



GPU Accelerated Numerical Methods For Tsunami Modeling

R Gandham, D S Medina, and T Warburton
Department of Computational and Applied Mathematics, Rice University

Context

Goal: To develop portable high performance computational tools and numerical models to predict tsunami impacts in a timely manner.

Challenge: Tsunami waves travel at very high speeds after the occurrence of earthquakes in the ocean bed and reach coastal areas within few hours.

Numerical method: Discontinuous Galerkin methods are high order accurate, flexible in handling unstructured coastal aligned meshes, and highly parallelizable on many core architectures making them attractive for simulating tsunami wave propagation.

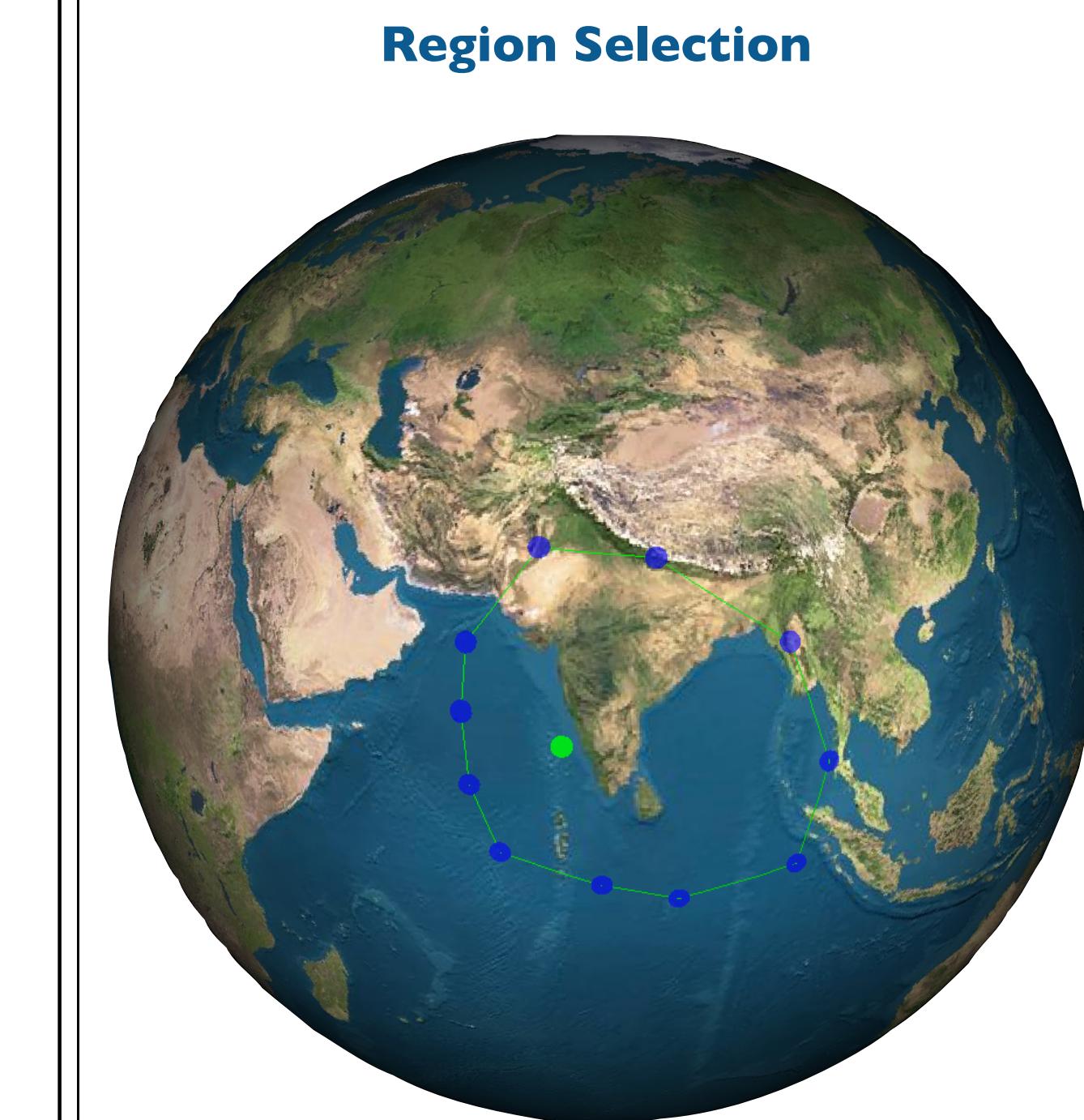
Programming model: OCCA unified threading model provides portability across threading models OpenCL, CUDA, and OpenMP.

