RICE

Unconventional Wisdom

OCCA: Portability Layer for Many-core Thread Programming

Rice Oil & Gas Workshop 2015
March 3, 2015.

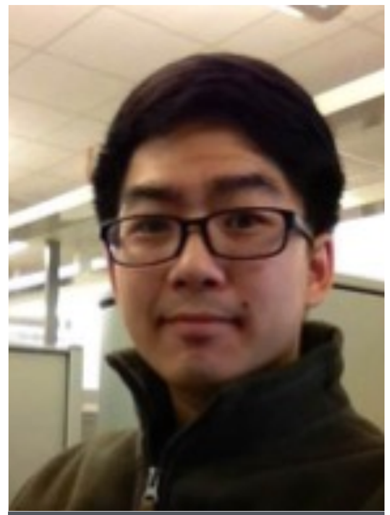
Tim Warburton
Department of Computational & Applied Mathematics @ Rice University

Extended Research Team

I am fortunate to work with a team of excellent researchers


JF Remacle
Meshing & Numerics
Sabbatical @ Rice


Amik St-Cyr
Royal Dutch Shell
Adjunct Assoc Prof


Jesse Chan
Advanced Numerics
Post-doc


Axel Modave
Accelerate Numerics
Industrial Post-doc


Florian Kummer
Multi-phase Flows
Visiting scholar

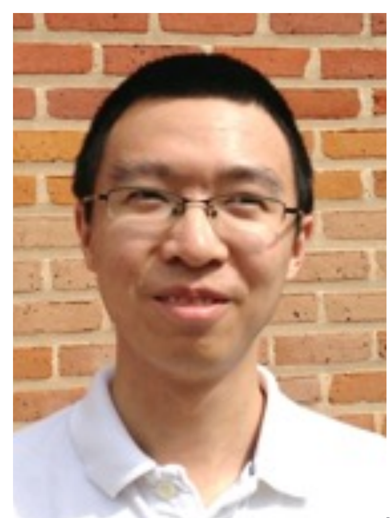

Ali Karakus
Two Phase Flows
Visiting student

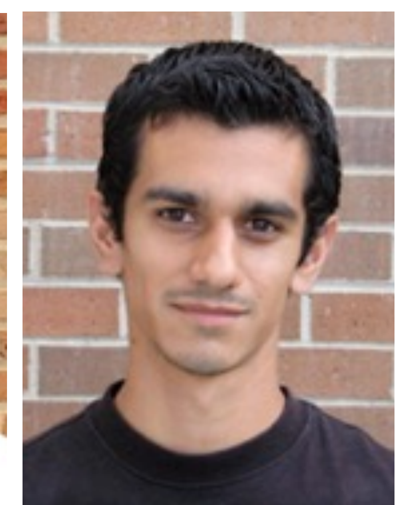

Nichole Stilwell
CFD
CAAM Grad > USAF

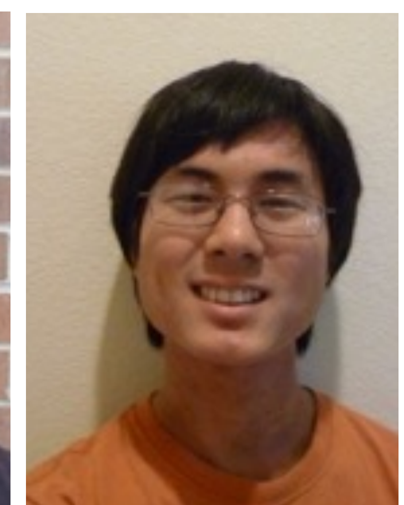

Rajesh Gandham
Oceans & AMG
CAAM Grad

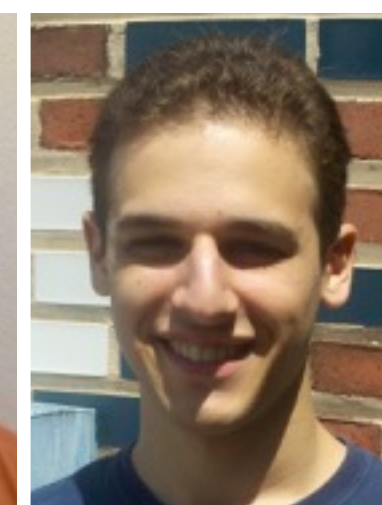

David Medina
HPC & Accelerators
CAAM Grad


Reid Atcheson
Adaptivity/Helmholtz
CAAM Grad


Zheng (Frank) Wang
Radiation Transport
CAAM Grad


Arturo Vargas
Hermite+DG
CAAM Grad


Tim Moon
Bioheat modeling
Ugrad => Stanford


Michael Frano
Bioheat modeling
Ugrad => BP

Industrial internships, projects & fellowships:
Shell, BP, Halliburton, Hess, Stoneridge Technology, Hypercomp, Z-terra, ExxonMobil*

Overview

Interlude on Many-core Computing:

- Challenges of modern computing: uncertainty, fragmentation, parallel algorithms.
- OCCA: unified programming model for multi- and many-core computing.

OCCA:

- Introduce API and kernel languages, OKL and OFL
- Mention applications and benchmarks with performance comparisons

Live Demo

- Download and install OCCA from scratch
- Display features in the programming interface and kernel languages

Interactive Demo

- We let you (the attendees) get hands-on experience using OCCA
- Members in our research group will walk around to help if needed

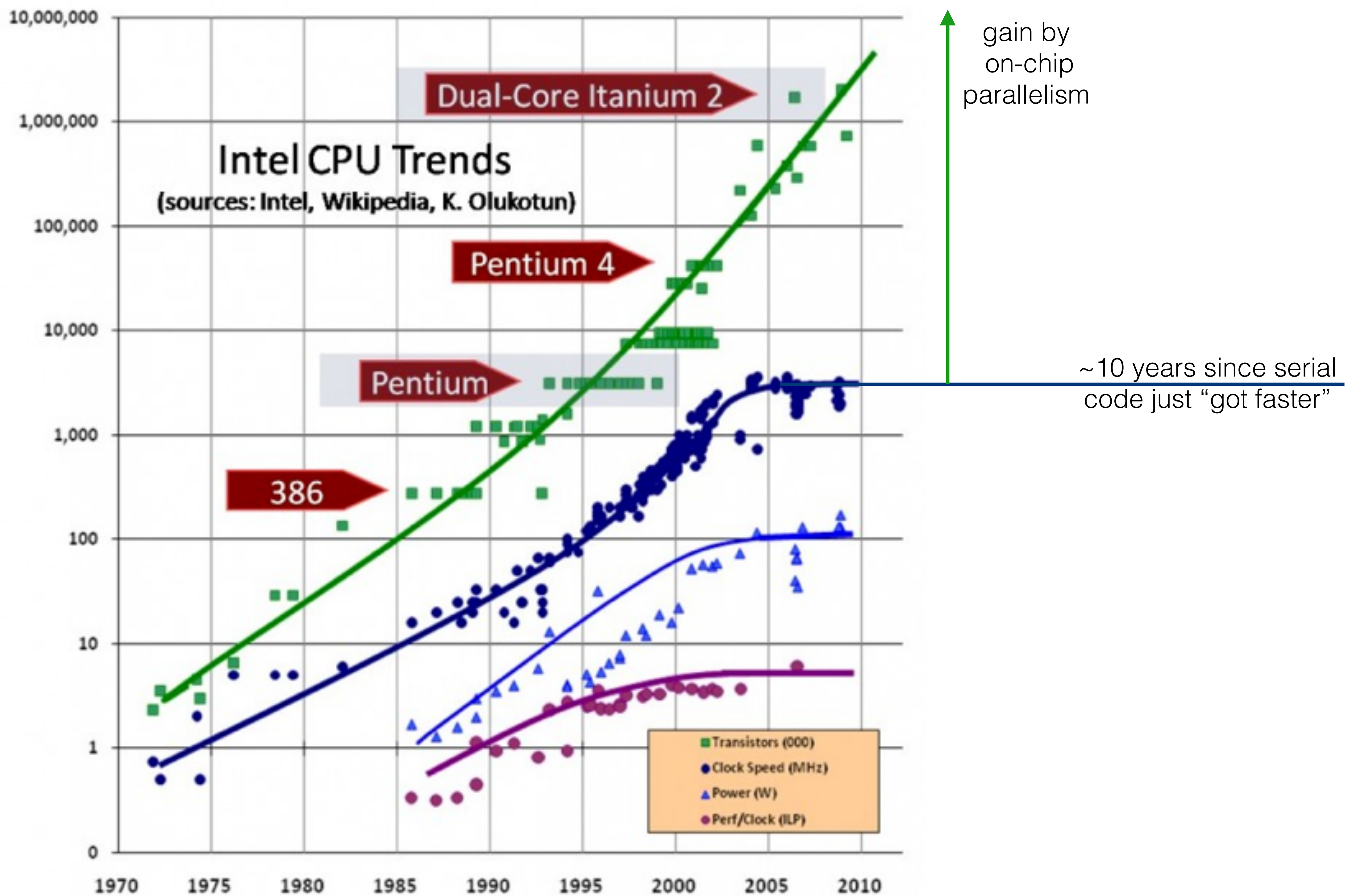
OCCA: portable many-core library

Project with David Medina (Rice) & contributors.

Funded by the Argonne CESAR Exascale Co-design center

Moore’s Law: transition to many-core

There is no escaping parallel computing any more even on a laptop.



http://wccftech.com/moores-law-will-be-dead-2020-claim-experts-fab-process-cpug-pus-7-nm-5-nm/

Ubiquitous Multicore: no escape

Multicore and heterogeneous compute is prevalent from
Intel's Broadwell-U mobile processor to the upcoming Intel Knights Landing accelerator.

Broadwell die shot
(removed)

Broadwell-U: 14nm process with dual cores
& integrated GPU [1.9B transistors]

Knights landing die shot
(removed)

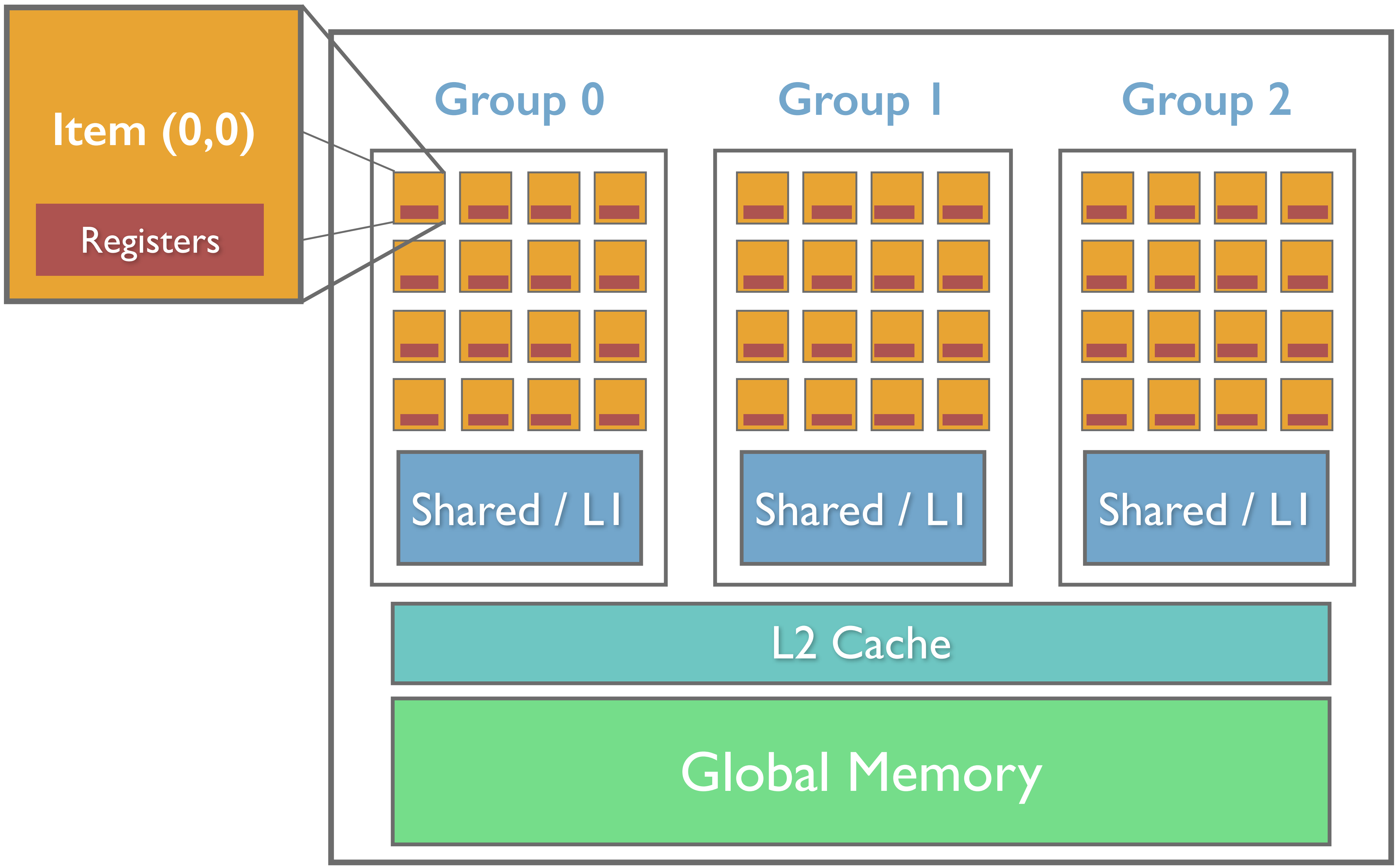
Knights Landing: 72 Silvermont Atom cores
with 512 bit vector registers. Stacked ram.

