The OCCA abstract threading model

Implementation and performance for high-order finite-element computations

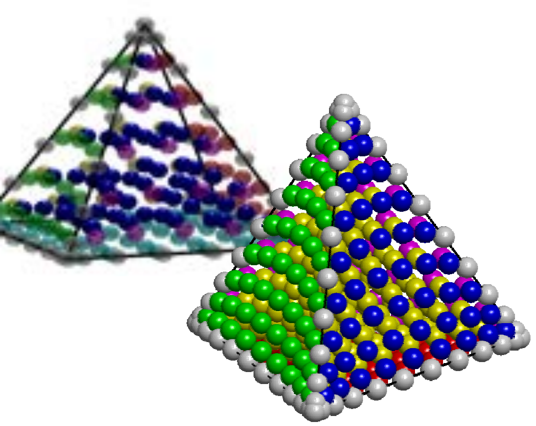
Tim Warburton David Medina
Axel Modave

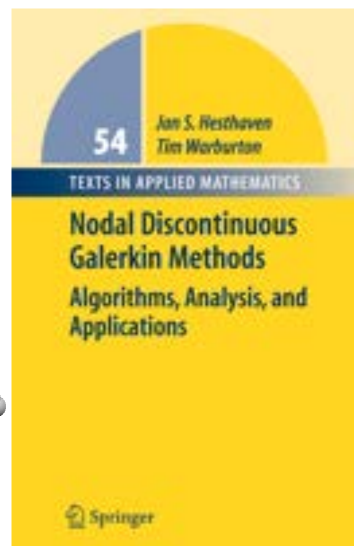
Virginia Tech, USA

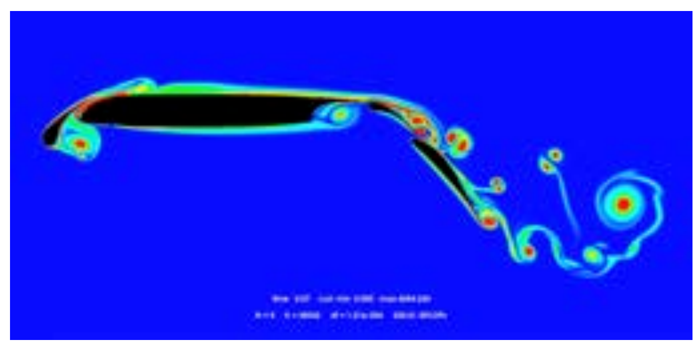
US Department of Energy - NSF - Air Force Office of Scientific Research - Office of Naval Research
Shell - Halliburton - TOTAL E&P - Argonne National Laboratory - MD Anderson Cancer Center - Intel - Nvidia - AMD

Towards efficient HPC applications for industrial simulations

Multidisciplinary research







Cluster NewRiver at Virginia Tech (*)

Approximation Theory	Numerical Analysis	Numerical & Physical PDE Modeling	Accelerated Computing	High Performance Scalability

Basic science → Application → Industrial Scale

(*) **Advanced Research Computing at Virginia Tech**
https://secure.hosting.vt.edu/www.arc.vt.edu/computing/newriver/

Towards efficient HPC applications for industrial simulations

Programming approach for HPC applications
with many-core devices

MPI + **X** = 😎

Which **X** for multi-threading ?

Challenges for efficient HPC applications

Portability

Code portability

- CUDA, OpenCL, OpenMP, OpenACC, Intel TBB… are not code compatible.
- Not all APIs are installed on any given system.

Performance portability

- Logically similar kernels differ in performance (GCC & ICPC, OpenCL & CUDA)
- Naively porting OpenMP to CUDA or OpenCL will likely yield low performance

Uncertainty

- Code life cycle measured in decades.
- Architecture & API life cycles measured in Moore doubling periods.
- Example: IBM Cell processor, IBM Blue Gene Q

Need an efficient, durable, portable, open-source,
vendor-independent approach for many-core programming

Portable programming framework - OCCA

Kernel Language (OKL) - API

Applications - Performance

Portable programming framework - OCCA

Kernel Language (OKL) - API

Applications - Performance

Portable approaches for many-core programming (1/3)

Directive approach

- Use of optional [#pragma]'s to give compiler transformation hints
- Aims for portability, performance and programmability
- OpenACC and OpenMP begin to resemble an API rather than code decorations

OpenACC

- Introduced for accelerator support through directives (2012).
- There are compilers which support the 1.0 specifications.
- OpenACC 2.0 introduces support for inlined functions.

OpenMP

- OpenMP has been around for a while (1997).
- OpenMP 4.0 specifications (2013) includes accelerator support.
- Few compilers (ROSE) support parts of the 4.0 specifications.

OpenMP

```
#pragma omp target teams distribute parallel for
for(int i = 0; i < N; ++i){
  y[i] = a*x[i] + y[i];
}
```

Code taken from:
WHAT'S NEW IN OPENACC 2.0 AND OPENMP 4.0, GTC '14

Portable approaches for many-core programming (2/3)

Wrapper approach

- Create a tailored library with optimized functions
- Restricted to a set of operations with flexibility from functors/lambdas

Thrust

- C++ library masking OpenMP, Intel's TBB and CUDA for x86 processors and NVIDIA GPUs
- Vector library, such as the standard template library (STL)

Kokkos

- Kokkos is from Sandia National Laboratories
- C++ vector library with linear algebra routines
- Uses OpenMP and CUDA for x86 and NVIDIA GPU support