Current Target Model : Shallow Water Equations

$$\begin{aligned} \frac{\partial h}{\partial t} + \frac{\partial(hu)}{\partial x} + \frac{\partial(hv)}{\partial y} &= 0 \\ \frac{\partial(hu)}{\partial t} + \frac{\partial}{\partial x}\left(hu^2 + \frac{1}{2}gh^2\right) + \frac{\partial}{\partial y}(huv) &= -gh\frac{\partial B}{\partial x} \\ \frac{\partial(hv)}{\partial t} + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}\left(hv^2 + \frac{1}{2}gh^2\right) &= -gh\frac{\partial B}{\partial y} \end{aligned}$$

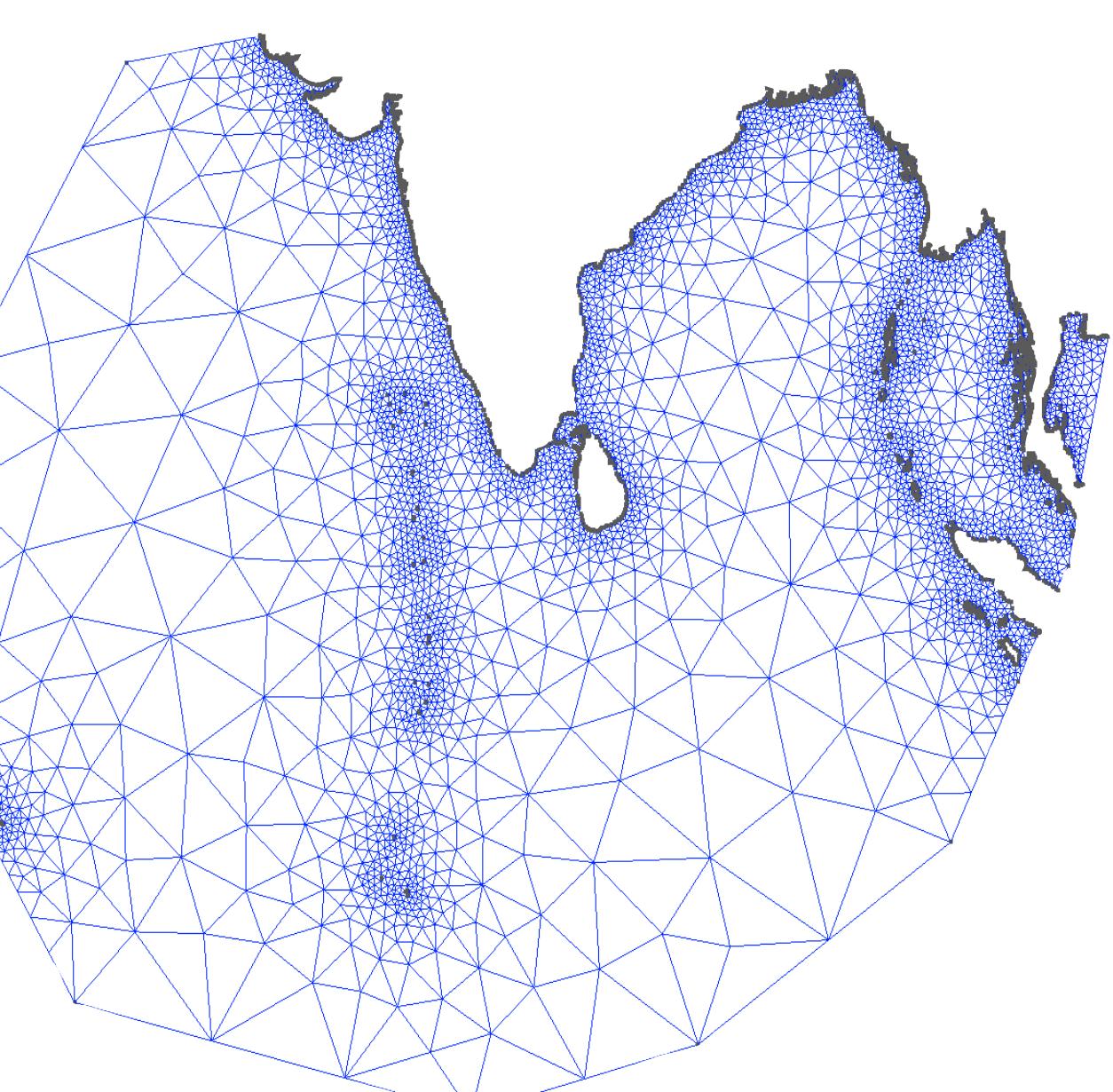
h: fluid height *u*: longitudinal velocity *v*: latitudinal velocity *B*: Bathymetry

