

OpenMP[®]

SC25 OpenMP Tech Talk Series



PyOMP: Parallel Programming with OpenMP in Python

**Giorgis Georgakoudis,
Lawrence Livermore National Laboratory**

PyOMP: Parallel Programming with OpenMP in Python



Giorgis Georgakoudis (LLNL), Todd A. Anderson (Bodo.ai), Stuart Archibald (Anaconda Inc.),

Bronis R. De Supinski (LLNL), Timothy G. Mattson (Merly.ai)

PyOMP: Portable, productive, high-performance parallel programming in Python

OpenMP

- easy, portable, popular

 **python**TM

- easy, productive, popular

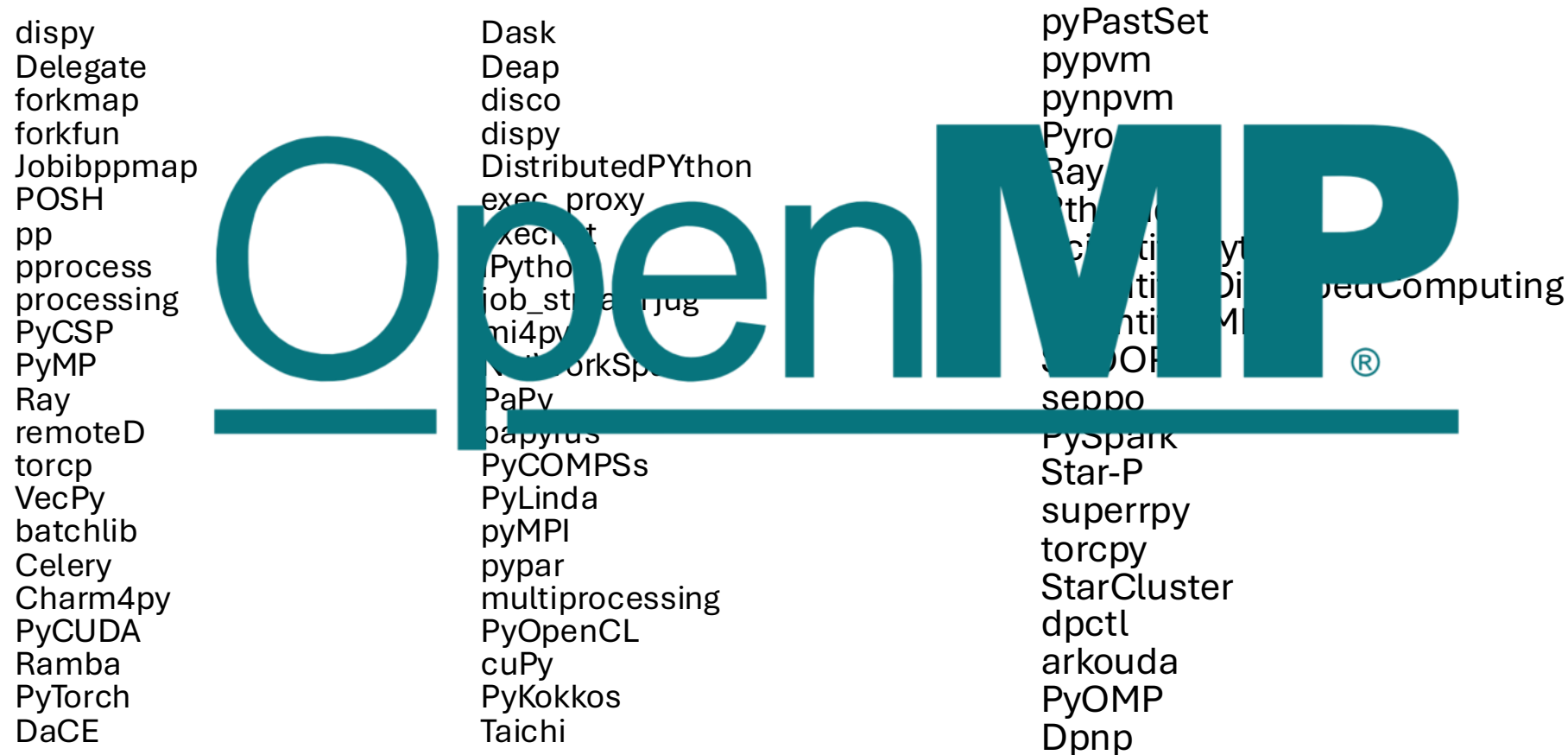
 **Numba**

- easy, fast, extensible JIT compiler

PyOMP



Python's parallel programming software landscape is highly fragmented



PyOMP supports familiar OpenMP syntax in a pythonic way

```
Imports { 1 from numba.openmp import njit
          2 from numba.openmp import openmp_context as openmp
          3 from numba.openmp import omp_get_thread_num
          4
          5 @njit
          6 def hello():
with context construct { 7     with openmp("parallel"):
                        8         print("Hello from thread", omp_get_thread_num())
                        9
                        10 hello()
```

```
Hello from thread 31
Hello from thread 3
...
Hello from thread 20
...
```

PyOMP supports the OpenMP common core on CPUs and...