http://www.pcworld.com/article/2863885/intel-brings-next-gen-broadwell-processor-tech-to-mainstream-notebooks-desktops.html

GPU Architecture (abstract)

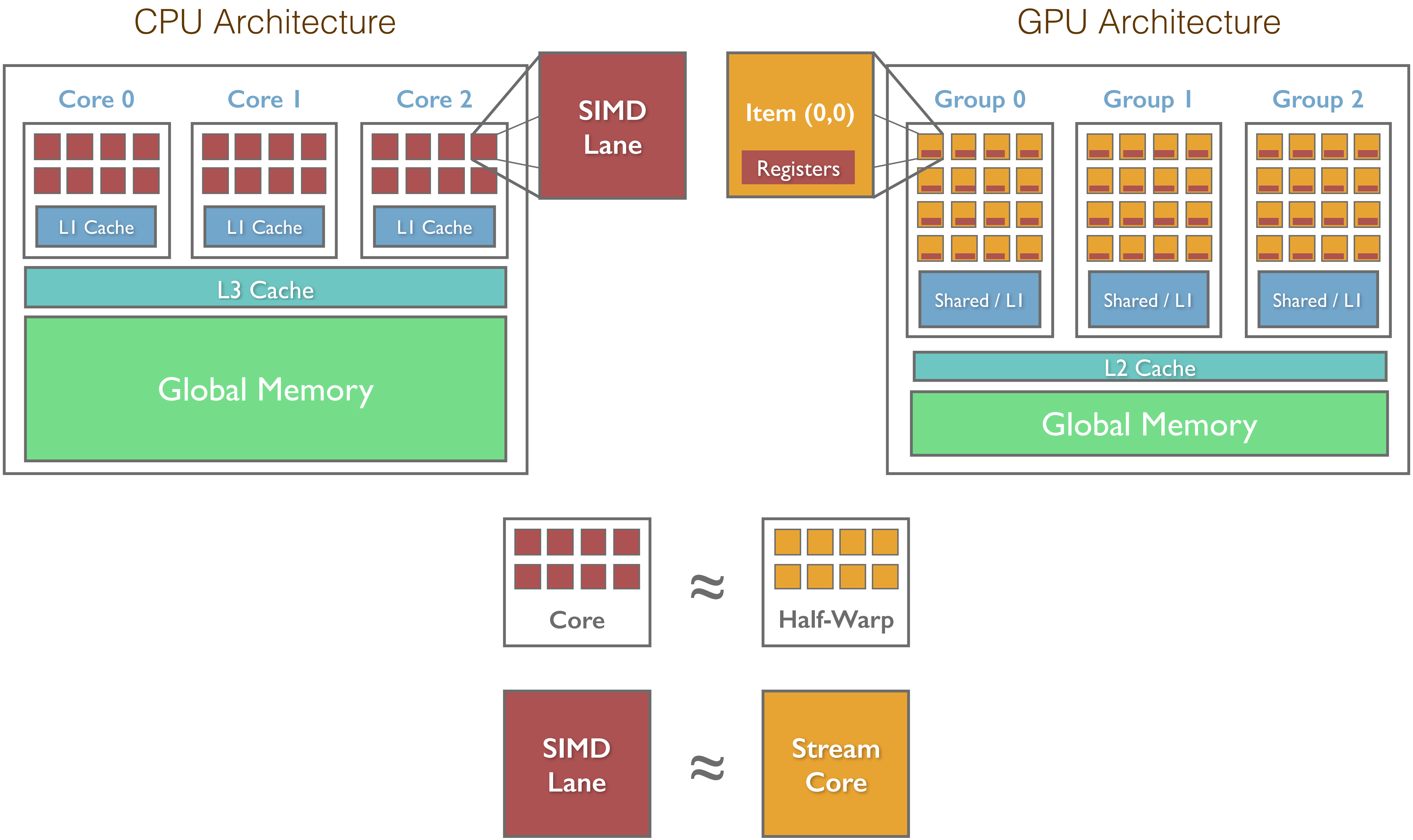
Parallel model

- Independent work-groups are launched
- Work-groups contain groups of work-items, “parallel” threads.



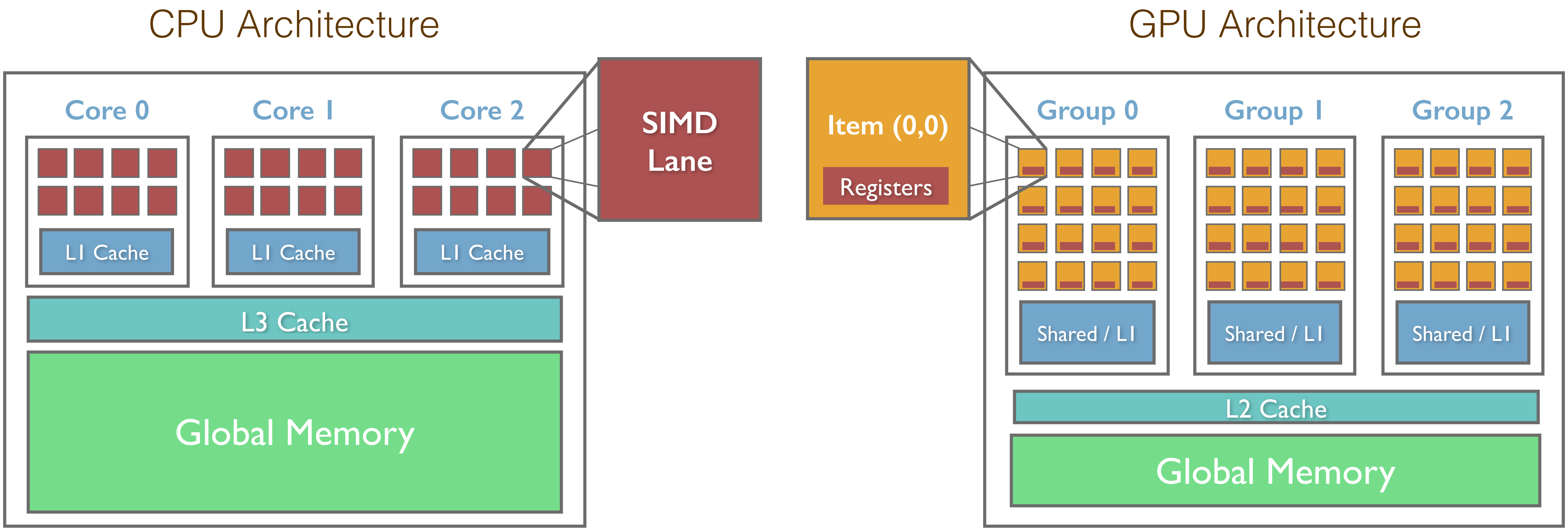
GPU Functions (Kernels) execute at the work-item level