SkePU

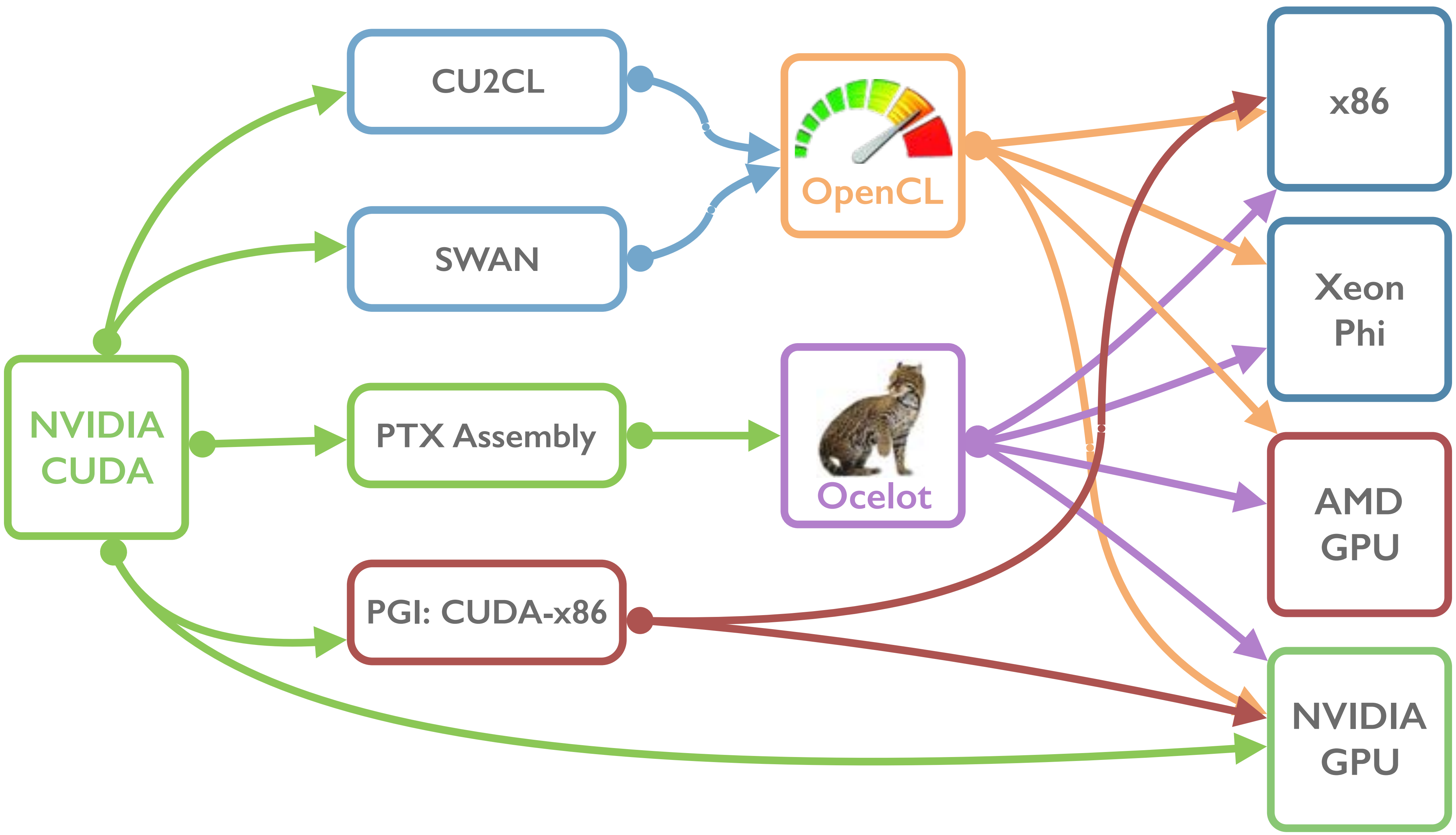
- C++ template library
- Uses code skeletons for map, reduce, scan, mapreduce, …
- Uses OpenMP, OpenCL and CUDA as backends

All C++ libraries with tailored functionalities
HPC has a large C and Fortran community!

Portable approaches for many-core programming (3/3)

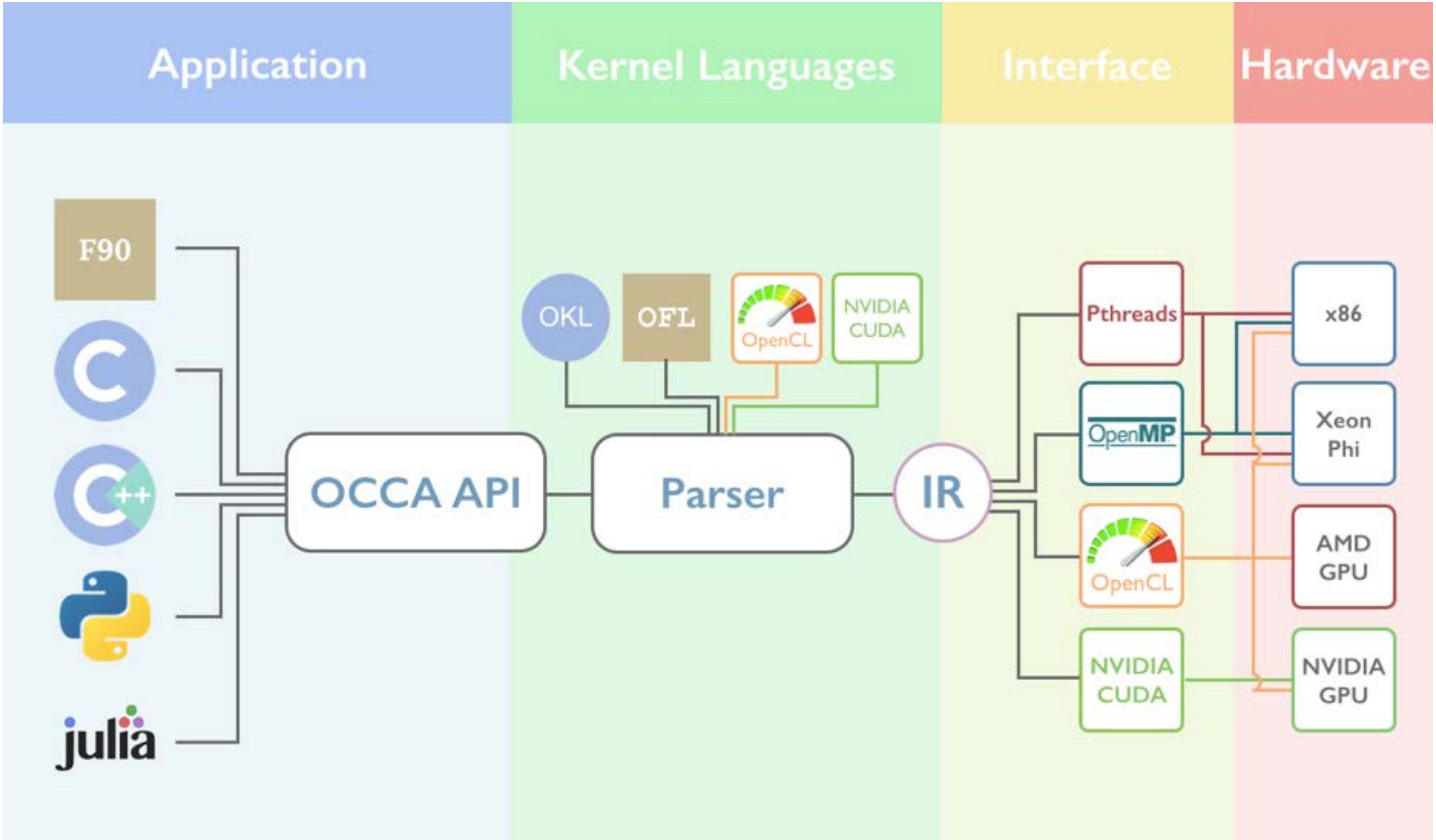
Source-to-source approach

- CU2CL & SWAN have limited CUDA support (3.2 and 2.0 respectively) Update ?
- GPU Ocelot supports PTX from CUDA 4.2 (5.0 partially)
- PGI: CUDA-x86 appears to have been put in hiatus since 2011



Open **C**oncurrent **C**ompute **A**rchitecture — **OCCA**

Portability Accessibility Lightweight



libocca.org github.com/libocca/occa (MIT license)

What does OCCA **not** do?

