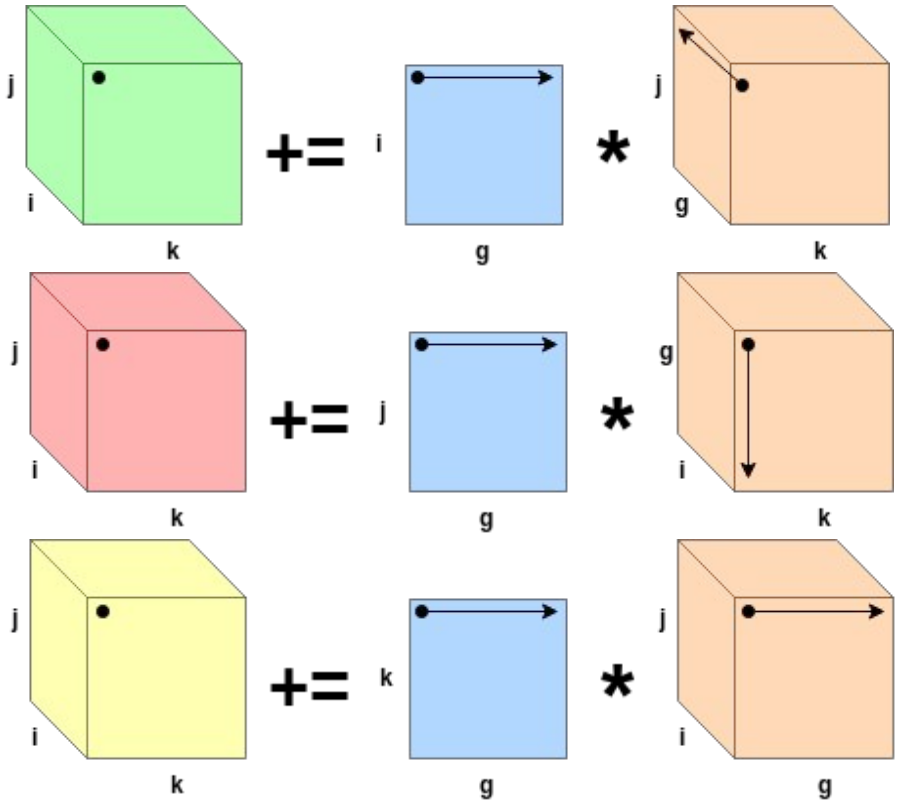
# Target Compute Kernel: FPGA Pipeline




- N parallel Multiply-Accumulate (MAC) pipelines
  - Pre-load each path with 1 row of 2D kernel matrix
  - Stream input ternix, performing all MACs in parallel
    - Each input read only once per derivative
  - Produces results every clock cycle once pipeline is saturated
  - Wide + deep parallelism + reduced memory intensity yields significant performance benefits