Parallelization Paradigm



Computational hierarchies are similar

Parallelization Paradigm

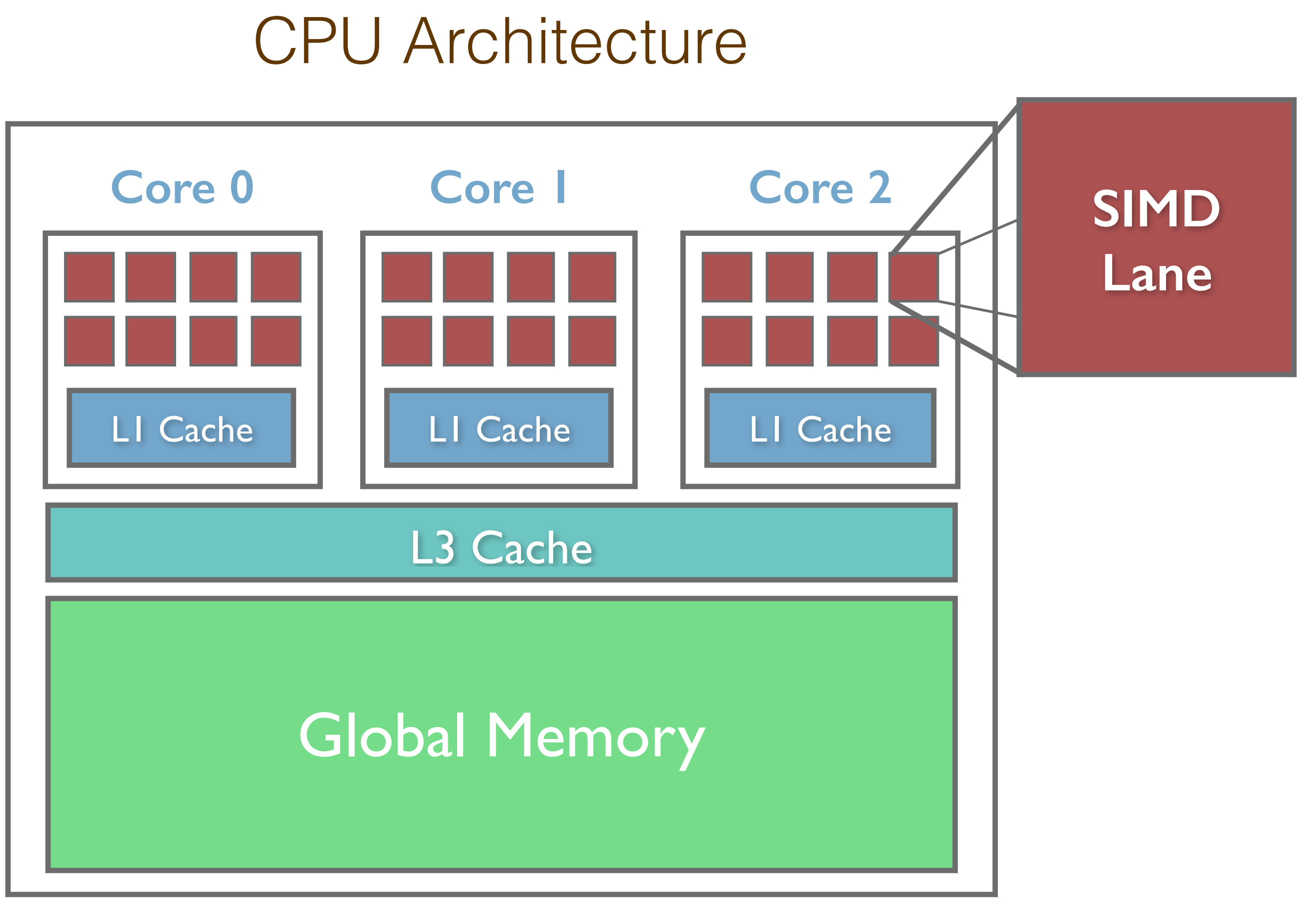


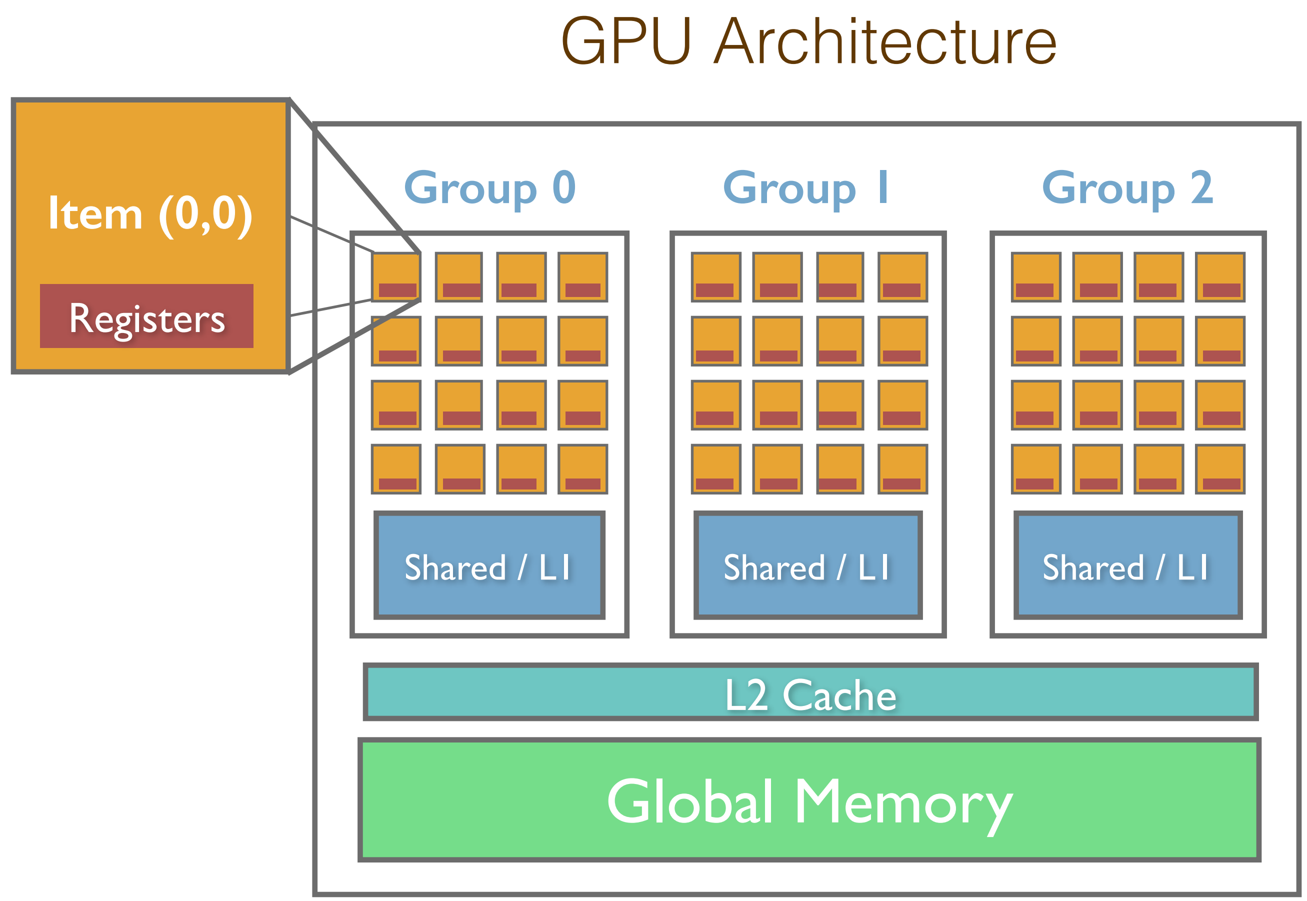

```
void cpuFunction(){
  #pragma omp parallel for
  for(int i = 0; i < work; ++i){

    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

Parallelization Paradigm






```
void ompFunction(){
// for each thread {
  for(thread's work){

    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

Many-core: challenges

Yet programming massively parallel processors is not mainstream…

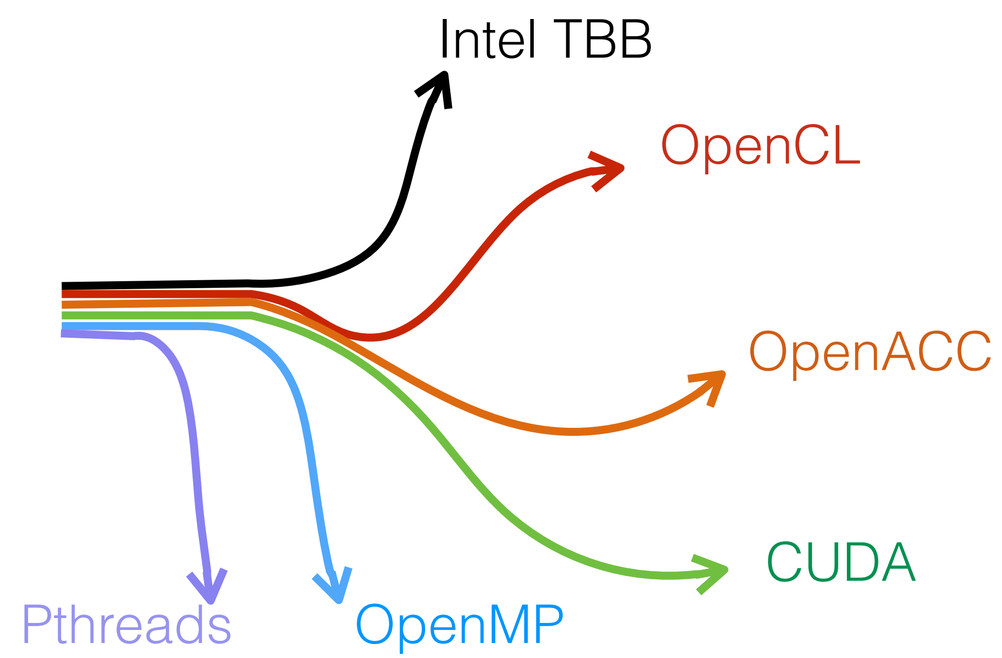
Cell processor die shot (removed)

Processor die shot (removed)

IBM Cell Processors: 2001-2009

IBM Blue Gene/Q 2011-2015*

Uncertain: fast turnover of architectures & toolchains



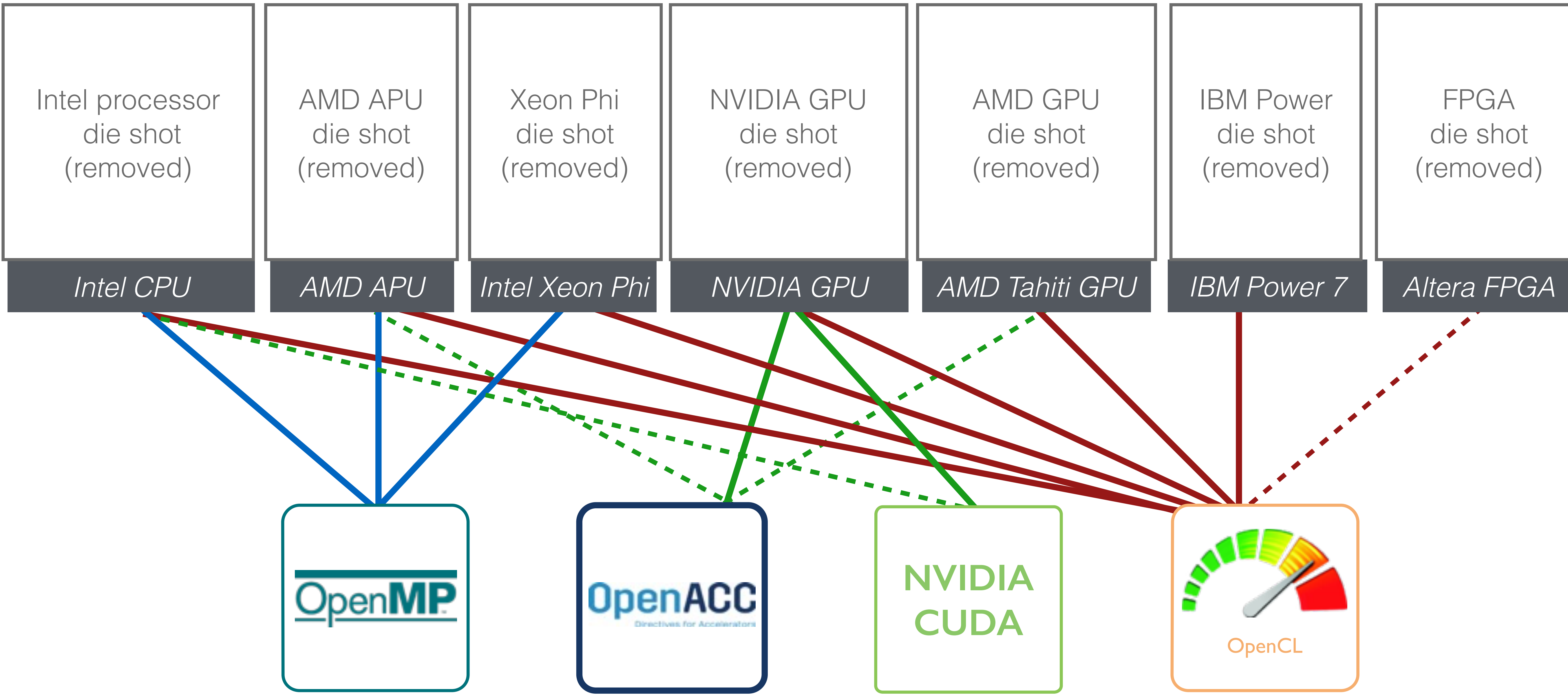
Fragmentation: variety of threading languages

"In the context of today's CPU landscape, then, redesigning your application to run multithreaded on a multicore machine is a little like learning to swim by jumping into the deep end"

The Free Lunch Is Over, Herb Sutter, 2009.

Many-core: fragmentation

Zoo of competing architectures and programming models (with vendor bias)



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

Motivation

Uncertainty

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

Portability

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

Performance

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

Programmability

- Expose parallel paradigm … without introducing an exotic programming model

Portability Approaches

Directive approach

- Use of optional [`#pragma`]'s to give compiler transformation hints
- Aims for portability, performance and programmability

Source-to-source approach

- Compiler tools can be used to translate across specifications/languages
- Performance is not always portable
- Maintenance of original and translated codes

Wrapper approach

- Create a tailored library with optimized functions
- Restricted to a pre-canned set of operations
- Flexibility comes from functors/lambdas at compile-time

Portability Approaches

Directive approach

- Use of optional [#pragma]'s to give compiler transformation hints
- Aims for portability, performance and programmability

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

Portability Approaches

Directive approach

- Not centralized anymore due to the offload model
- OpenACC and OpenMP begin to resemble an API rather than code decorations

OpenACC

```
double a[100];
#pragma acc enter data copyin(a)
// OpenACC code
#pragma acc exit data copyout(a)
```

OpenACC

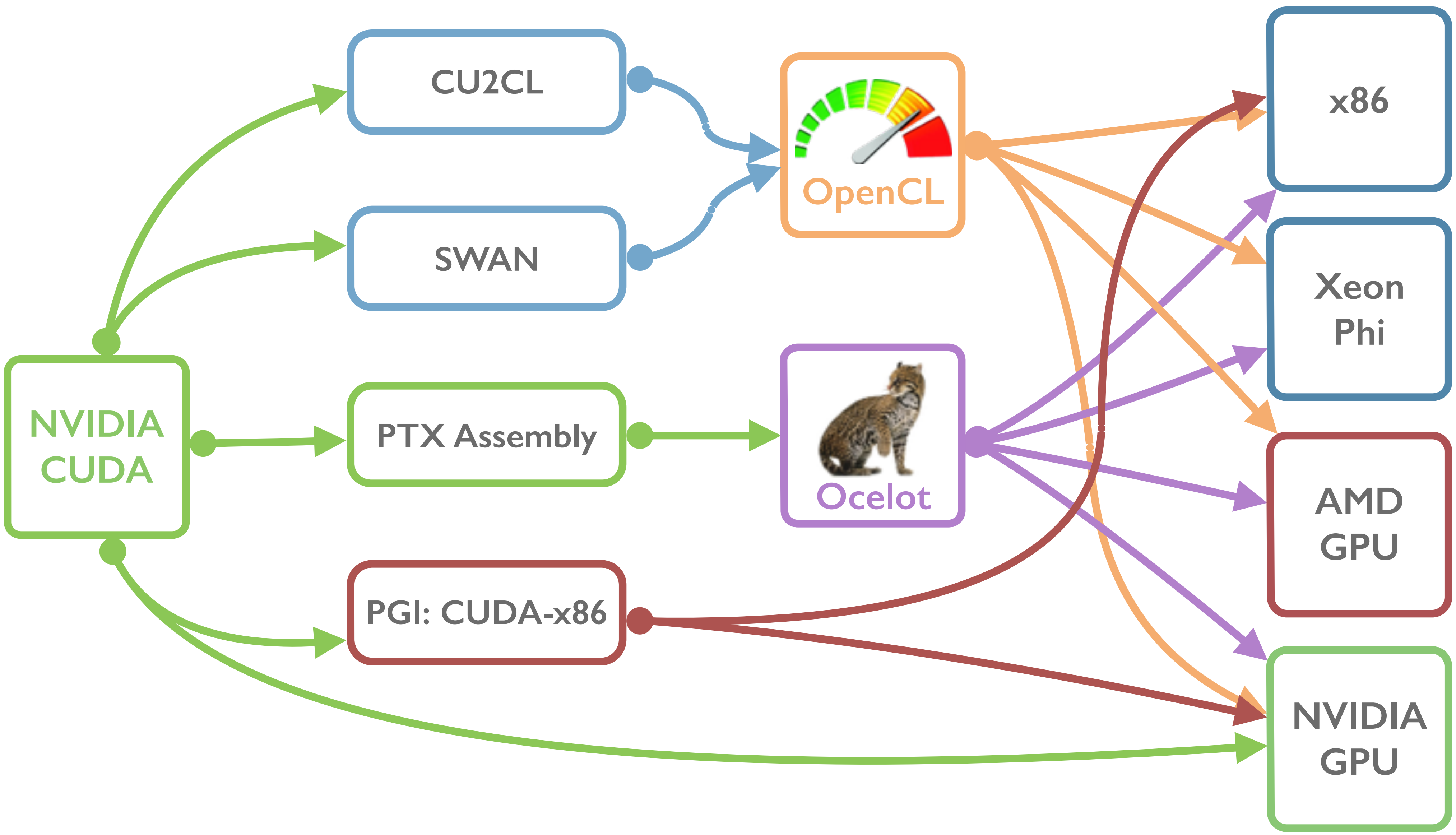
```
class Matrix {
  double *v;
  int len
  Matrix(int n) {
    len = n;
    v = new double[len];
    #pragma acc enter data create(v[0:len])
  }
  ~Matrix() {
    #pragma acc exit data delete(v[0:len])
    delete[] v;
  }
};
```