Open Concurrent Compute Architecture

Auto-parallelize:

- Some programmer intervention is required to identify parallel for loops.

Auto-optimize:

- Programmer knowledge of architecture is still invaluable.

Auto-layout:

- The programmer needs to decide how data is arranged in memory.

Auto-distribute:

- You can use MPI+OCCA but you have to write the MPI code.
- We considered M-OCCA but it devolves quickly into a PGAS.

Low-level code:

- We do not circumvent the vendor compilers.

Portable programming framework - OCCA

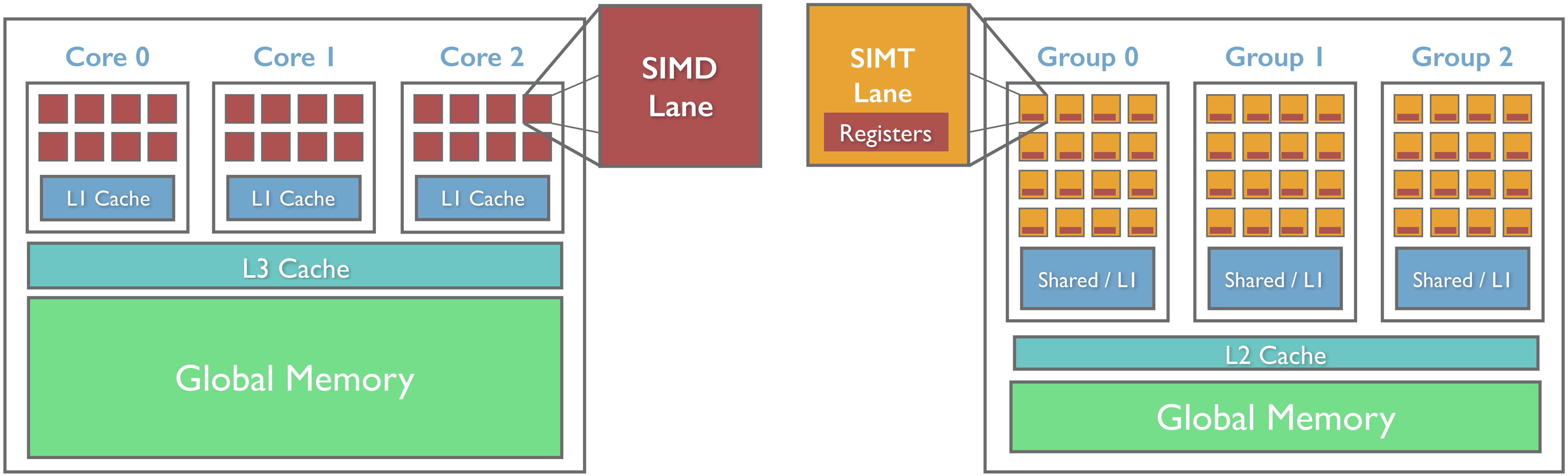
Kernel Language (OKL) - API

Applications - Performance

Parallelization Paradigm

CPU Architecture

GPU Architecture




```
void cpuFunction(){
  #pragma omp parallel for
  for(int i = 0; i < work; ++i){

    Do [hopefully thread-independent] work

  }

}
```

```
__kernel void gpuFunction(){
// for each work-group {
//   for each work-item in group {

  Do [group-independent] work

//   }
// }
}
```

Computational hierarchies are similar

Kernel Language OKL

Description

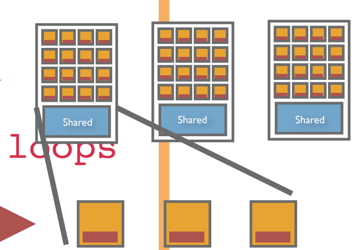
- Minimal extensions to C, familiar for regular programmers
- Explicit loops expose parallelism for modern multicore CPUs and accelerators
- Parallel loops are explicit through the fourth for-loop inner and outer labels

```
kernel void kernelName(...){
  ...

  for(int groupZ = 0; groupZ < zGroups; ++groupZ; outer2){
    for(int groupY = 0; groupY < yGroups; ++groupY; outer1){
      for(int groupX = 0; groupX < xGroups; ++groupX; outer0){     // Work-group implicit loops

        for(int itemZ = 0; itemZ < zItems; ++itemZ; inner2){
          for(int itemY = 0; itemY < yItems; ++itemY; inner1){
            for(int itemX = 0; itemX < xItems; ++itemX; inner0){  // Work-item implicit loops
            // GPU Kernel Scope
        }}}
  }}}

  ...
}
```



OKL

NVIDIA CUDA

```
dim3 blockDim(xGroups,yGroups,zGroups);
dim3 threadDim(xItems,yItems,zItems);
kernelName<<< blockDim , threadDim >>>(…);
```

The concept of iterating over groups and items is simple

Kernel Language OKL

Outer-loops

- Outer-loops are synonymous with CUDA and OpenCL kernels
- Extension: allow for multiple outer-loops per kernel

```
kernel void kernelName(…){

  if(expr){
    for(outer){
      for(inner){
      }
    }
  else{
    for(outer){
      for(inner){
      }
    }
  }

  while(expr){
    for(outer){
      for(inner){
      }
    }
  }

}
```

OKL

Data dependencies are found through a variable dependency graph

Kernel Language OKL

Shared memory

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  shared int sharedVar[16];

    for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
      sharedVar[itemX] = itemX;
    }

    // Auto-insert [barrier(localMemFence);]

    for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
      int i = (sharedVar[itemX] + sharedVar[(itemX + 1) % 16]);
    }
}
```

OKL

Exclusive memory

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  exclusive int exclusiveVar, exclusiveArray[10];

    for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
      exclusiveVar = itemX;  // Pre-fetch
    }

    // Auto-insert [barrier(localMemFence);]

    for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
      int i = exclusiveVar;  // Use pre-fetched data
    }
}
```

OKL

Local barriers are auto-inserted

Example of code (Adding two vectors)

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  device.setup(“mode = OpenCL, platformID = 0, deviceID = 0”);

  float *a  = new float[5];
  float *b  = new float[5];
  float *ab = new float[5];

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource(“addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

https://github.com/tcew/OCCA2/tree/master/examples/addVectors

Example of code (Adding two vectors)

```
#include "occa.hpp"

int ma...

  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