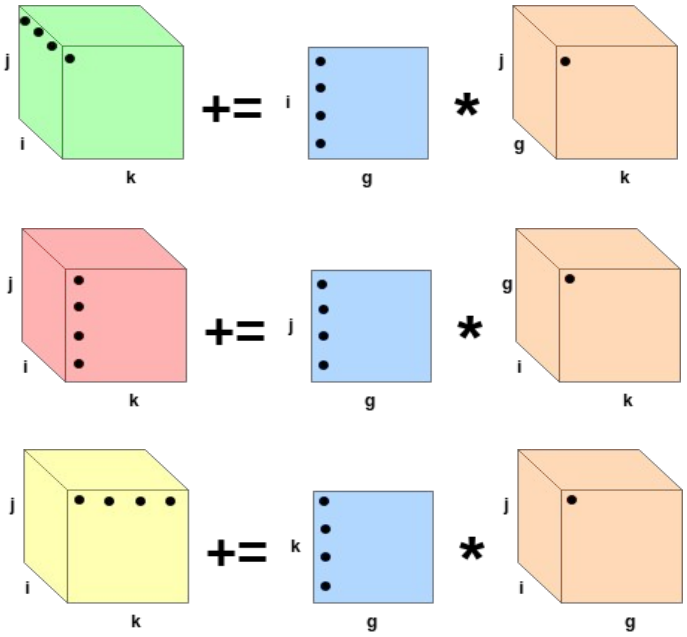
# 3D Complexity

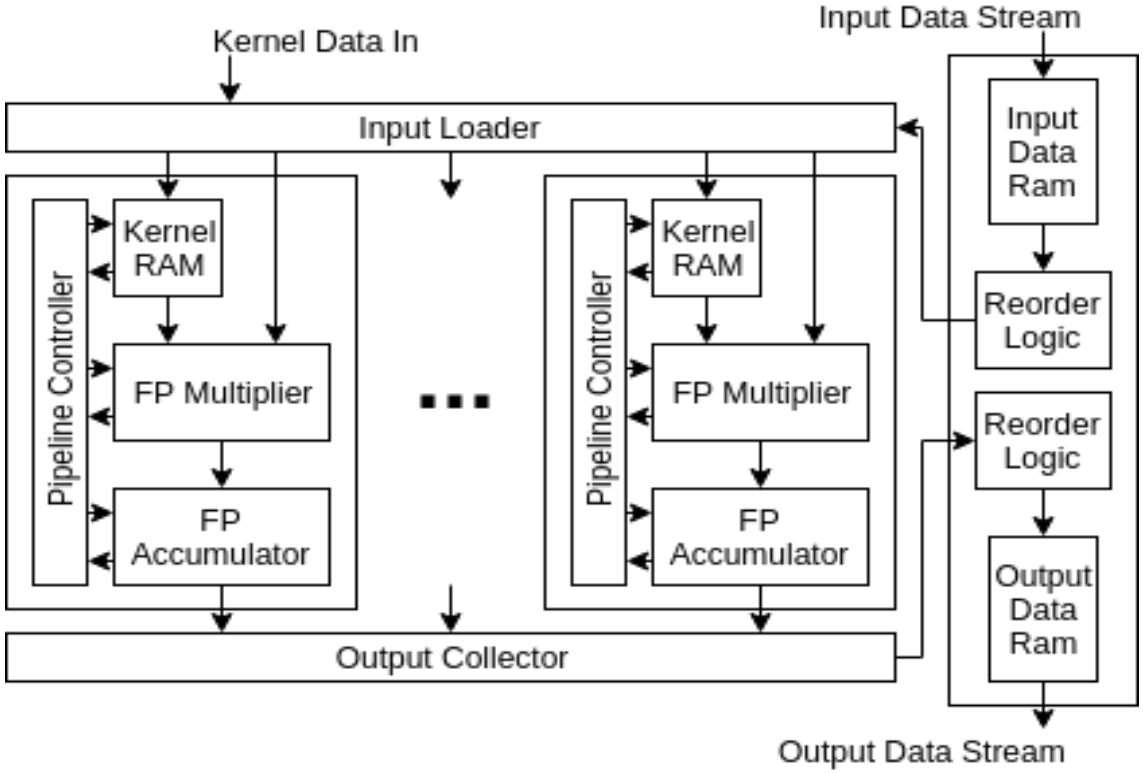



```
void operation_dr(matrix A, ternix B, ternix C)
/* Perform the R axis derivative operation, with kernel A and result C. */
{
  zero_ternix(C);

  int k, j, i, g;

  for (k = 0; k < ELEMENT_SIZE; k++) {
    for (j = 0; j < ELEMENT_SIZE; j++) {
      for (i = 0; i < ELEMENT_SIZE; i++) {
        for (g = 0; g < ELEMENT_SIZE; g++) {
          C->T[i][j][k] += A->M[i][g] * B->T[g][j][k]; } } } }
}

void operation_ds(matrix A, ternix B, ternix C)
/* Perform the S axis derivative operation, with kernel A and result C. */
{
  zero_ternix(C);

  int k, j, i, g;

  for (k = 0; k < ELEMENT_SIZE; k++) {
    for (j = 0; j < ELEMENT_SIZE; j++) {
      for (i = 0; i < ELEMENT_SIZE; i++) {
        for (g = 0; g < ELEMENT_SIZE; g++) {
          C->T[i][j][k] += A->M[j][g] * B->T[i][g][k]; } } } }
}

void operation_dt(matrix A, ternix B, ternix C)
/* Perform the T axis derivative operation, with kernel A and result C. */
{
  zero_ternix(C);

  int k, j, i, g;

  for (k = 0; k < ELEMENT_SIZE; k++) {
    for (j = 0; j < ELEMENT_SIZE; j++) {
      for (i = 0; i < ELEMENT_SIZE; i++) {
        for (g = 0; g < ELEMENT_SIZE; g++) {
          C->T[i][j][k] += A->M[k][g] * B->T[i][j][g]; } } } }
}
```