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

Portability Approaches

Source-to-source approach

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



GPU Ocelot seems to be the most active of these projects

Portability Approaches

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 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!

Portability Approaches

Numerous approaches to portability

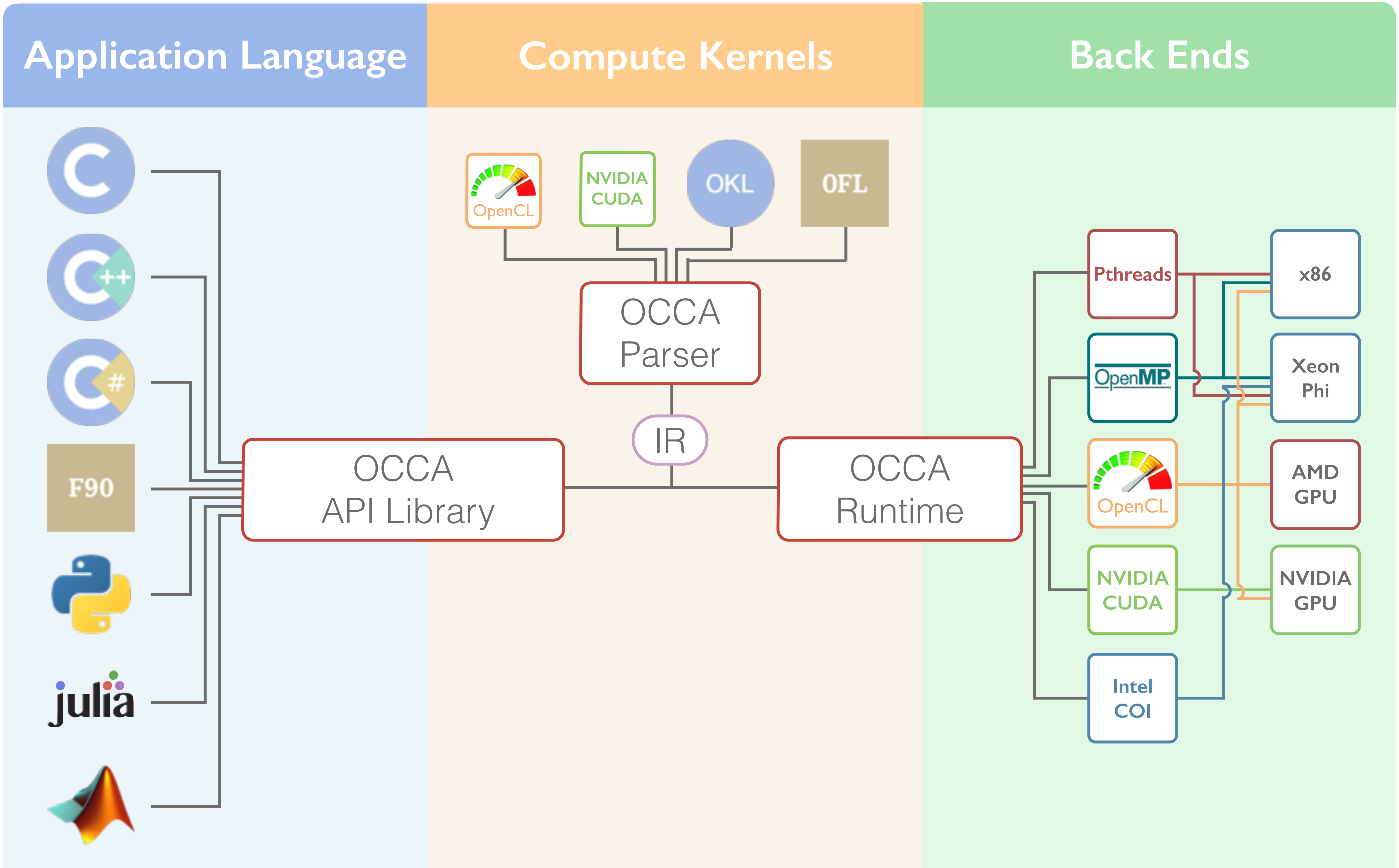
API	Type	Front-ends	Kernel	Back-ends
Kokkos	ND arrays	C++	Custom	CUDA & OpenMP
VexCL	Vector class	C++	-	CUDA & OpenCL
RAJA	Library	C++	C++ Lambdas	CUDA, OpenMP, OpenACC
OCCA2	API, Source-to-source, Kernel Languages	C,C++,C#, F90, Python,MATLAB, Julia	OpenCL, CUDA,& custom unified kernel language	CUDA, OpenCL, pThreads,OpenMP, Intel COI
CU2CL *	Source-to-source	App	CUDA	OpenCL
Insieme	Source-to-source compiler	C	OpenMP,Cilk, MPI, OpenCL	OpenCL,MPI, Insieme IR runtime
Trellis	Directives	C/C++	#pragma trellis	OpenMP, OpenACC, CUDA
OmpSs	Directives + kernels	C,C++	Hybrid OpenMP, OpenCL, CUDA	OpenMP, OpenCL, CUDA
Ocelot	PTX Translator	CUDA	CUDA	OpenCL

DSL

Compiler

OCCA emphasis: lightweight and extensible.
** Wu Feng et al @ VT !*

OCCA: accessible portability



github.com/tcew/OCCA2
libocca.org

OCCA Application Programming Interface (API)

`occa::device` (C++ Class)

`occa::memory` (C++ Class)

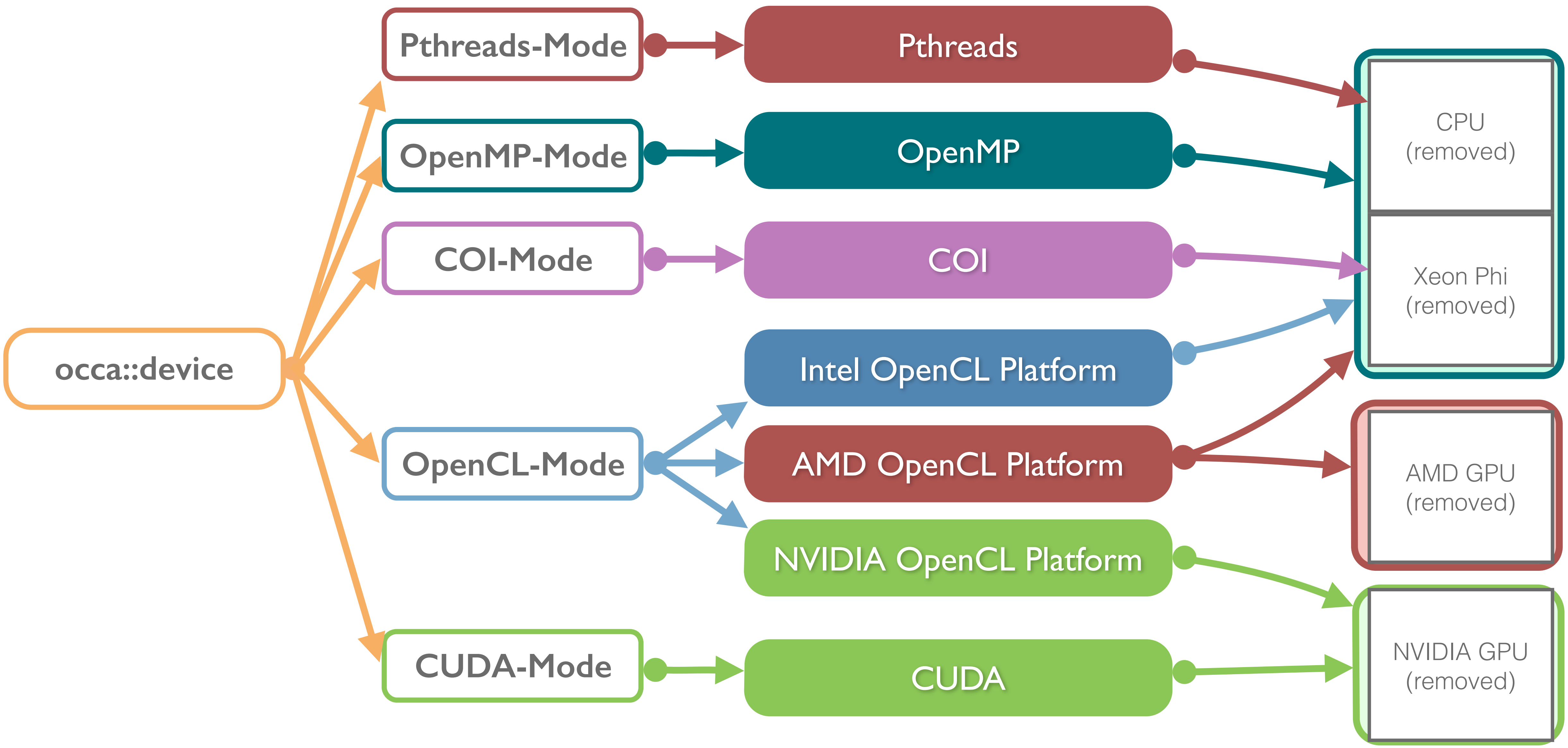
`occa::kernel` (C++ Class)

Offloading model is generalized to 3 parts

OCCA Application Programming Interface (API)

`occa::device` (C++ Class)

- Choose between using the CPU or available accelerators
- In charge of allocating memory and compiling kernels

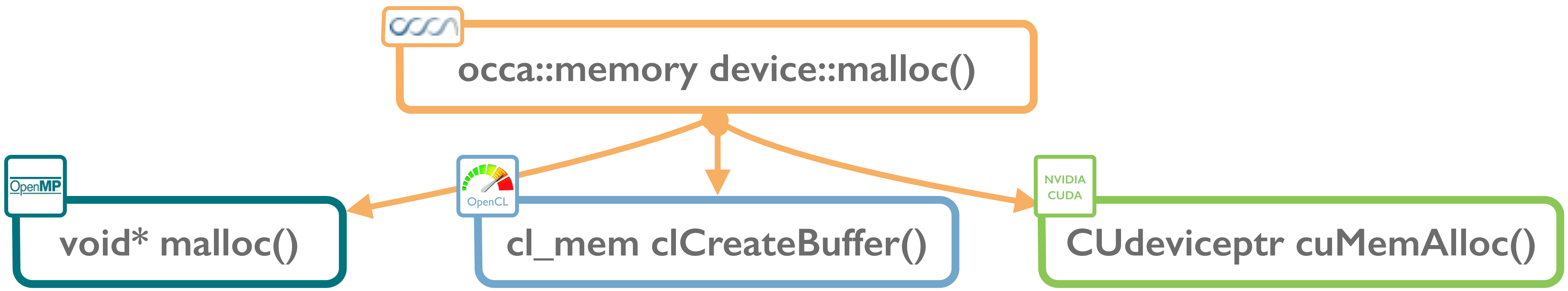


Offloading model is generalized to 3 parts

OCCA Application Programming Interface (API)