```
kernel void addVectors(const int entries,
                       const float *a,
                       const float *b,
                       float *ab){

  for(int group = 0; group < ((entries + 15)/16); ++group; outer0){
    for(int item = 0; item < 16; ++item; inner0){
      const int i = item + (16 * group);

      if(i < entries)
        ab[i] = a[i] + b[i];
    }
  }
}
```

OKL

https://github.com/tcew/OCCA2/tree/master/examples/addVectors

Example of feature (UVA + Managed memory)

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  float *a  = new float[5];
  float *b  = new float[5];
  float *ab = new float[5];

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

https://github.com/tcew/OCCA2/tree/master/examples/addVectors

Example of feature (UVA + Managed memory)

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  float *a  = (float*) device.managedUvaAlloc(5 * sizeof(float));
  float *b  = (float*) device.managedUvaAlloc(5 * sizeof(float));
  float *ab = (float*) device.managedUvaAlloc(5 * sizeof(float));

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, a, b, ab);

  occa::finish() // o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

https://github.com/tcew/OCCA2/tree/master/examples/uvaAddVectors

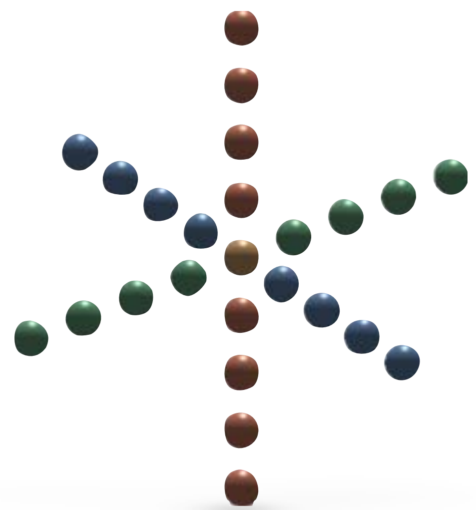
Portable programming framework - OCCA

Kernel Language (OKL) - API

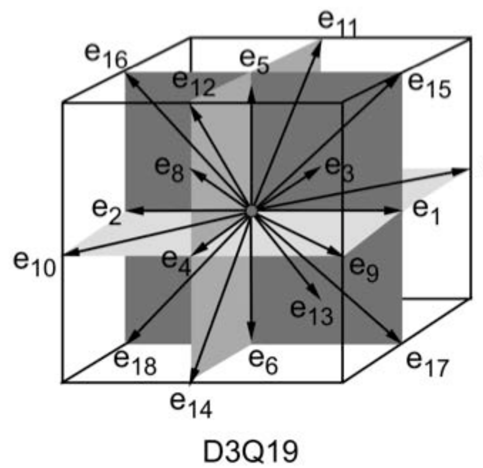