1 Numerical Simulation

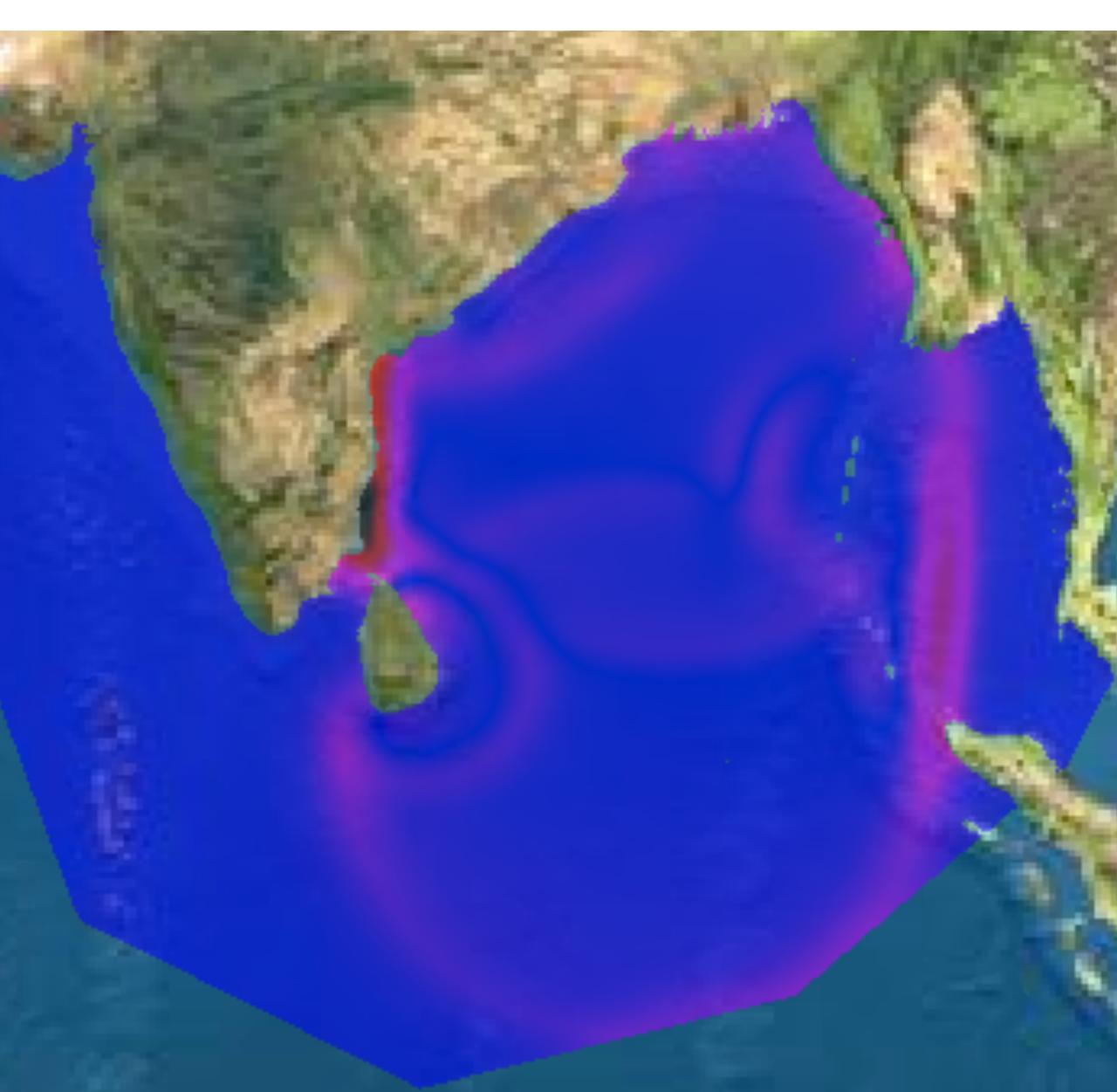


Region Selection

Coastal Aligned Triangulation



Simulation on GPUs



2 Discontinuous Galerkin

General Conservation Law

