

OPENACC BOF at SC21

More Science. Less programming.

<https://www.openacc.org/events/openacc-birds-feather-bof-sc21>

Tuesday, November 16, 2021 | 5:15 to 6:45 PM CST | Online

OpenACC BOF @ SC21

Tuesday, Nov 16, 2021

- Welcome and OpenACC Organizational Update – Jack Wells, NVIDIA (**5 Minutes**)
- OpenACC Specification Update – Jeff Larkin, NVIDIA (**7 Minutes**)
- Compiler Implementations
 - HPE Updates: Barbara Chapman, HPE (**6 minutes**)
 - GCC Updates: Catherine Moore, Siemens (**6 minutes**)
- Porting Scientific Applications with OpenACC: Real-world Use Cases (**7 minutes**)
 - On the Road to Code Portability – Stéphane Ethier, PPPL
 - Can Fortran's 'do concurrent' Replace Directives for Accelerated Computing? Ron Caplan (Predictive Science)
- Training and Education, Julia Levites, NVIDIA (7 minutes)
- Questions, General Discussion from the BOF. (45 minutes)

OPENACC ORGANIZATION MISSION

The OpenACC Organization is dedicated to helping the research and developer community advance science by expanding their accelerated and parallel computing skills. We have 3 areas of focus: participating in computing ecosystem development, providing training and education on programming models, resources and tools, and developing the OpenACC specification.

Ecosystem
Development

Training/Education

OpenACC Specification

PILLARS OF OPENACC ORGANIZATION

ECOSYSTEM DEVELOPMENT

Work with Language Standards committees

- Apply collective lessons learned
- Bridge gaps in base languages
- Develop roadmap for interoperability

Strive for performant interoperability

TRAINING & EDUCATION

Hackathons/Bootcamps

- Focus on accelerated computing

Training Materials:

- Courses
- Containers
- Workshops

Programs for educators, students, mentors

OPENACC SPECIFICATION

Develop OpenACC Specification by introducing new features and functionality

OPENACC – CELEBRATING 10 YEARS

Accelerating Science. Building Community.



Thank you, OpenACC members, for all the hard work and support!



OPENACC DIRECTIVES

a directive-based parallel programming model designed for
usability, performance and portability

APPLICATIONS

250+
3 out of Top 5

PLATFORMS SUPPORTED

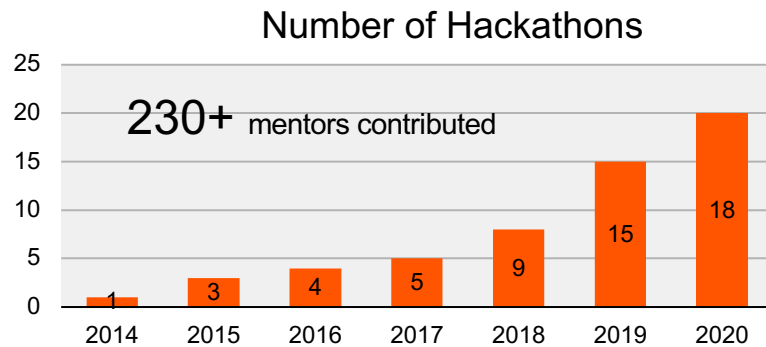
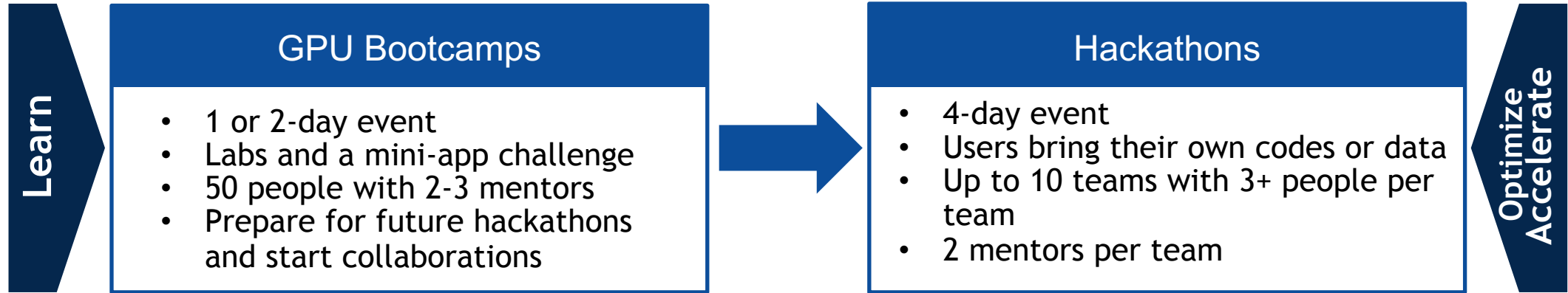
NVIDIA GPU
X86 CPU
POWER CPU
Sunway
ARM CPU
AMD GPU