Applications - Performance

OCCA applications/benchmarks

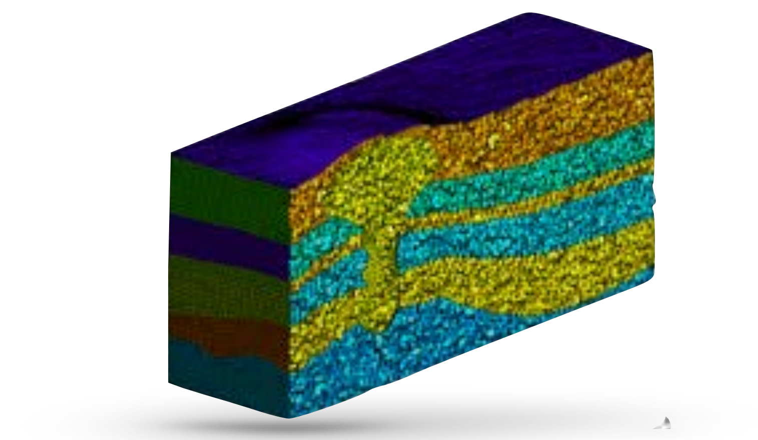
OCCA apps can perform close to or exceed native apps across platforms.



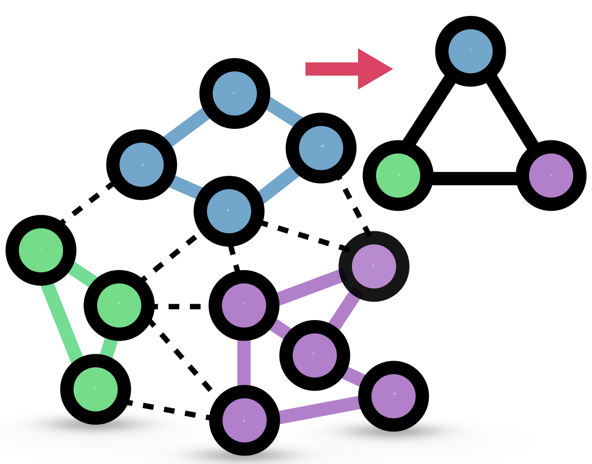
FDTD for seismic wave equation



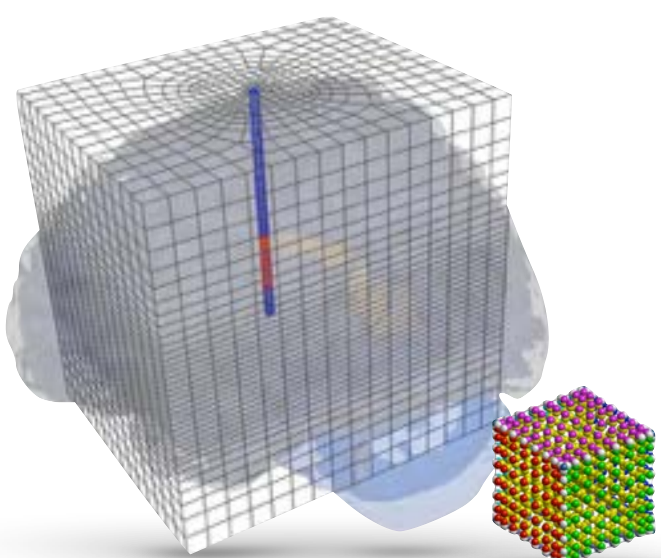
Lattice Boltzmann for Core Sample Analysis



RiDG: DG finite element for seismic imaging



ALMOND: algebraic multigrid library

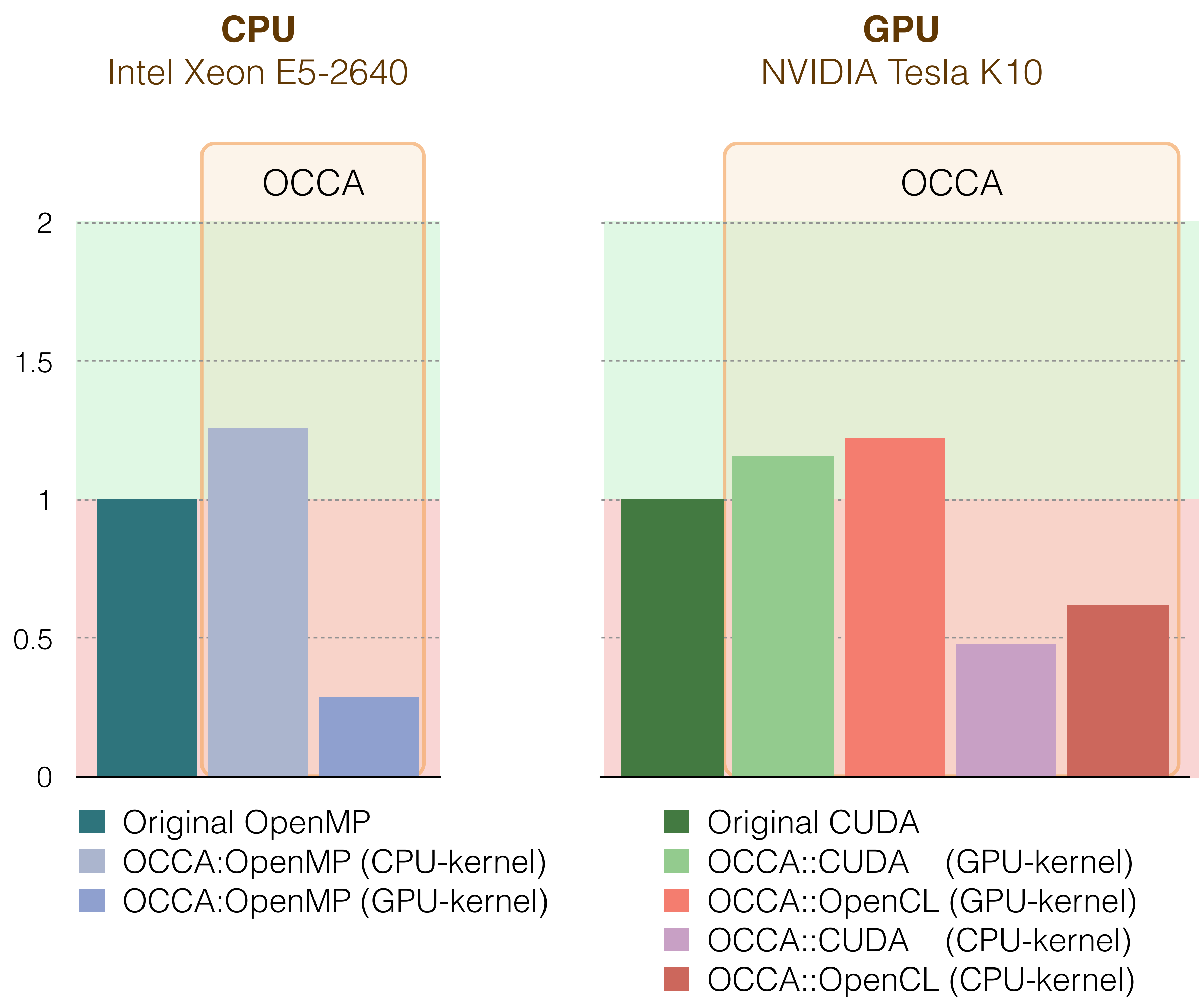


MDACC: FEM model of laser tumor ablation

DG compressible Navier-Stokes solver

DG shallow-water & 3D ocean modeling

Results - Finite difference for seismic waves

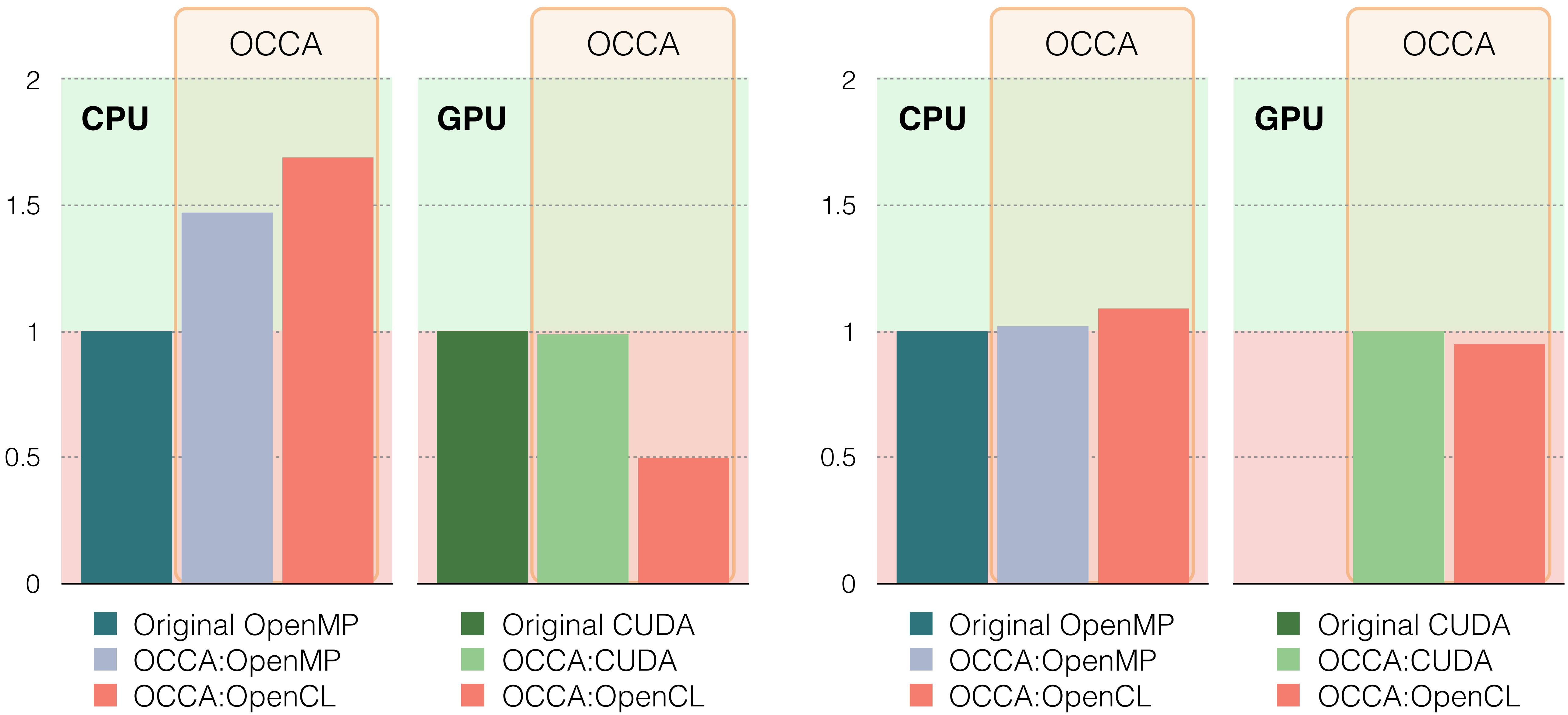


