NPS-NRL-Rice-UIUC Collaboration on Navy Atmosphere-Ocean Coupled Models on Many-Core Computer Architectures: Overview and Progress

Lucas C. Wilcox (LCW)\textsuperscript{1}  
Timothy Campbell (TC)\textsuperscript{2}  
Timothy Warburton (TCW)\textsuperscript{4}  
Francis X. Giraldo (FXG)\textsuperscript{1}  
Andreas Klöckner (AK)\textsuperscript{3}  
Timothy Whitcomb (TW)\textsuperscript{5}

\textsuperscript{1}Naval Postgraduate School  
\textsuperscript{2}Naval Research Laboratory Stennis Space Center  
\textsuperscript{3}University of Illinois at Urbana-Champaign  
\textsuperscript{4}Virginia Tech  
\textsuperscript{5}Naval Research Laboratory Monterey

ESPC AOLI Meeting 2015
Open Concurrent Computing Abstraction

gNUMA Mini-app
Partial NUMA/NEPTUNE Software Stack

- NEPTUNE
- NUMA
- bigNUMA
- gNUMA
- bfam
- Loo.py
- p4est
- OCCA
- MPI
- OpenCL
- CUDA
- OpenMP
Open Concurrent Computing Abstraction (OCCA)

- **Goal:**
  - Portable performance.
  - Forward planning: transitions in many-core programming models.
  - Unified lightweight abstraction that does not reduce performance.
  - Address “immaturity” of Accelerator compilers and frameworks.

- **Core Features:**
  - Native APIs for C, F90, C++, Python, and Julia applications.
  - Runtime offload of tasks to OpenMP, CUDA, or OpenCL.
  - Simple C based OCCA Kernel Language (OKL).

- **New in 2015:**
  - **F90-like** OCCA Fortran Language (OFL).
  - Compute kernels: source code lexer, parser, analyzer, & translators.
  - Kernel hashing and caching for scalable runtime compilation.
  - Optional: automated data movement between HOST and DEVICE.
## Existing Portability Solutions

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Front-ends</th>
<th>Kernel</th>
<th>Back-ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kokkos</td>
<td>ND arrays</td>
<td>C++</td>
<td>Custom</td>
<td>CUDA &amp; OpenMP</td>
</tr>
<tr>
<td>VexCL</td>
<td>Vector class</td>
<td>C++</td>
<td>-</td>
<td>CUDA, OpenCL, OpenACC</td>
</tr>
<tr>
<td>RAJA</td>
<td>Library</td>
<td>C++</td>
<td>C++ Lambdas</td>
<td>CUDA, OpenMP, OpenACC</td>
</tr>
<tr>
<td>OCCA</td>
<td>API</td>
<td>C, C++, F90, Python, ...</td>
<td>OpenCL, OKL, OFL, CUDA</td>
<td>CUDA, OpenCL, OpenMP</td>
</tr>
<tr>
<td>CU2CL</td>
<td>Source-to-source</td>
<td>App</td>
<td>CUDA</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Insieme</td>
<td>Source-to-source</td>
<td>C</td>
<td>OpenMP, Cilk, MPI, OpenCL</td>
<td>OpenCL, MPI</td>
</tr>
<tr>
<td>Trellis</td>
<td>Directives</td>
<td>C/C++</td>
<td>#pragma trellis</td>
<td>CUDA, OpenMP, OpenACC</td>
</tr>
<tr>
<td>OmpSs</td>
<td>Directives + kernels</td>
<td>C,C++</td>
<td>Hybrid OpenMP, OpenCL, CUDA</td>
<td>OpenMP, OpenCL, CUDA</td>
</tr>
<tr>
<td>Ocelot</td>
<td>PTX Translator</td>
<td>CUDA</td>
<td>CUDA</td>
<td>OpenCL</td>
</tr>
</tbody>
</table>

**Table**: Comparison of approaches to many-core portability
OCCA Overview

Figure: Workflow for the OCCA framework.
program main
  use occa

  ! HOST variables
  integer(4) :: i, N = 5, platformID = 0, deviceID = 0
  real(4), allocatable :: a (:), b (:), c (:)

  ! OCCA structures
  type(occaDevice) :: device
  type(occaKernel) :: add
  type(occaMemory) :: o_a, o_b, o_c

  ! allocate and populate HOST arrays
  allocate(a(1:N), b(1:N), c(1:N), stat = alloc_err)
  do i=1,N
    a(i) = i - 1
    b(i) = 2 - i
  end do