- Same input data traversed in 3 different directions
  - Send input data to FPGA only once per 3 derivative calculations
  - Only 1 derivative traverses data in memory order (row-major)
  - Other 2 directions require data re-ordering for best efficiency

# 3D Complexity: FPGA Solution

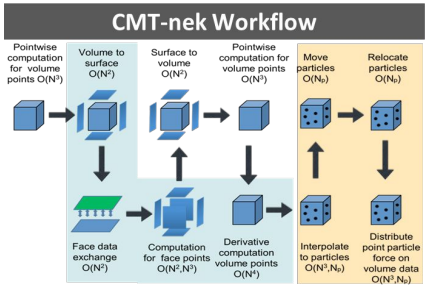


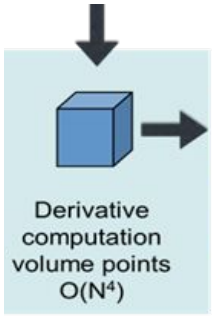



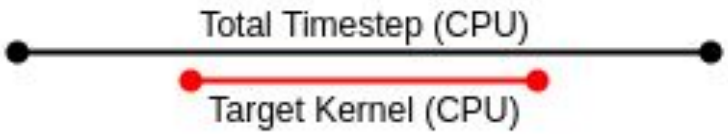

- Surround pipeline with data re-order buffers
  - Receive input ternix in row-major order; store in input buffer while also streaming through pipe for derivative *dt*
  - Stream input through pipe with step size of N for *ds*, then $N^2$ for *dr*
  - Collect results (*ds, dr*) back into row-major order in output buffer

# Experimental Methodology

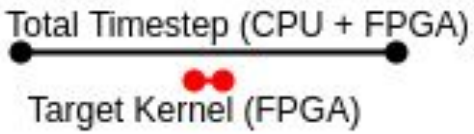
- What execution time is measured, and why?
  - Total timestep = work that each CPU core performs on its assigned portion of volume at every simulated instant
    - Each core has work, whether accelerated or not
    - Limits overall performance speedup by Amdahl's Law
  - Target kernel = portion of total timestep spent performing partial derivative computations
    - Measured on CPU core, GPU and FPGA to determine fine-grained parallelism
    - Speeding up compute bottleneck reduces total run time