Medina, St-Cyr & Warburton (2015)
Proceedings of ICOSAHOM 2014. 365-373

Results - Monte Carlo for neutronics

Collaborations with Argonne National Lab

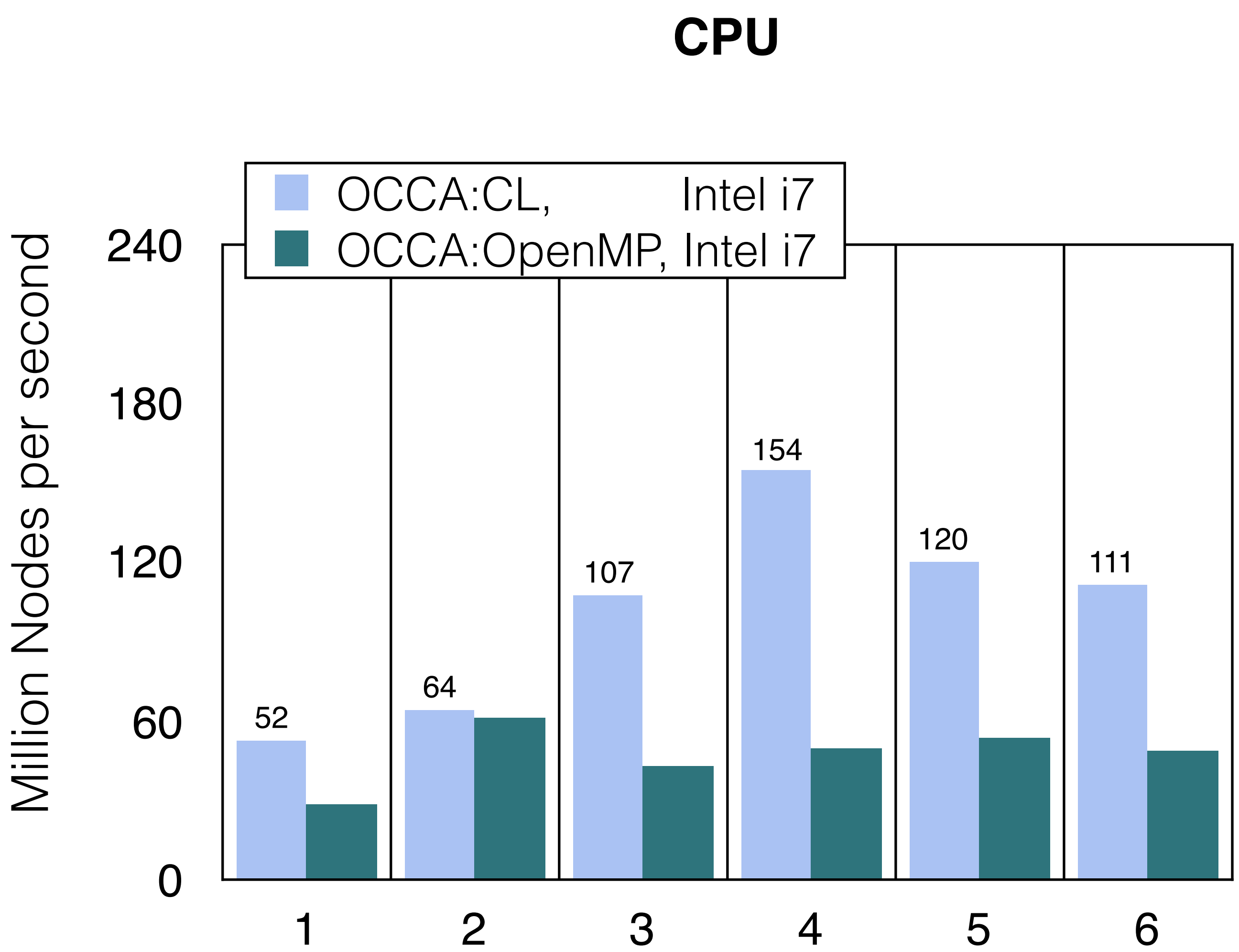
XSBench RSBench

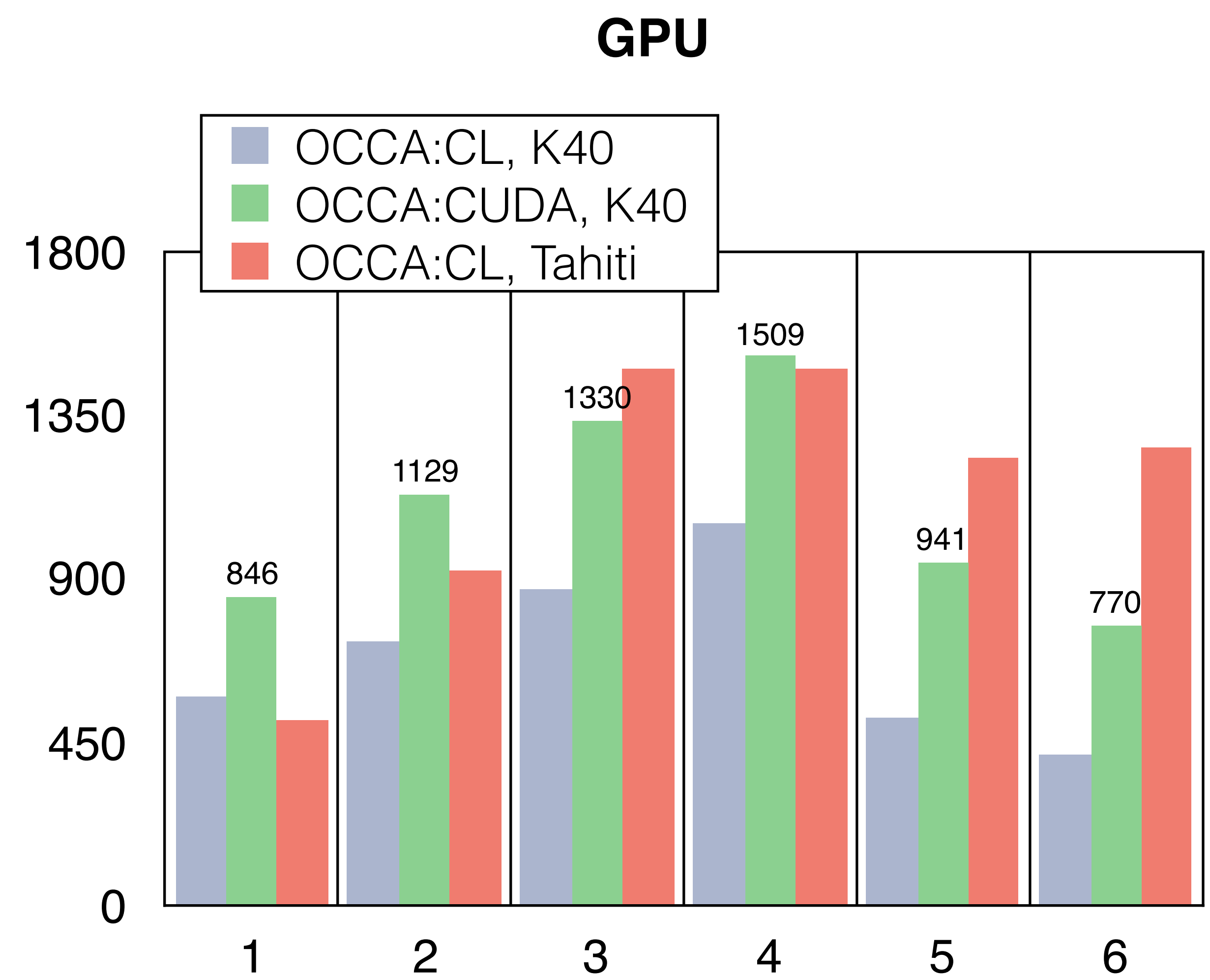


Rahaman, Medina, Lund, Tramm, Warburton, Siegel (2015)
Conference on Exascale Applications and Software

OpenMP : Intel Xeon CPU E5-2650
OpenCL/CUDA : NVIDIA Tesla K20c

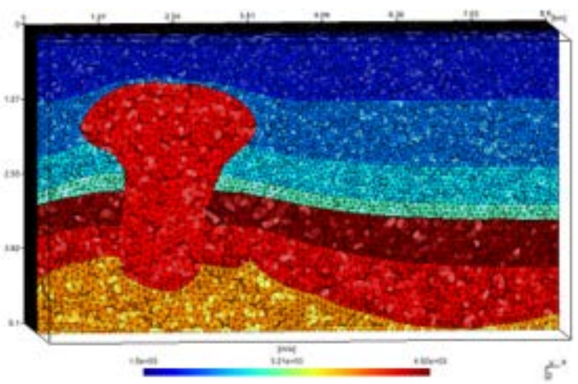
Results - DG finite element for oceanographic waves



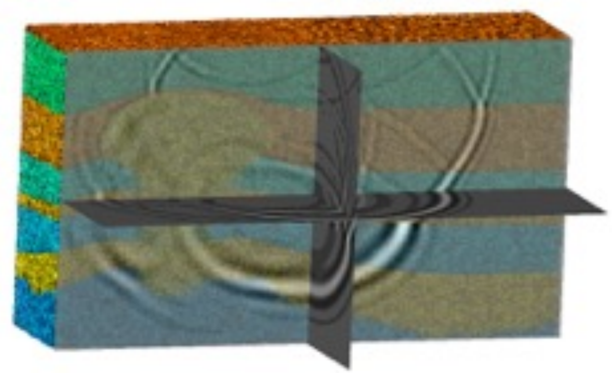


Gandham, Medina & Warburton (2015)
Communications in Computational Physics 18:37-64

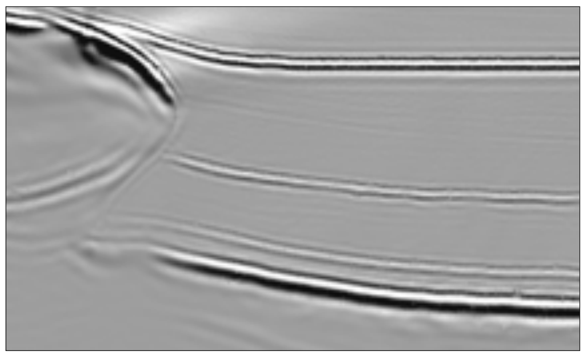