  ! set up OCCA device
  device = occaCreateDevice("mode=CUDA, deviceID=0")

  ! build OCCA compute kernel
  add = occaDeviceBuildKernel(device, "add.okl", "add")

  ! allocate DEVICE storage and populate
  o_a = occaDeviceMalloc(device, int(N,8)*4_8)
  o_b = occaDeviceMalloc(device, int(N,8)*4_8)
  o_c = occaDeviceMalloc(device, int(N,8)*4_8)
  call occaCopyPtrToMem(o_a, a(1), int(N,8)*4_8, 0_8);
  call occaCopyPtrToMem(o_b, b(1));

  ! queue kernel task
  call occaKernelRun(add, occaTypeMem_t(N), o_a, o_b, o_c)

  ! copy data to HOST and print
  call occaCopyMemToPtr(c(1), o_c);
  print *, "c = ", c(:)
end program main

kernel void add(int N, float *a, float *b, float *c){
  // tile loops by 16
  for(int offset = 0; offset < N; offset+=16; outer0)
    for(int i = offset; i < offset+16; ++i; inner0)
      if (i < N)
        c[i] = a[i] + b[i];
}

kernel subroutine add(N, a, b, c)

  integer(4), intent(in) :: N
  real(4) , intent(in) :: a (:), b (:)
  real(4) , intent(out) :: c (:)
  integer :: offset , i

  ! tile loop by 16
  do offset = 1, N, 16, outer0
    do i = offset , offset+16
      if (i <= N) then
        c(i) = a(i) + b(i)
      end if
    end do
  end do
end subroutine add
OCCA: roadmap

- Enhance OpenMP backend:
  - Collaboration with Alexander Heinecke @ Intel Research.
  - SSE/AVX/AVX2/AVX512 intrinsics.
  - Flatten exclusive variables.
  - Flexible data remapping.
  - Intel Xeon Phi optimizations.
  - Add new OpenMP4 backend?

- Portable primitive operations library:
  - MAGMA BLAS interface.
  - Example reduction:
    - OpenCL: uni-barrier reductions.
    - CUDA: register shuffle.
    - OpenMP: #pragma omp parallel reduction.

---

Figure: Intel Knights Landing

Open Concurrent Computing Abstraction

gNUMA Mini-app
gNUMA: testbed mini-app for accelerating NUMA

- Testbed for rapidly tuning OCCA based NUMA kernels.

- Challenges:
  - Exploit hex tensor product.
  - Minimize data movement.
  - High-order elements.
  - Work partitioning.

- Solutions:
  - Scanning slabs of nodes.
  - Coalescing solution access.
  - 24+ compute kernels.
  - On-demand geo-factors.

- Roadmap:
  - Intel KNL reference kernels.

Figure: Thread slab strategies
gNUMA:OCCA:CUDA explicit dynamics kernels

- **Device:**
  - NVIDIA GTX 980.
  - Maxwell class GPU.
  - Bandwidth peak 224GB/s.
  - Streaming reduction benchmark \( \approx 170\)GB/s.

- **Thread model:**
  - OCCA:CUDA

- **Kernels:**
  - Horizontal flux divergence.
  - Vertical flux divergence.
  - Surface flux lift.
  - Time step update.

---

![Memory bandwidth (GB/s) vs. Polynomial Degree](chart)

- **Memory bandwidth (GB/s):**
  - Horizontal volume
  - Vertical volume
  - Surface
  - Update
Device:
- NVIDIA GTX 980.
- Maxwell class GPU.
- Bandwidth peak 224GB/s.
- Streaming reduction benchmark \( \approx 170 \text{GB/s} \).

Thread model:
- OCCA:OPENCL

Kernels:
- Horizontal flux divergence.
- Vertical flux divergence.
- Surface flux lift.
- Time step update.
Device:
- AMD HD7970
- Tahiti class GPU.
- Bandwidth peak 264GB/s.
- Streaming reduction benchmark \(\approx 220\text{GB/s}\).

Thread model:
- OCCA:OPENCL

Kernels:
- Horizontal flux divergence.
- Vertical flux divergence.
- Surface flux lift.
- Time step update.
**gNUMA:OCCA:CUDA explicit FEM diffusion kernels**

**Device:**
- NVIDIA GTX 980.
- Maxwell class GPU.
- Bandwidth peak 224GB/s.
- Streaming reduction benchmark $\approx$ 170GB/s.

**Work in progress:**
- Online geometric factor.
- Repartition diffusion ops.
- Change thread slabbing.
- Loo.py . . .