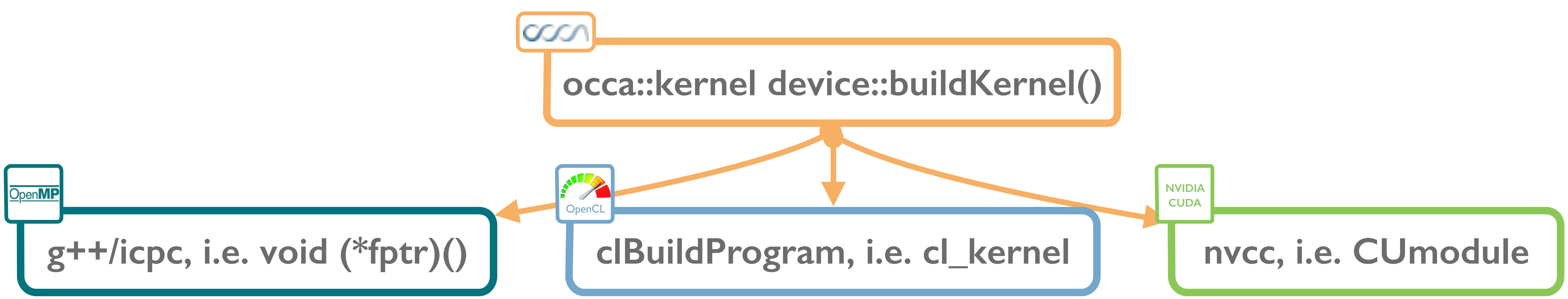
`occa::memory` (C++ Class)

- Abstracts the memory handles found in each language
- Asynchronous memory transfers are supported



`occa::kernel` (C++ Class)

- Uses run-time compilation
- Kernel binaries are cached to prevent re-compiling



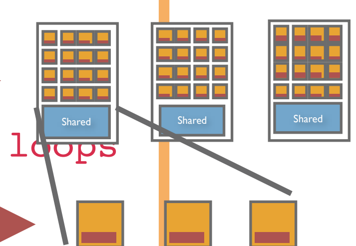
Offloading model is generalized to 3 parts

OKL: OCCA Kernel Language

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

OKL: OCCA Kernel Language

Outer-loops

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

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



  for(outer){
    for(inner){
    }
  }




  ...
}
```

OKL

Data dependencies are found through a variable dependency graph

OKL: OCCA Kernel Language

Outer-loops

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

```
kernel void kernelName(...){
  for(outer){
    for(inner){
    }
  }

  for(outer){
    for(inner){
    }
  }

  for(outer){
    for(inner){
    }
  }
}
```

OKL

Data dependencies are found through a variable dependency graph

OKL: OCCA Kernel Language

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

OKL: OCCA Kernel Language

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

OKL: OCCA Kernel Language

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
   }
}
```

```
exclusiveVar = 0
exclusiveVar = 1
exclusiveVar = 2
.
.
.
```

OKL

Local barriers are auto-inserted

OFL: OCCA Fortran Language

Description

- Translates to OKL and then to OCCA IR with code transformations
- Parallel loops are explicit through the inner and outer DO-labels

```
kernel subroutine kernelName(...)
  ...

  DO groupY = 1, yGroups, outer1
   DO groupX = 1, xGroups, outer0 // Work-group implicit loops
    DO itemY = 1, yItems, inner1
     DO itemX = 1, xItems, inner0 // Work-item implicit loops
      // GPU Kernel Scope
     END DO
    END DO
   END DO
  END DO

  ...
end subroutine kernelName
```

OFL

Shared and exclusive memory

```
integer(4), shared    :: sharedVar(16,30)
integer(4), exclusive :: exclusiveVar, exclusiveArray(10)
```

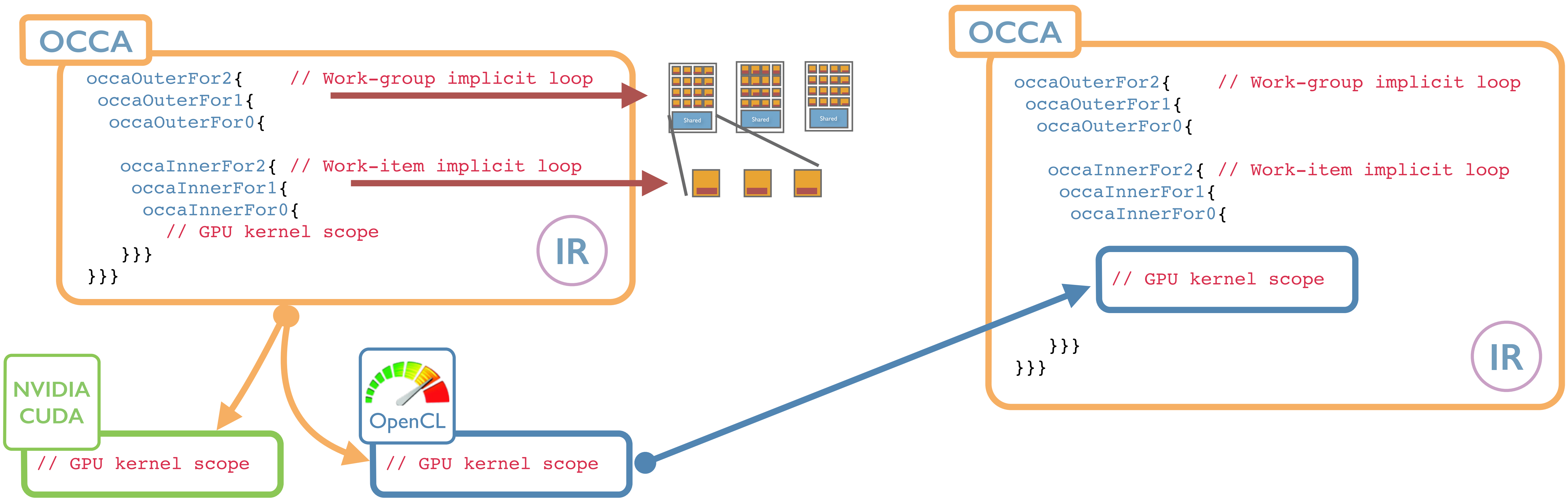
OFL

Because [OFL -> OKL], all features added to OKL are inherintly added to OFL

OpenCL/CUDA to OCCA IR

Description

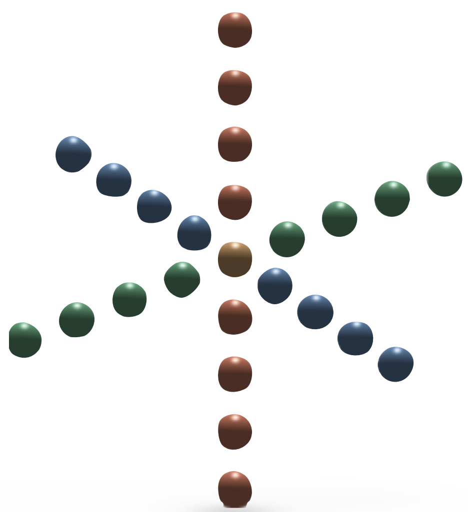
- Parser can translate OpenCL/CUDA kernels to OCCA IR*
- Although OCCA IR was derived from the GPU model, there are complexities



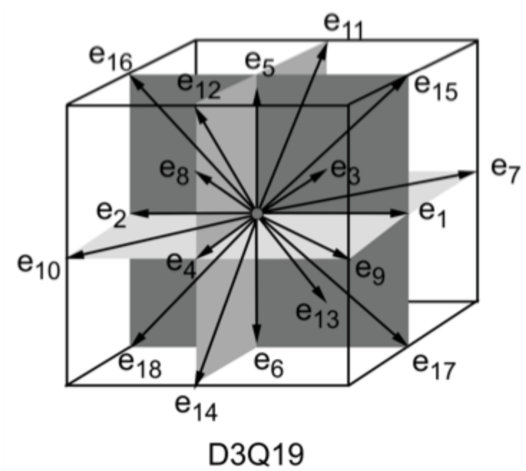
Since we derived OCCA IR from the GPU model, the inverse should be easy … right?

OCCA2: apps & benchmarks

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

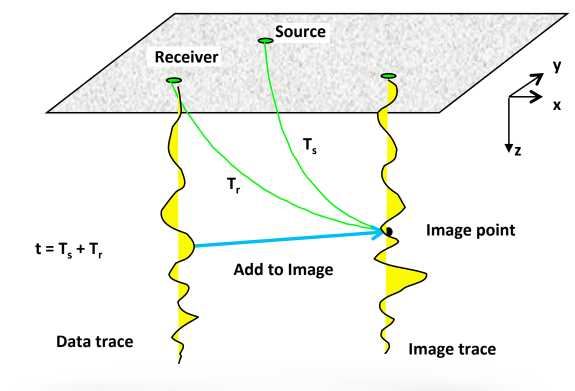


FDTD RTM:wave equation (+15-33%)

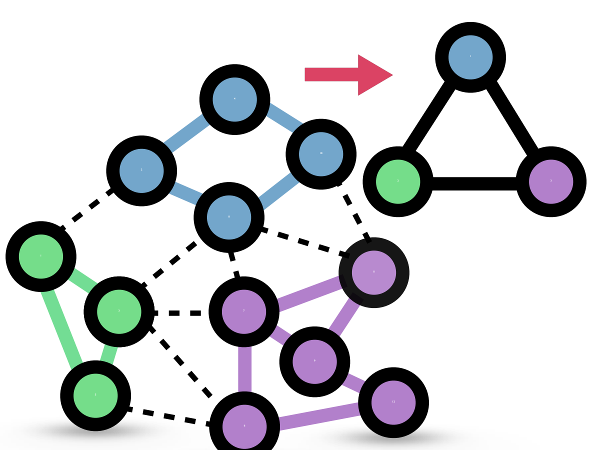


Lattice Boltzmann for Core Sample Analysis

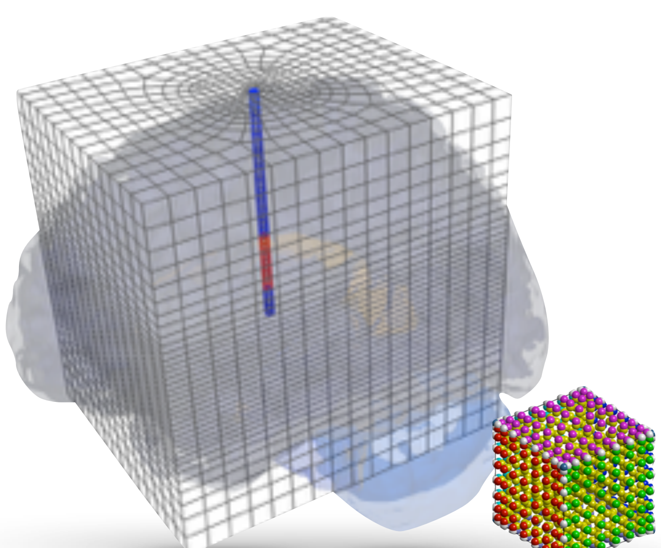
DG for seismic wave simulations



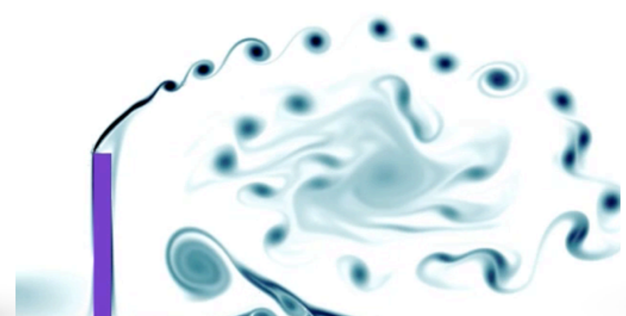
Kirchhoff migration (kernel up to 2x faster).