Results - DG finite element for seismic imaging



Mesh of underground

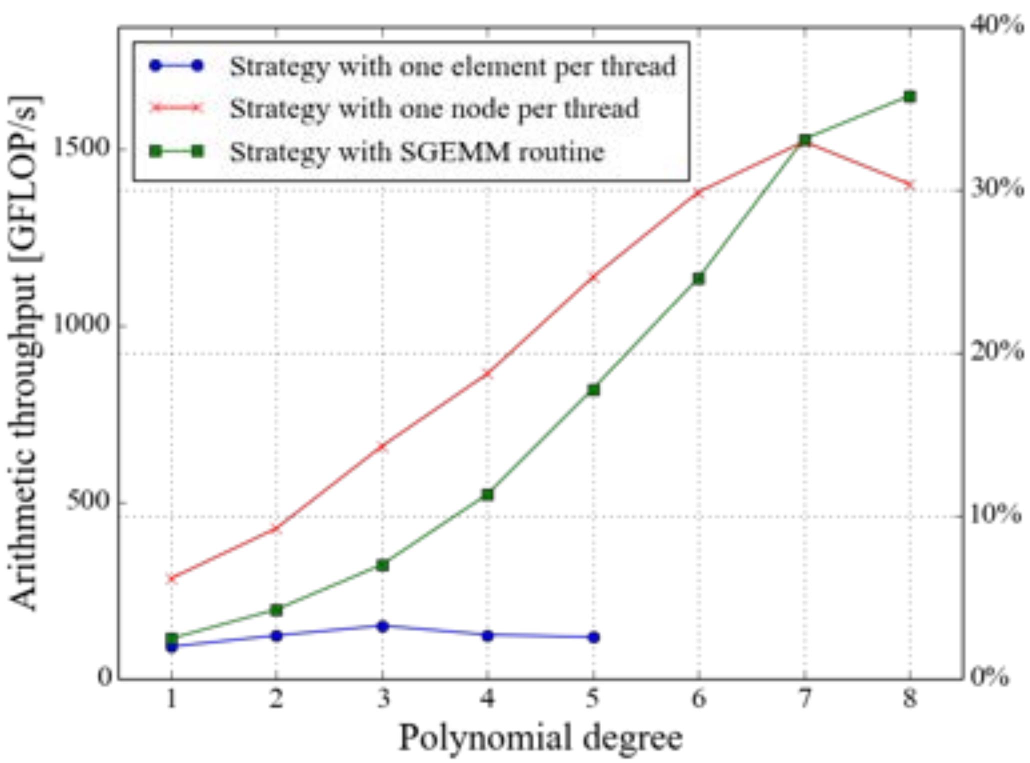


A lot of expensive simulations !!!



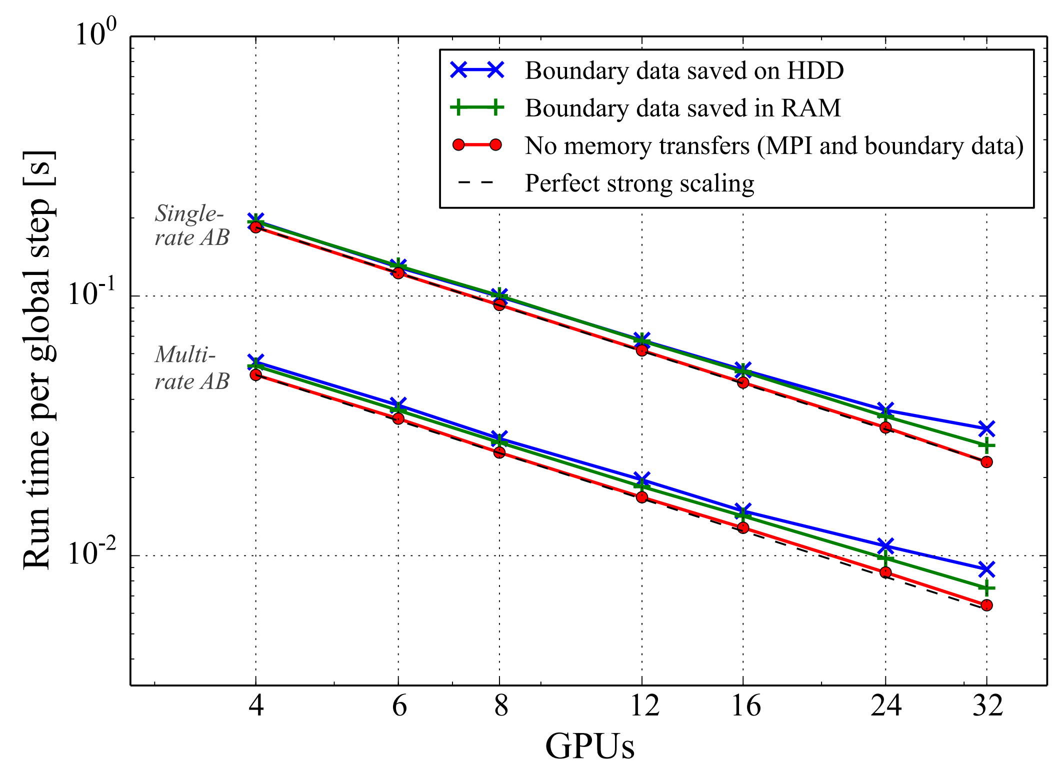
Seismic image

Can reach **37%** of **theoretical peak GFLOP/s**
OCCA::CUDA (Nvidia GTX980)



Modave, St-Cyr & Warburton (2016)
Computers & Geosciences 18:37-64

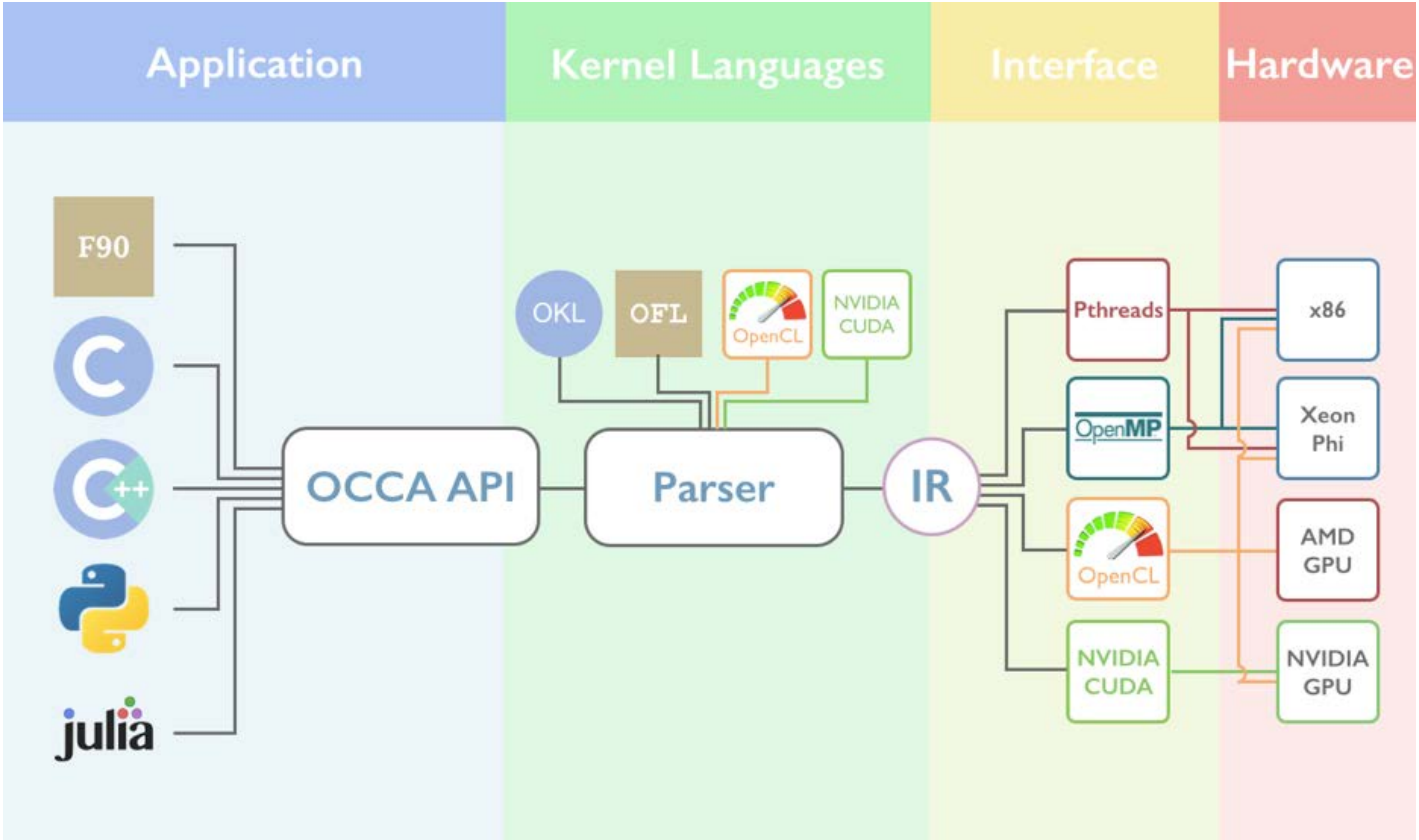
Strong scalability with **32 GPUs**
MPI + OCCA::CUDA (Nvidia K20)



Modave, St-Cyr, Mulder & Warburton (2015)
Geophysical Journal International 203(2):1419-1435

Open **C**oncurrent **C**ompute **A**rchitecture — **OCCA**

Portability Accessibility Lightweight



libocca.org github.com/libocca/occa (MIT license)