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F} = \mathbf{S}$$

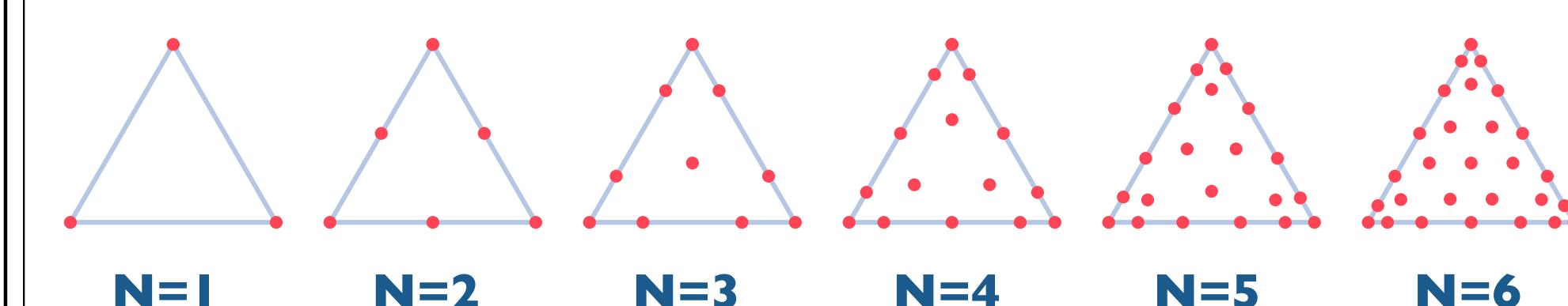
\mathbf{q} : field variables
 \mathbf{F} : nonlinear fluxes
 \mathbf{S} : source terms