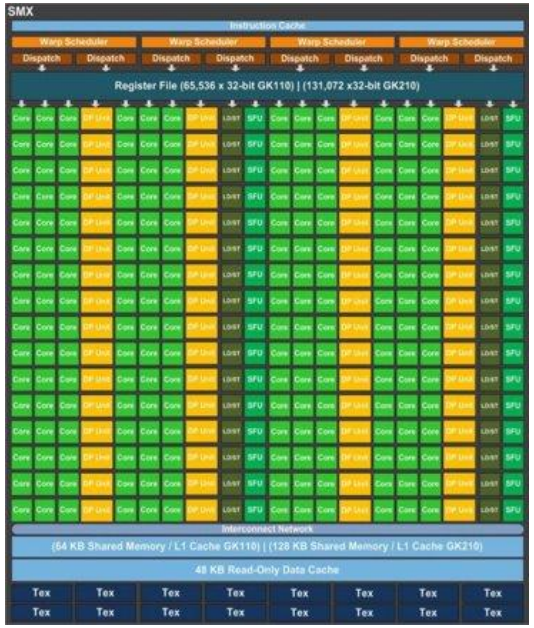





CCMT

# Test Environment

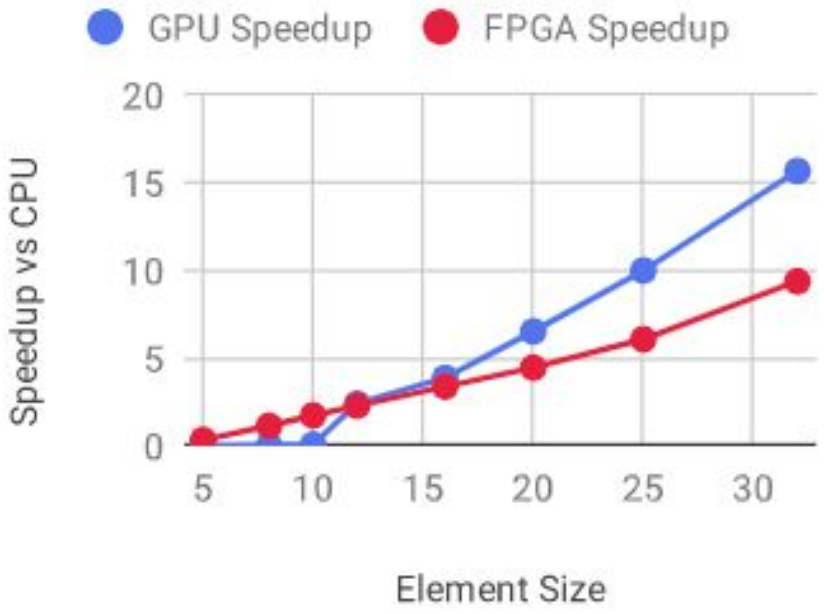
- Application: CMT-bone-BE
  - Representative mini-app of CMT-nek
- HiPerGator 2.0 @ UF
  - CPU: Intel Xeon E5
  - GPU: NVIDIA Tesla K80
  - Nvcc compiler with O2 optimization
- Intel DevCloud
  - FPGA: Intel Arria 10 GX 1150
  - Quartus Prime Pro 19.2.0
    - Synthesized Fmax 231-250 MHz
  - CPU: Intel Xeon E5*
    - *Substitute HiPerGator CPU times for consistency




# Target Kernel Performance Results

- Target kernel faster on FPGA than CPU core (all but 1 case)
  - GPU underperforms CPU core in 3 cases
  - GPU higher peak computational bandwidth FPGA
  - FPGA 28x faster than GPU for sizes 5-10

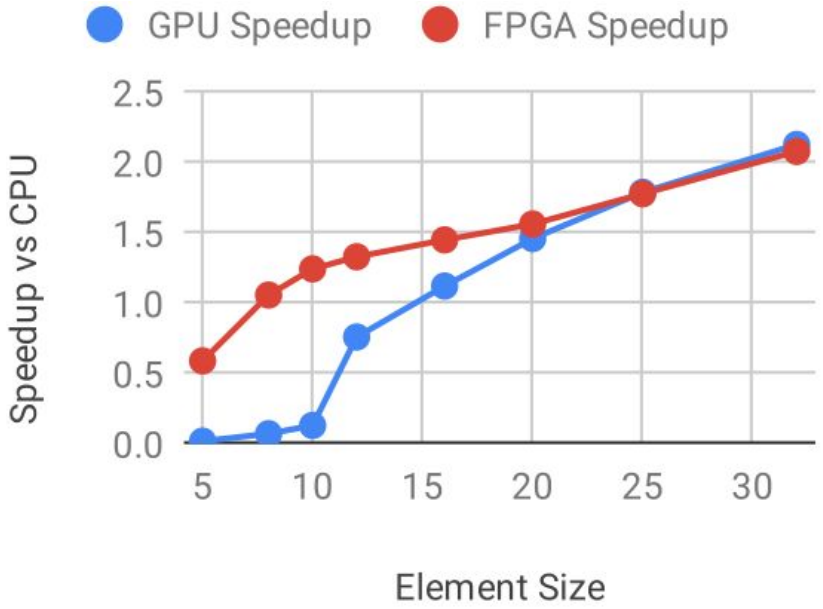
| Target Kernel Speedup vs CPU | | | | | |
|---|---|---|---|---|---|
| Element Size | CPU Time (ms) | GPU Time (ms) | GPU Speedup | FPGA Time (ms) | FPGA Speedup |
| 5 | 0.003 | 0.279 | 0.01 | 0.009 | 0.35 |
| 8 | 0.017 | 0.423 | 0.04 | 0.015 | 1.15 |
| 10 | 0.041 | 0.516 | 0.08 | 0.023 | 1.74 |
| 12 | 0.082 | 0.034 | 2.41 | 0.036 | 2.30 |
| 16 | 0.263 | 0.068 | 3.88 | 0.077 | 3.39 |
| 20 | 0.659 | 0.101 | 6.53 | 0.148 | 4.46 |
| 25 | 1.710 | 0.171 | 10.01 | 0.281 | 6.09 |
| 32 | 5.575 | 0.355 | 15.68 | 0.594 | 9.38 |