COMMUNITY

~2900
Slack Members

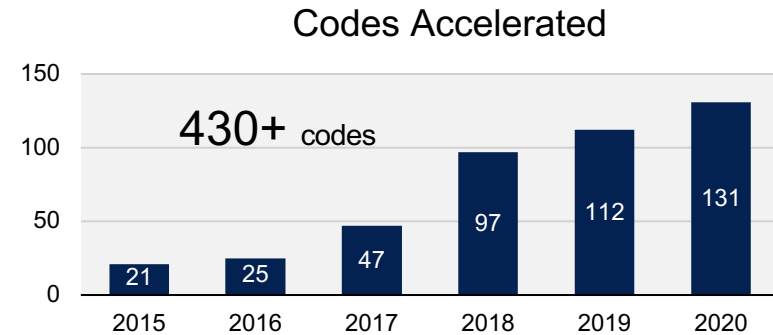
TRAINING AND EDUCATION

GPU Hackathons and Bootcamps



23
Bootcamps
in 2019

27
Bootcamps
in 2020



APPLY TO GPU HACKATHONS

Accelerate your code on GPUs with mentors by your side

- Over 20 events globally.
- 4 full days over 2 weeks.
- Online or in-person.
- 10 teams. 2 mentors per team.
- Free to participate.
- GPU resource is provided.



www.gpuhackathons.org/events

JOIN GPU BOOTCAMPS

Build Confidence in GPU Programming

<https://github.com/gpuhackathons-org/gpubootcamp>

- Global Events
- Self-paced materials
- Instructor-led bootcamps
- AI and HPC content
- Team work and code challenges
- 8 hours per event



www.gpuhackathons.org/events

Technical Committee Update

Activities since SC20

- OpenACC 3.2 is out!!
 - Error Handler
 - Initialize/Shut Down individual devices from runtime API
 - Acc Wait Any
 - Asynchronous Structured Data Regions
 - Many clarifications and reorganizations
- LLVM Upstreaming
 - Community effort to upstream CLACC and FLACC Efforts
 - Participation to-date by several vendors, labs, and universities
 - Must more help is needed!

Error Handler

- Developer or Tool may register an Error Callback
- From the error callback:
 - Inspect/Diagnose the issue
 - Clean-up and/or checkpoint as-needed
 - Gracefully shutdown
- No error recovery, only inspection
- Great Side Effect: Significantly improved definition of error conditions throughout the specification!

Improved Device Initialization

Before 3.2

```
// Initialize all devices of the
// default type
acc_init(acc_device_default);

// Initialize individual device via
// the pragma
#pragma acc init \
device_type(acc_device_default) \
device_num(0)
```

Version 3.2

```
// Initialize just device 0 of
// default type
acc_init_device(acc_device_default, 0);

// Initialize individual device via
// the pragma
#pragma acc init \
device_type(acc_device_default) \
device_num(0)
```

Wait Any

```
#pragma acc data copyin(list[0:10])
{
    int queues[10];
    for ( int i=0; i < 10; i++ )
    {
        // Do some unbalanced operation on several queues
        #pragma acc enter data copyin(list[i].member[0:list[i].size]) async(i)
        // Put the queue number in the queues list, the index and queue number
        // do not need to match, like they do here.
        queues[i] = i;
    }
    int next;
    // Look for queue that is ready to process
    while ( (next = acc_wait_any(10,queues)) >= 0 )
    {
        // Remove this queue from consideration next time around
        queues[next] = acc_async_sync;
        // Process work dependent on above
        #pragma acc parallel loop
        {
            for ( int j=0; j < list[i].size; j++ )
            {
                // do stuff
            }
        }
    }
}
```

With `acc_wait_any` it's possible to poll asynchronous work queues and find a queue that's ready now.

Useful for load-imbalances and other unpredictable timings.

Modeled after `MPI_Waitany` and similar APIs.

Asynchronous Structure Data Regions

```
// Mark this entire data region as asynchronous on queue 0
#pragma acc data copy(A[0:N]) async(0)
{
    // Execution MAY continue here before data allocation and
    // copies complete

    // This region MUST wait on queue 0 to ensure data is ready or
    // enqueue itself
    // in queue 0 as well.
    #pragma acc parallel loop async(1) wait(0)
    for ( int i=0; i < N; i++ ) { ; }

    // Since the data region MUST NOT copy or deallocate A until
    // the parallel
    // region has finished, this wait is necessary.
    #pragma acc wait(1) async(0)
}
// Execution MAY continue here before data copies and
// deallocation occurs

// It's necessary to wait on queue 0 before operating on A to
// ensure the device
// has finished any data operations.

#pragma acc wait(0)
```

Previously structured data regions had to be synchronous, in part due to data allocation.

Now data operations at the beginning and end of data regions can be made asynchronous.

Please be sure you understand the ramifications.

LLVM Upstreaming

Bringing OpenACC to Clang and Flang

- The CLACC and FLACC efforts to create OpenACC compilers are being upstreamed.
- Monthly telecom with collaborators.
- We need more help!
 - Review patches
 - Create patches
 - Create test cases