DG Variational Statement

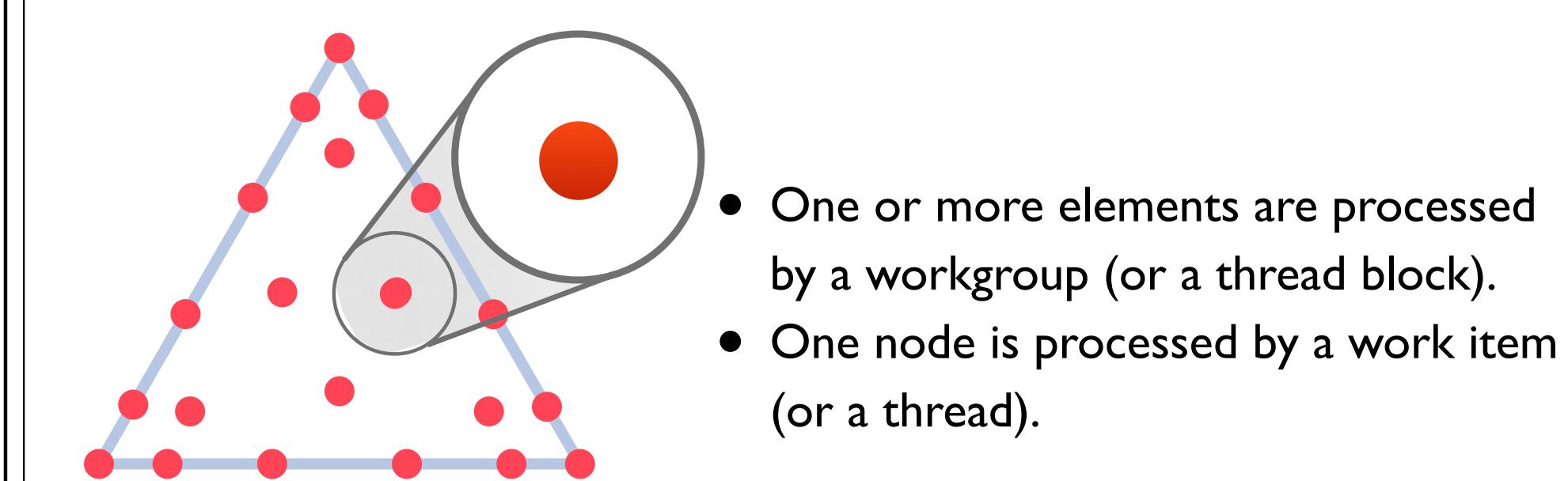
$$\left(\frac{\partial \mathbf{q}}{\partial t}, \phi \right)_{D^k} = (\mathbf{F}, \nabla \phi)_{D^k} + (\mathbf{S}, \phi)_{D^k} - ((\mathbf{F} \cdot \vec{n})^*, \phi)_{\partial D^k} \quad \forall \phi \in P^N(D^k)$$