# Total Timestep Performance Results

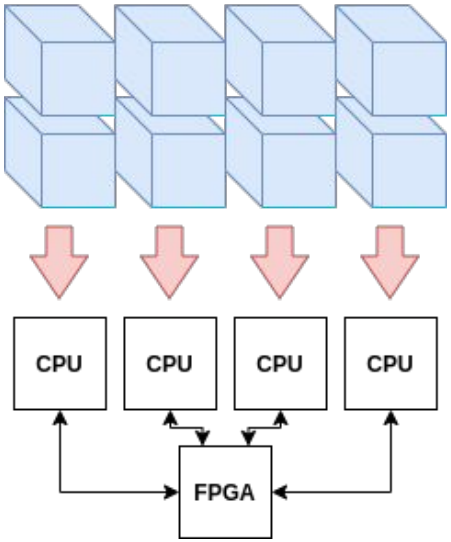
- Total timestep performance CPU with FPGA accelerated kernel vs CPU with GPU accelerated kernel
  - CPU+FPGA 18x faster than CPU+GPU for sizes 5-10
  - Competitive performance for larger element sizes
  - FPGA offers low-power performance vs GPU
    - Massive energy savings at small sizes (E=P*t)

| Total Timestep Speedup vs CPU | | | | | |
|---|---|---|---|---|---|
| Element Size | CPU Time (ms) | GPU Time (ms) | GPU Speedup | FPGA Time (ms) | FPGA Speedup |
| 5 | 0.008 | 0.379 | 0.02 | 0.013 | 0.59 |
| 8 | 0.039 | 0.531 | 0.07 | 0.037 | 1.06 |
| 10 | 0.085 | 0.642 | 0.13 | 0.068 | 1.25 |
| 12 | 0.178 | 0.234 | 0.76 | 0.134 | 1.33 |
| 16 | 0.571 | 0.508 | 1.12 | 0.393 | 1.45 |
| 20 | 1.360 | 0.931 | 1.46 | 0.869 | 1.56 |
| 25 | 3.130 | 1.748 | 1.79 | 1.758 | 1.78 |
| 32 | 9.203 | 4.318 | 2.13 | 4.421 | 2.08 |



# Projected Acceleration Scaling

- Accelerate multiple CPU cores with single FPGA
  - Replicate pipeline 8x comfortably (<1/15 resources each)
  - 8 parallel cores @ 1.5x average speedup = ~12 core equivalents
- Communication bandwidth is limiting factor
  - ~4 pipelines currently realistic
  - PCIe 5.0 could allow 12+ pipelines