ALMOND: algebraic multigrid library



MDACC: FEM model of laser tumor ablation

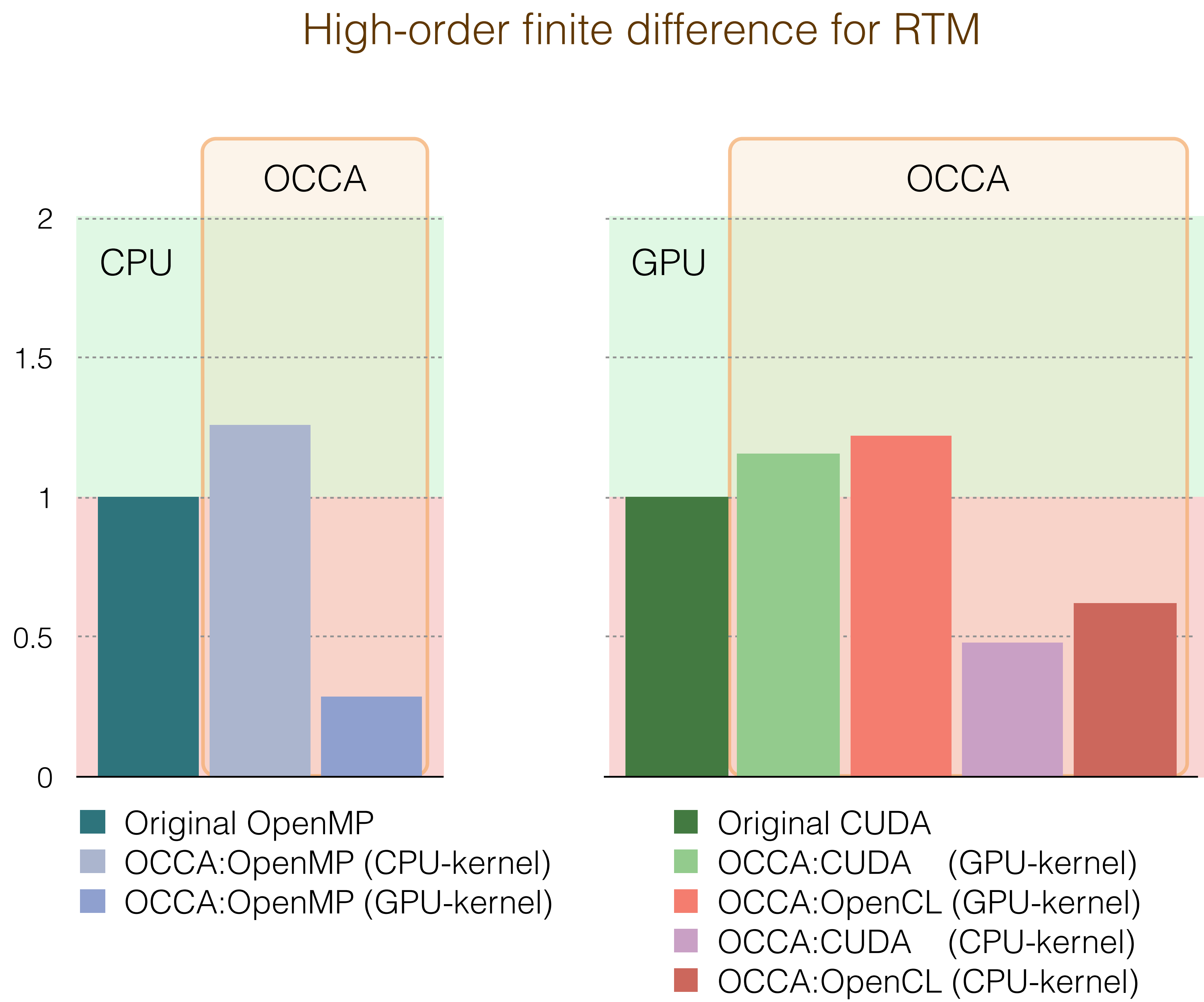


DG compressible Navier-Stokes solver

DG shallow-water & 3D ocean modeling

+ more applications

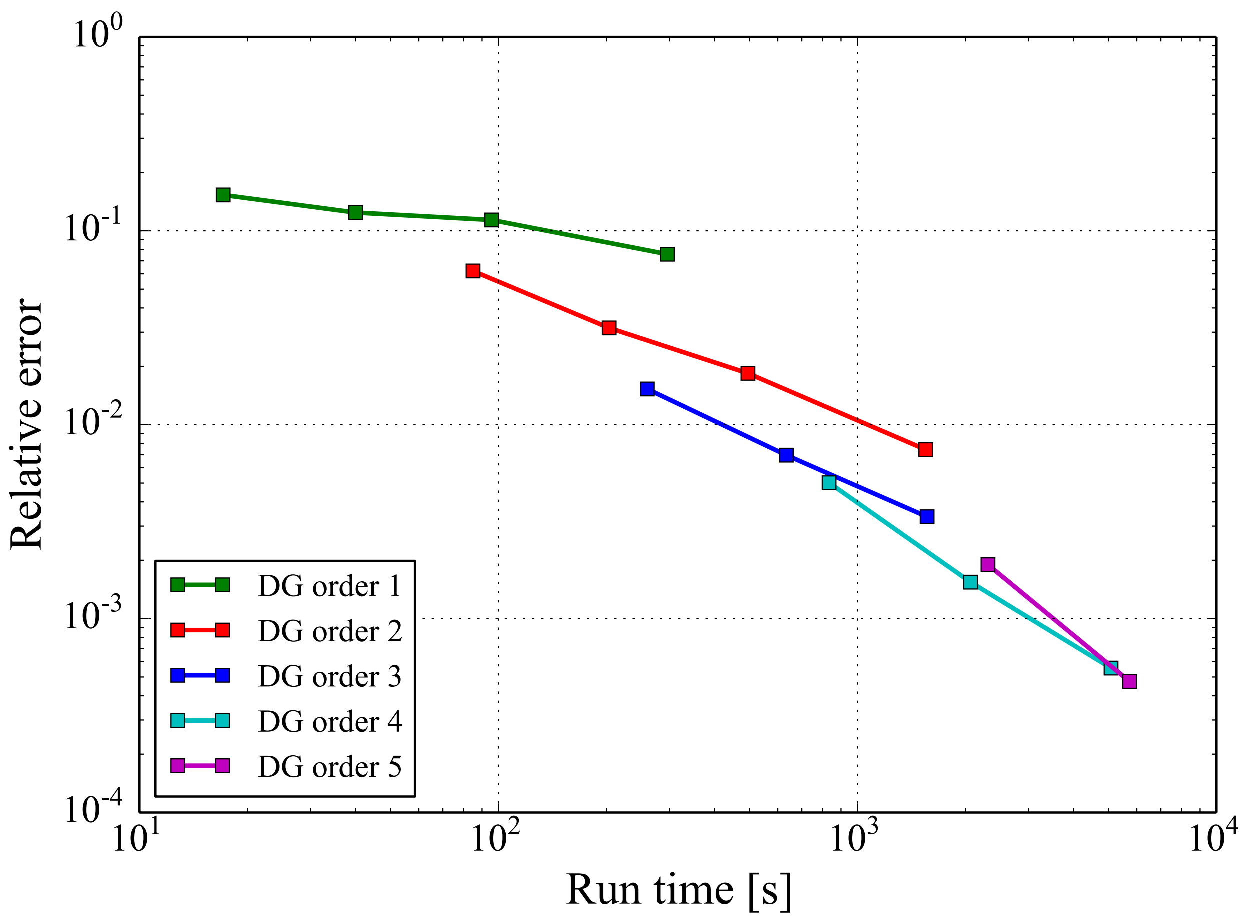
OCCA2: apps & benchmarks

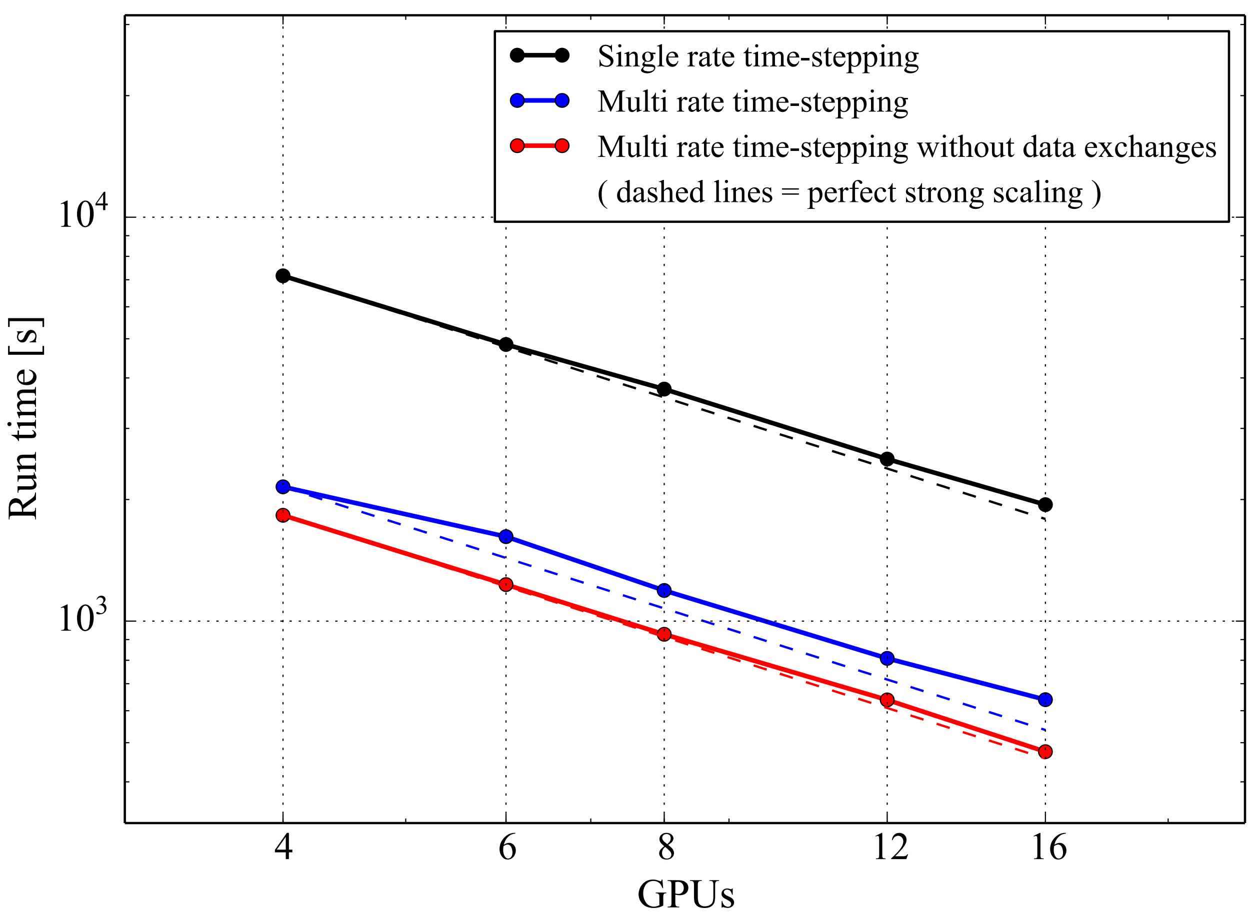


OpenMP : Intel Xeon CPU E5-2640
OpenCL/CUDA : NVIDIA Tesla K10

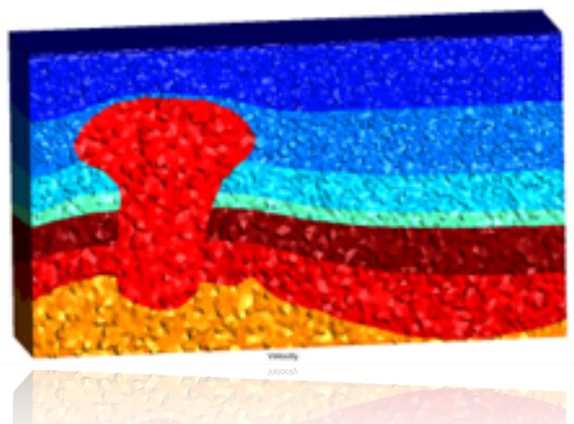
OCCA2: apps & benchmarks

High-order discontinuous Galerkin for RTM

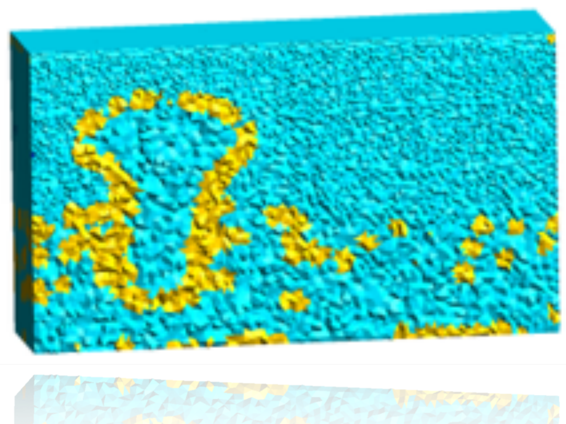




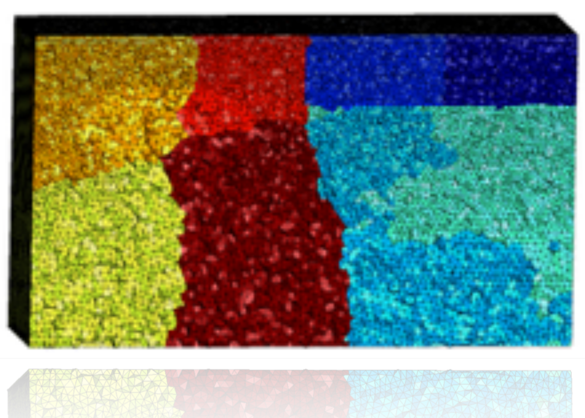
Discretization



Multi-rate scheme



Distributed



OCCA2: apps & benchmarks

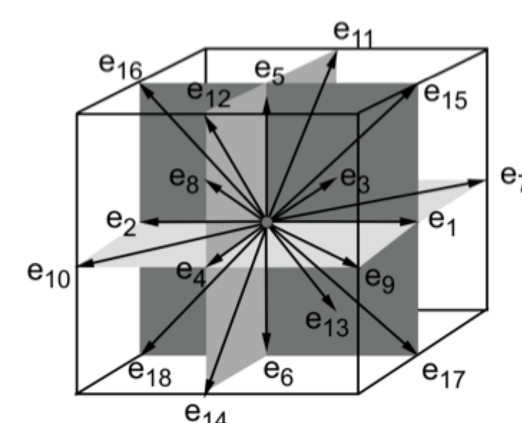
Lattice Boltzmann Method in Core Sample Analysis

Comparison across platforms (Normalized with original code)

	API Mode	Device	Model	Wall Clock	BW (GB/s)	Speedup