Update kernel Volume kernel Surface kernel

Discretization : Warp & Blend Nodal Elements

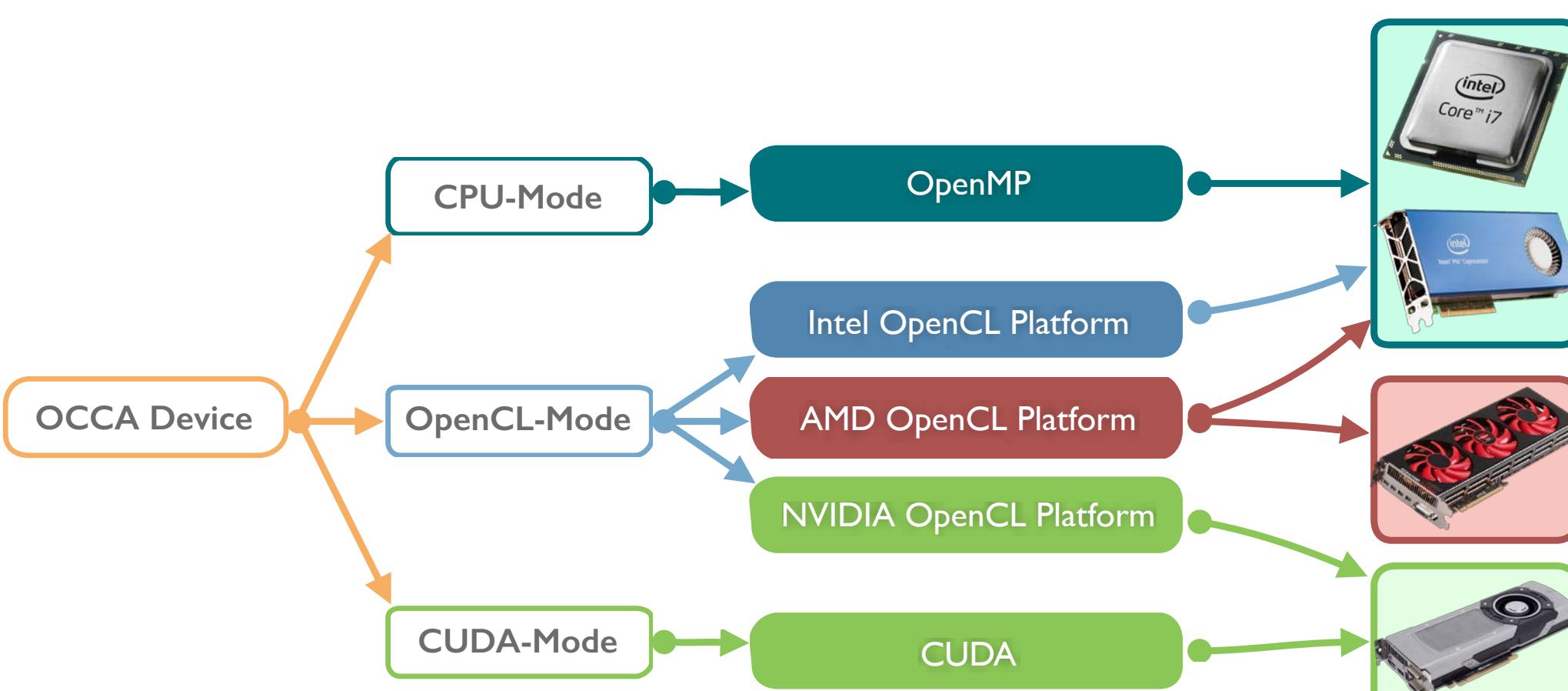


Fine-grain Parallelism [Kloeckner et al, 2009]



OCCA: Portable Multi-Threading Programming

4



OCCA Host API

- Abtracts different language APIs.
- Focuses on abstracting the device, memory and kernels.

OCCA Device API

- Relies on macros for masking the different supported languages.
- Uses the GPU programming model of work-groups and work-items.

OCCA Class

- Choose between using the CPU or available accelerators.
- OpenCL-mode: pick from supported platforms and devices.
- CUDA-mode: pick NVIDIA CUDA-enabled GPUs.

- Encapsulates function handles found in each language.
- Uses run-time compilation.