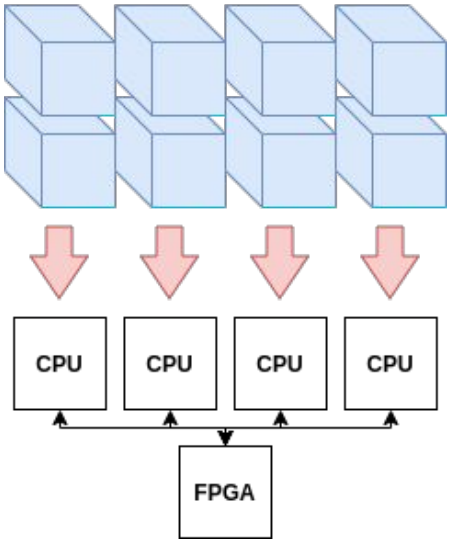


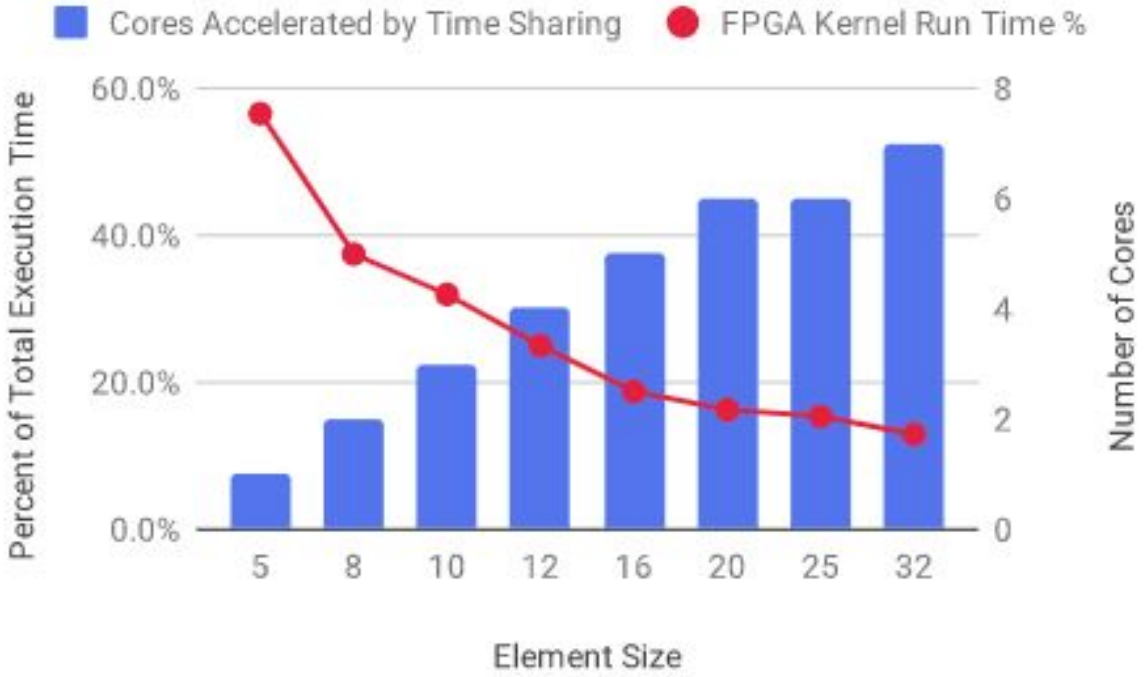

| | Arria 10 Resources | Pipeline Usage | Utilization |
|---|---|---|---|
| **ALM** | 427k | 27k | 6.3% |
| **Mem** | 67 Mb | 4 Mb | 6.0% |
| **DSP** | 1518 | 100 | 6.6% |

| # Pipelines | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| **Required Comm. B/W** | 4.0 GB/s | 8.0 GB/s | 16.0 GB/s | 24.0 GB/s | 32.0 GB/s |

# Projected FPGA Time Sharing

- FPGA pipeline only runs target kernel (else Idle)
  - CPU core only needs FPGA during target kernel
  - Multiple cores can share single pipeline in time
  - 5 cores sharing pipeline @ 1.5x average speedup = ~7.5 core equivalents from each shared pipeline
    - Multiple shared pipes increase Time & Area utilization of FPGA

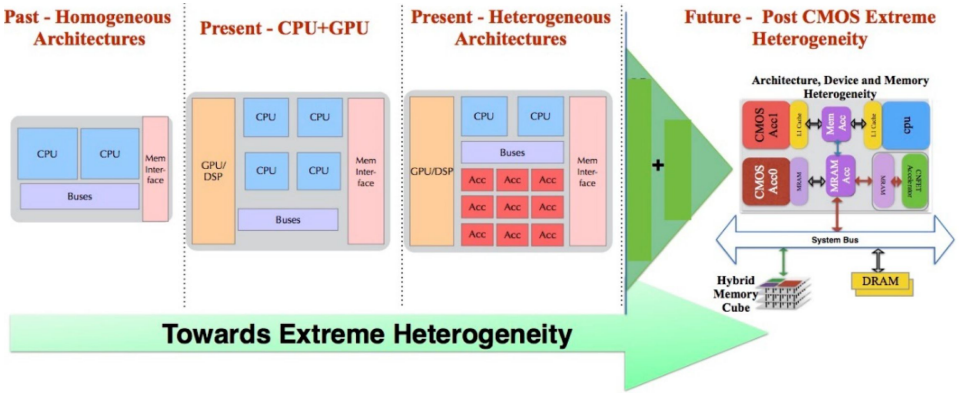


# Conclusions

- Fluid-flow workloads can be accelerated beyond coarse-grained CPU scaling
  - Over 9x speedup of targeted compute kernel on FPGA
  - Overall timestep speedup up to ~2x (Amdahl's Law)
  - More kernels could be targeted to improve overall speedup

- FPGAs show promise as hardware accelerators
  - Competitive performance vs GPU at high-end
  - More versatile performance across all sizes
  - More energy efficient than CPU, GPU



# Further Investigation

- Design Space Exploration of FPGA accelerator configurations
  - Accelerate multiple cores with replicated HW pipeline
  - Time-share each pipeline with multiple cores
- Collect power metrics of FPGA accelerator
  - Show energy-efficient performance vs CPU & GPU
- Accelerate more kernels for total speedup
  - Could consider productivity/performance tradeoffs of higher level hardware generation

CCMT

# QUESTIONS?

Ryan Blanchard

Greg Stitt, Herman Lam

Center for Compressible Multiphase Turbulence (CCMT)

ECE Department, University of Florida