	Ref dense code [-O3 in gcc 4.8]	CPU 1-core	Intel i7-5960X	1290	—	x 1
OCCA	OpenMP	CPU	Intel i7-5960X	11.12	22	x 116
OCCA	OpenCL: Intel	CPU	Intel i7-5960X	11.18	22	x 115
OCCA	OpenCL: AMD	GPU	AMD 7990	1.39	176	x 928
OCCA	OpenCL: NVIDIA	GPU	GTX 980	1.25	196	x 1032
OCCA	CUDA: NVIDIA	GPU	GTX 980	1.20	205	x 1075

Comparison across platforms (Normalized with OCCA::OpenMP)

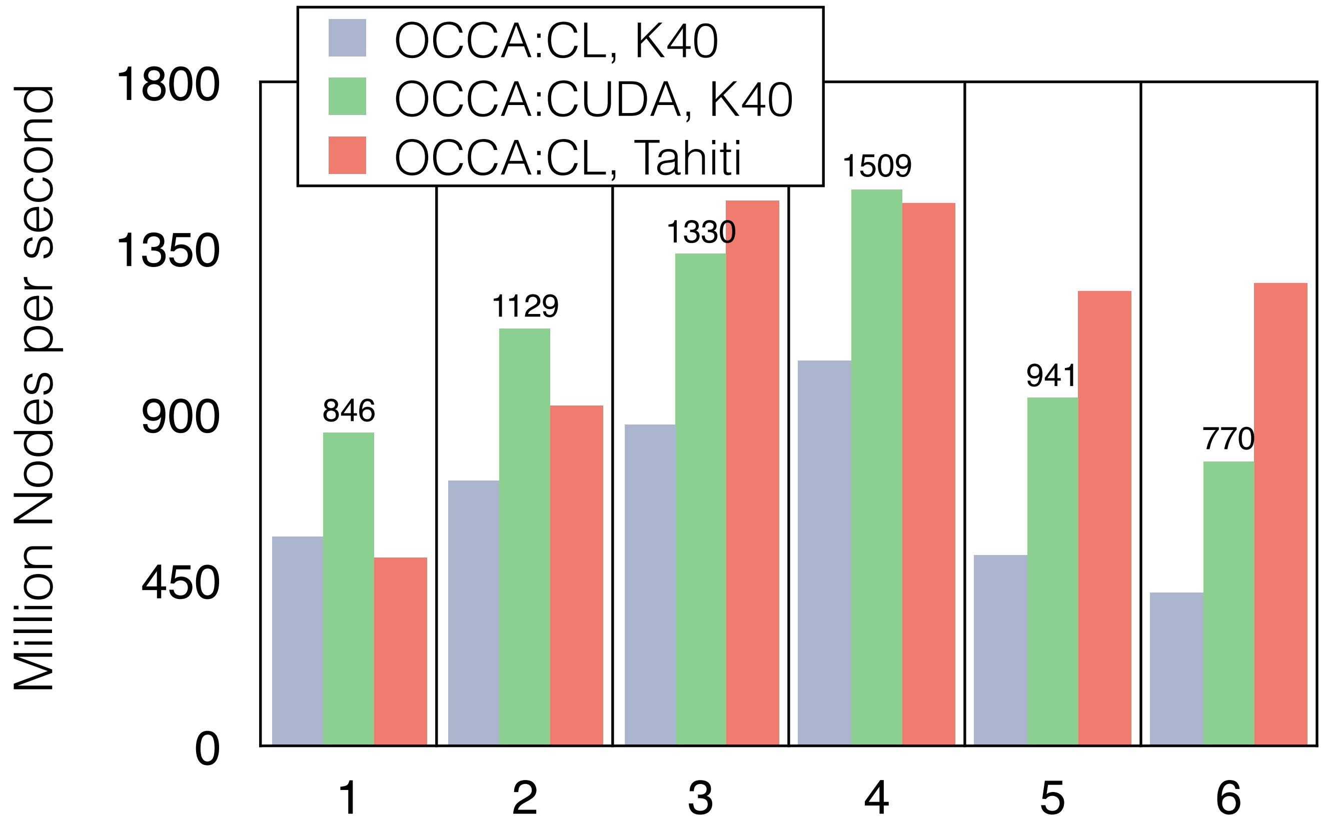
	API Mode	Device	Model	Wall Clock	BW (GB/s)	Speedup
OCCA	OpenMP	CPU	Intel i7-5960X	11.12	22	x 1.0
OCCA	OpenCL: Intel	CPU	Intel i7-5960X	11.18	22	x 1.0
OCCA	OpenCL: AMD	GPU	AMD 7990	1.39	176	x 8.0
OCCA	OpenCL: NVIDIA	GPU	GTX 980	1.25	196	x 8.9
OCCA	CUDA: NVIDIA	GPU	GTX 980	1.20	205	x 9.3

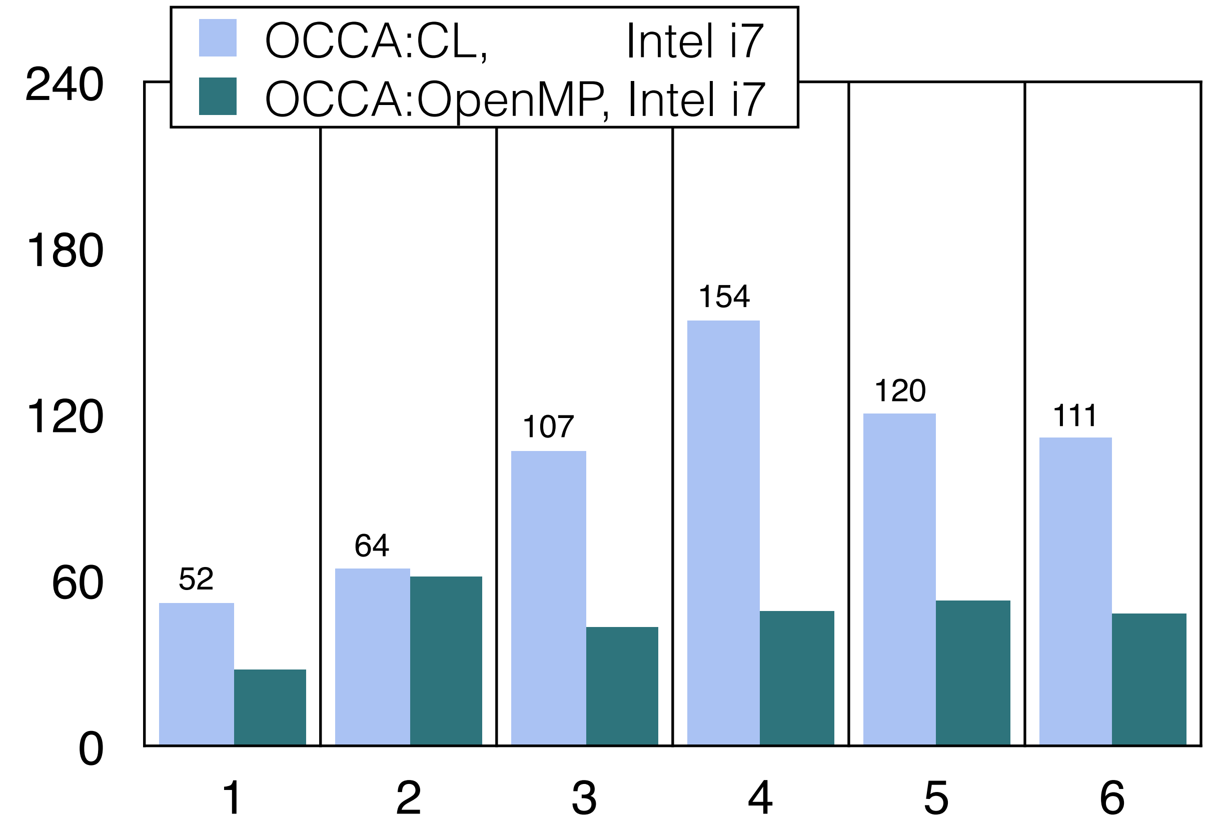


D3Q19

OCCA2: apps & benchmarks

Discontinuous Galerkin for shallow water equations





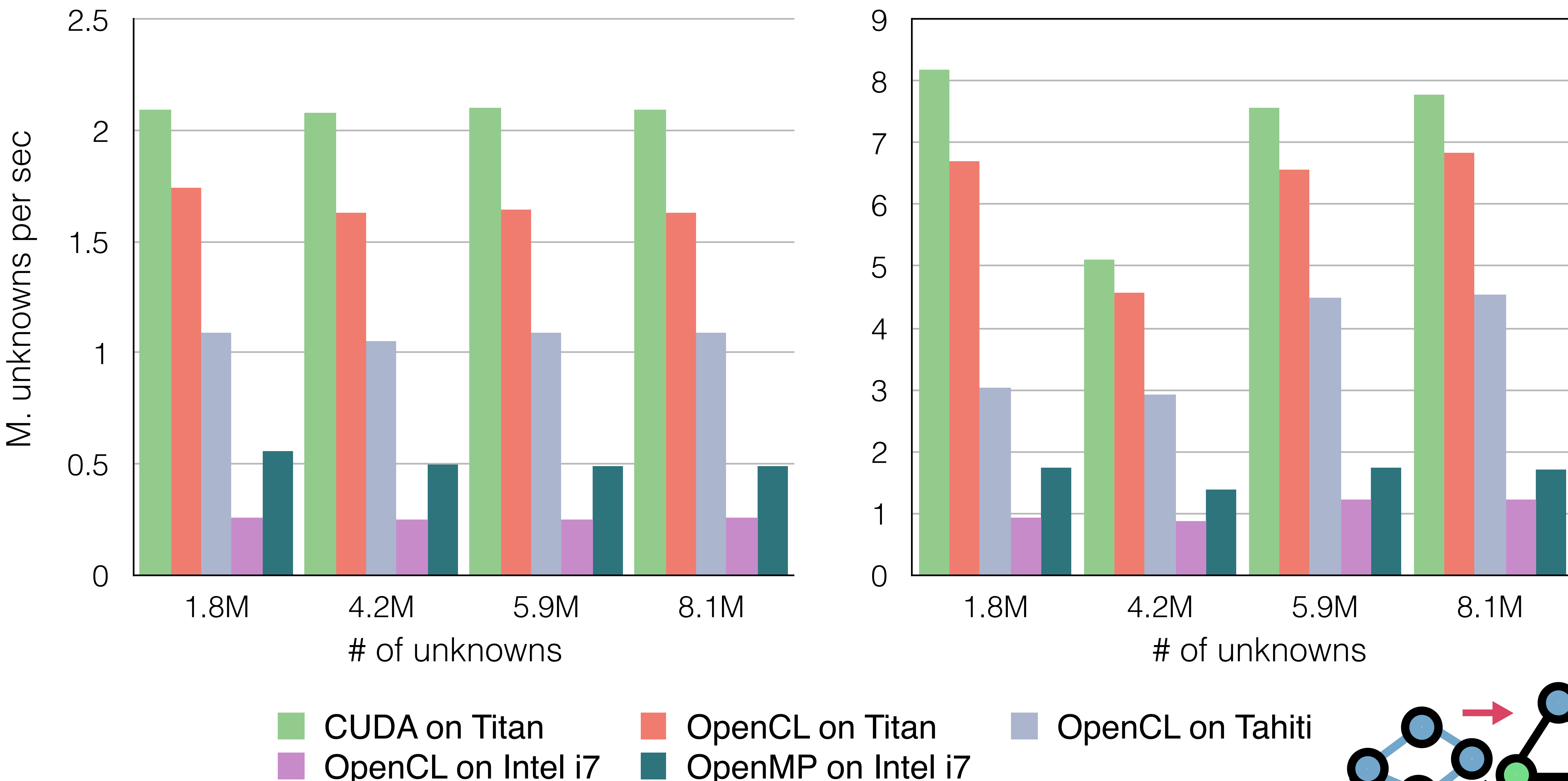
Polynomial Order	Compute-Time vs Real-Time
1	x650
2	x208
3	x95
4	x47