OCCA Memory

- Abstracts the memory handles found in each language.

```
// External variables are defined as compiler directives
occakernel void foo(occakernelInfo&rs, occaPointer double *a...){

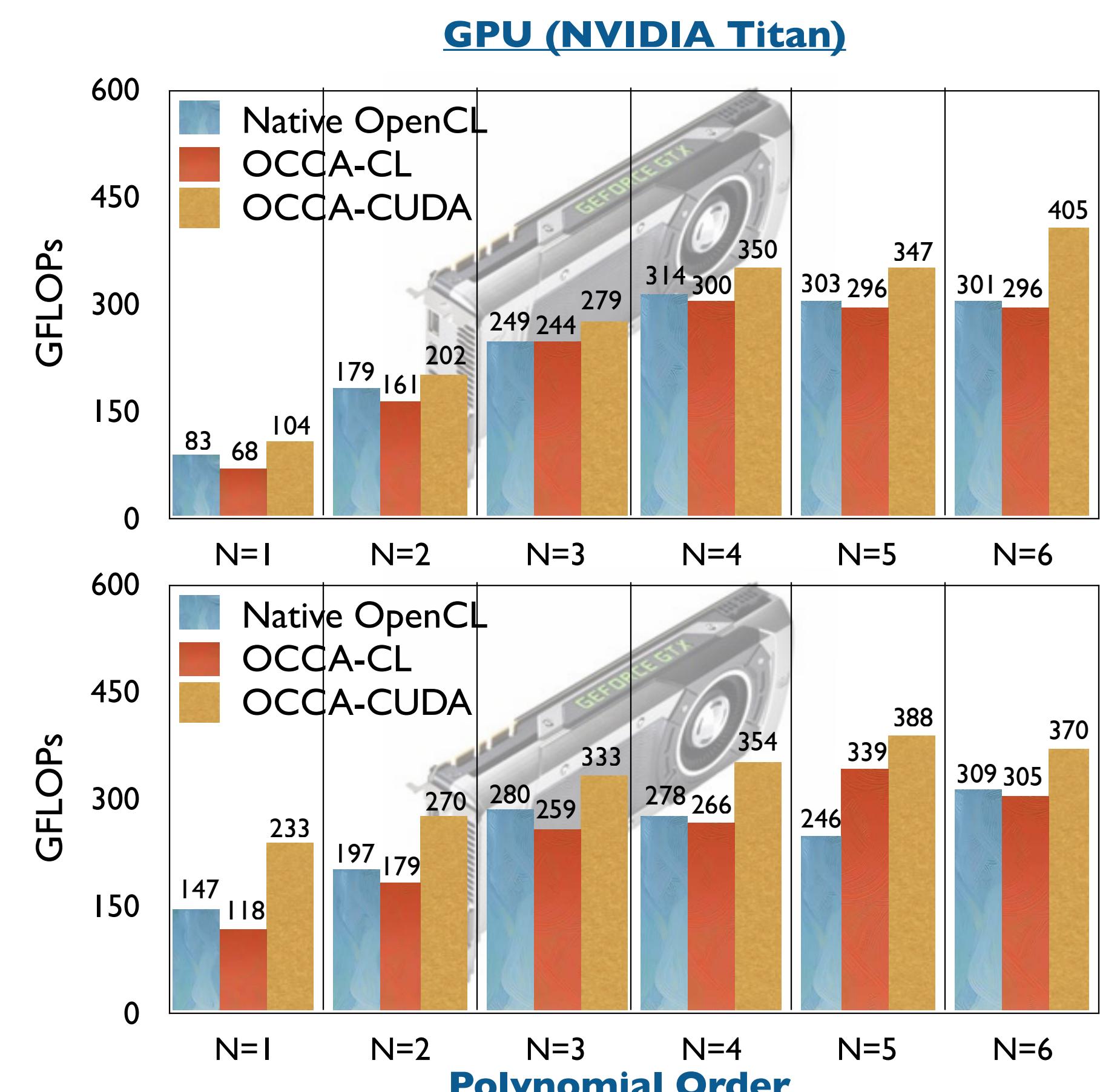
    occaOuterFor{ // work-group implicit loop
        occaInnerFor{ // work-item implicit loop
            occaInnerFor{ // GPU kernel scope
                ...
            }
        }
    }
}

// External variables are defined as compiler directives
extern "C" void foo(const int *Dims, double *a...){
    for(occaOuterId=0; occaOuterId<occOuterDim; ++occaOuterId2){
        for(occaOuterId1=0; occaOuterId1<occOuterDim; ++occaOuterId1){
            int occaInnerId=0;
            #pragma omp parallel for firstprivate(occaInnerId0, occaInnerId1, \
            occaOuterId2, occaDim0, occaDim1, occaDim2)
            for(occaInnerId2=0; occaInnerId2<occInnerDim; ++occaInnerId2){
                for(occaInnerId1=0; occaInnerId1<occInnerDim; ++occaInnerId1){
                    for(occaInnerId0=0; occaInnerId0<occInnerDim0; ++occaInnerId0)
                        ...
                }
            }
        }
    }
}
```

<https://github.com/tcew/OCCA>

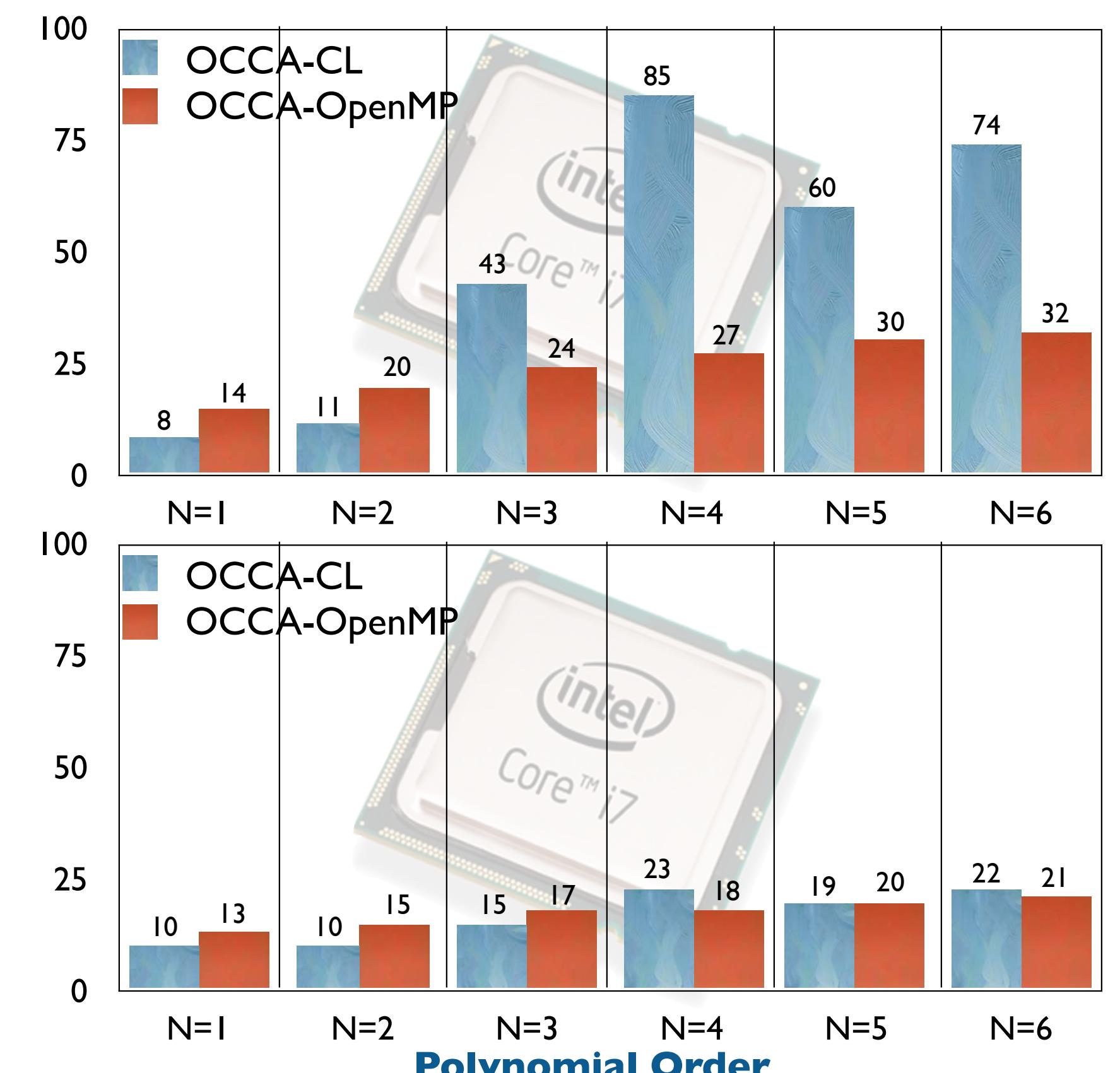
5 OCCA Based Simulation Performance

5



- OCCA kernels are tuned for each thread model separately.
- OCCA:OpenCL performance is similar to native OpenCL kernels.
- OCCA:CUDA runs faster compared to OCCA:OpenCL.

GPU (NVIDIA Titan)



Volume Kernel

Surface Kernel

- OCCA:OpenCL vectorizes when #work items is a multiple of 8 .
- OCCA:OpenCL kernels are tuned to benefit from vectorization.
- Surface kernel does not vectorize because of branching.