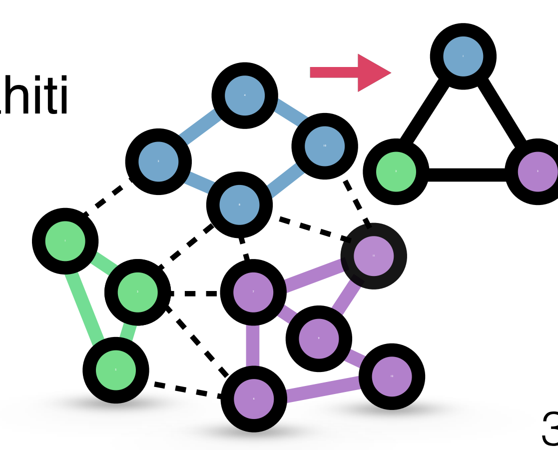
OCCA2: apps & benchmarks

Algebraic multigrid for elliptic problems

Setup Time

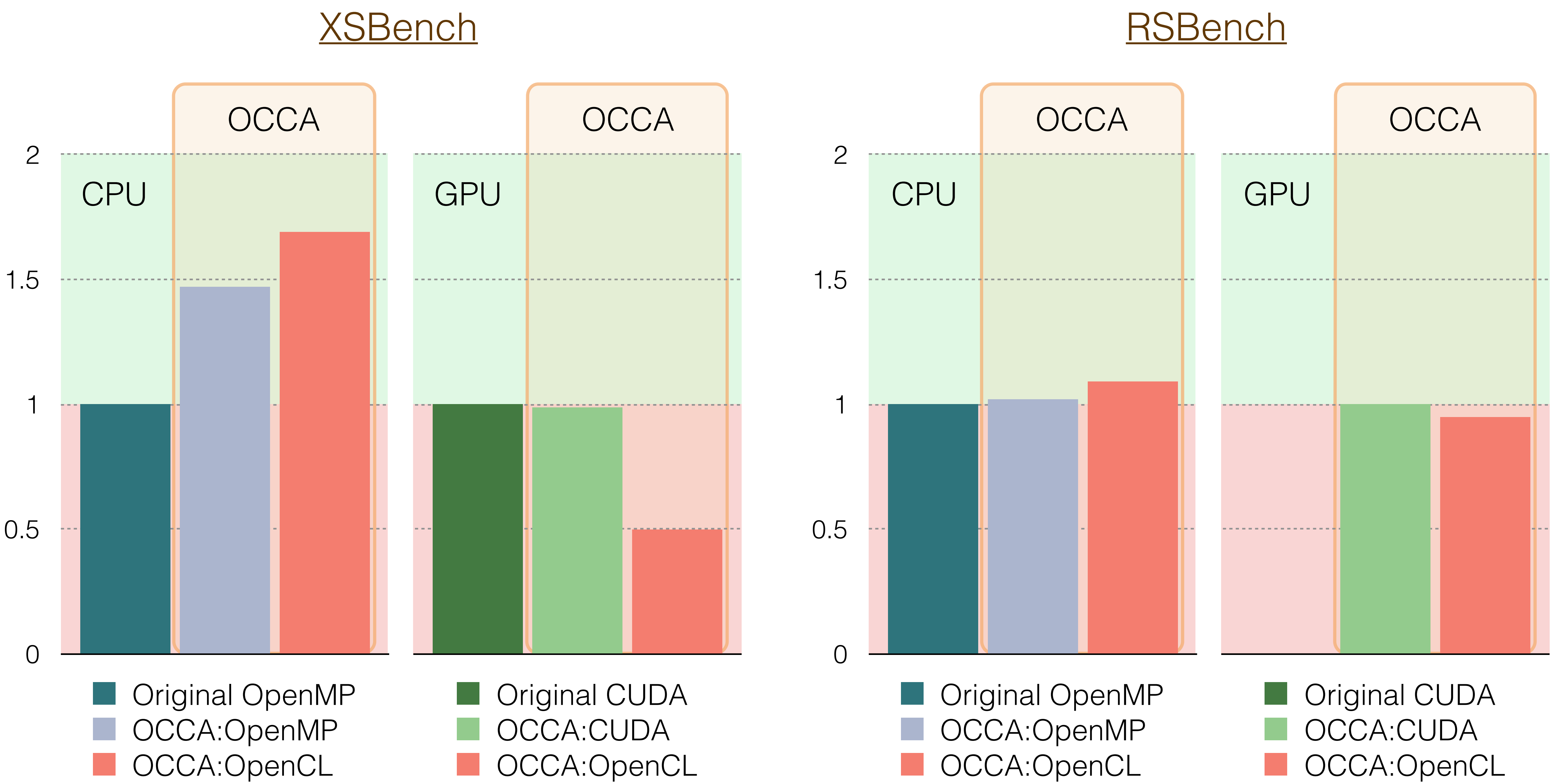
Solve Time





OCCA2: apps & benchmarks

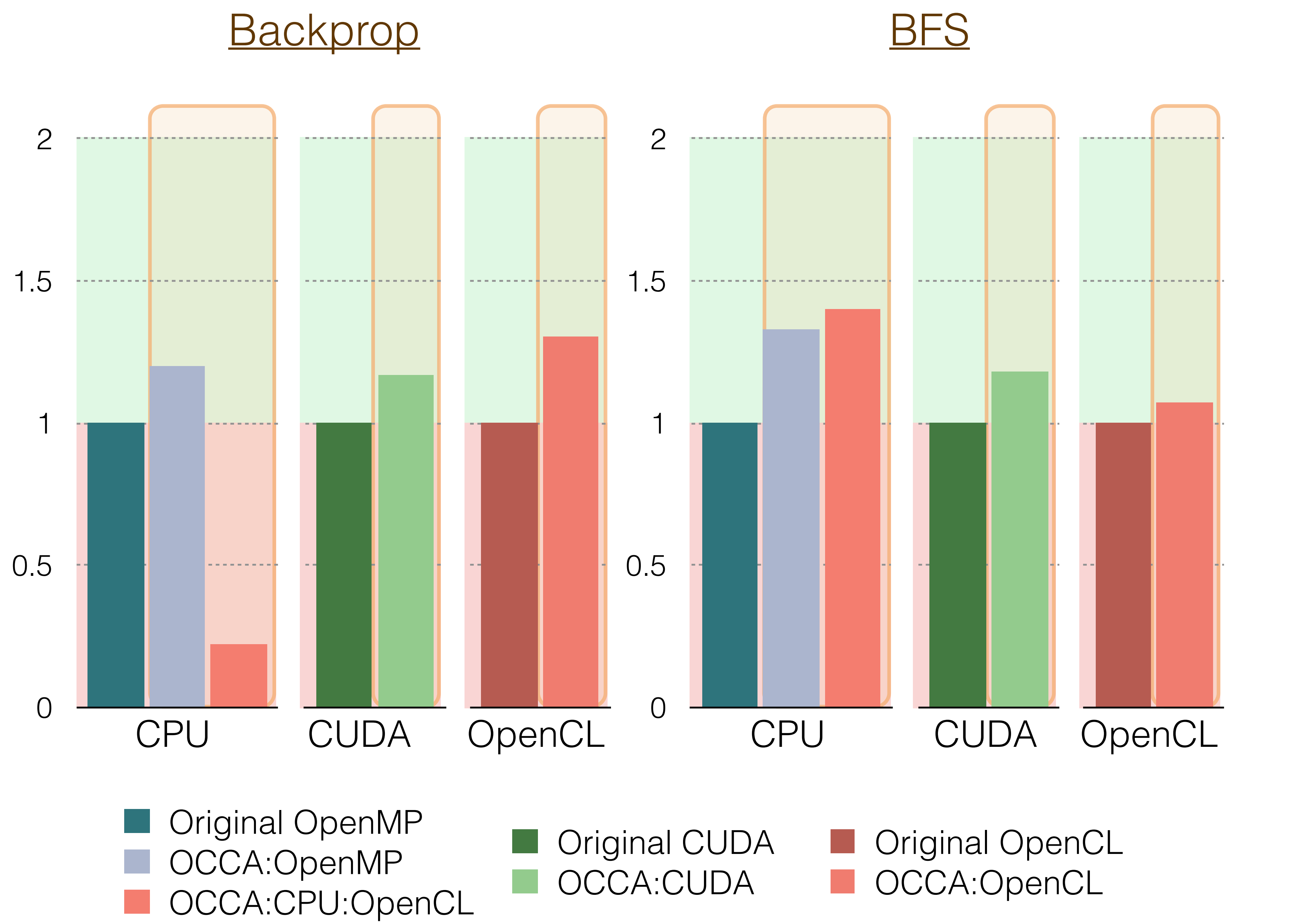
Monte Carlo for neutronics
Collaborations with Argonne National Lab



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

OCCA2: apps & benchmarks

Two of our ported Rodinia benchmarks, based on the “11 Dwarves”



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

https://github.com/dmed256/OCCA-Benchmarks

Live Demo

Installation

- Download OCCA and template code for this session at:

```
git clone https://github.com/tcew/OCCA2.git
git clone https://github.com/tcew/OG15.git
```

- Setup environment variables and compile

```
cd OCCA2
export OCCA_DIR=$PWD
export CXX=clang++ # or any compiler you want
export PATH=$PATH:$OCCA_DIR/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OCCA_DIR/lib
make -j
```

- Try it out

```
occainfo # Displays available devices
cd examples/addVectors
make -j
./main
```

Interactive Demo

Description

- Go to the previously downloaded template code:

```
cd $OCCA_DIR/../OG15 # If you downloaded it next to OCCA
cd matrixTranspose
```

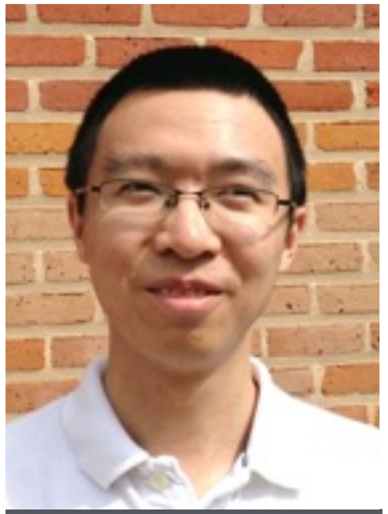
- This demo will let you (the attendees) try coding an OKL kernel (or OFL)

- Translate the serial matrix transpose seen in the code into an OKL/OFL kernel

Rajesh Gandham
Oceans & AMG
CAAM Grad

David Medina
HPC & Accelerators
CAAM Grad

Reid Atcheson
Adaptivity/Helmholtz
CAAM Grad



Zheng (Frank) Wang
Radiation Transport
CAAM Grad

Arturo Vargas
Hermite+DG
CAAM Grad

Jesse Chan
Advanced Numerics
Post-doc

Axel Modave
Accelerate Numerics
Industrial Post-doc

A few of us ^ will be going around, feel free to raise your hand for questions