<code>with openmp("parallel") :</code>	Create a team of threads. Execute a parallel region
<code>with openmp("for") :</code>	Use inside a parallel region. Split up a loop across the team.
<code>with openmp("parallel for") :</code>	A combined construct. Same as parallel followed by a for .
<code>with openmp ("single") :</code>	One thread does the work. Others wait for it to finish
<code>with openmp("task") :</code>	Create an explicit task for work within the construct.
<code>with openmp("taskwait") :</code>	Wait for all tasks in the current task to complete.
<code>with openmp("barrier") :</code>	All threads arrive at a barrier before any proceed.
<code>with openmp("critical") :</code>	Mutual exclusion. One thread at a time executes code
<code>schedule(static [,chunk])</code>	Map blocks of loop iterations across the team. Use with for .
<code>reduction(op:list)</code>	Combine values with op across the team. Used with for
<code>private(list)</code>	Make a local copy of variables for each thread. Use with parallel , for or task .
<code>firstprivate(list)</code>	private , but initialize with original value. Use with parallel , for or task
<code>shared(list)</code>	Variables shared between threads. Use with parallel , for or task .
<code>default(none)</code>	Force definition of variables as private or shared .
<code>omp_get_num_threads()</code>	Return the number of threads in a team
<code>omp_get_thread_num()</code>	Return an ID from 0 to the number of threads minus one
<code>omp_set_num_threads(int)</code>	Set the number of threads to request for parallel regions
<code>omp_get_wtime()</code>	Return a snapshot of the wall clock time.

... the general form of common core GPU programming

Data env
Target Region

```

with openmp ("target data map(to: A,B) map(tofrom: C)"):
  with openmp ("target teams distribute parallel for thread_limit(256)"):
    for x in range(Nx):
      for y in range(Ny):
        # operations with vectors A and B with results written to C

```

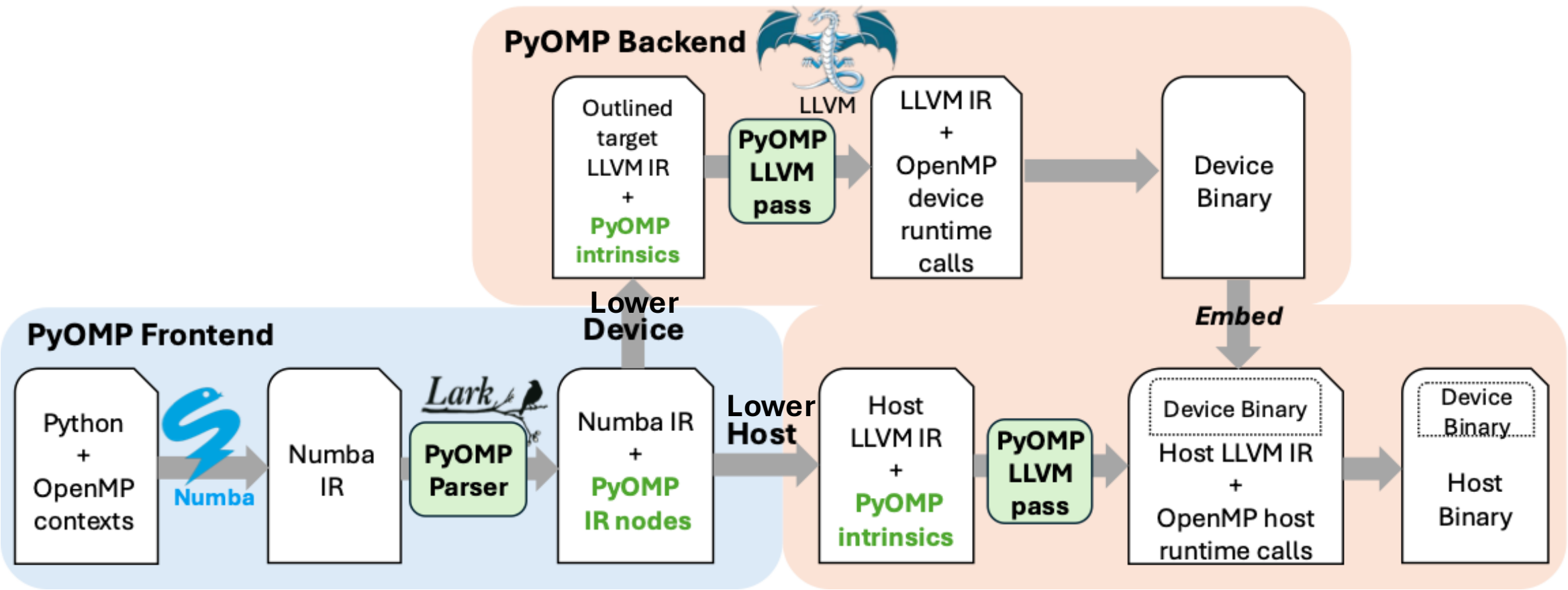
Construct/Clause	Description
target data	Create a data region on the device.
map(to:A,B)	Map arrays A and B to the device without copying back to host
map(tofrom:C)	Map array C to the device and copy back to the host
target teams	Offload to device and launch a league of teams.
distribute	Distribute loop iterations among the league of teams.
parallel for	Create a team of threads to execute loop iterations in parallel.
thread_limit(256)	Default number of threads (256) per team.

Pi on GPU

```
Imports { 1 from numba.openmp import njit
          2 from numba.openmp import openmp_context as openmp
          3 from numba.openmp import omp_get_thread_num
          4
          5 @njit
          6 def calc_pi(num_steps):
          7     step = 1.0/num_steps
          8     red_sum = 0.0
          9     with openmp("target map(tofrom: red_sum)"):
          10         with openmp("loop private(x) reduction(+:red_sum)"):
          11             for i in range(num_steps):
          12                 x = (i+0.5)*step
          13                 red_sum += 4.0 / (1.0 + x*x)
          14
          15     pi = step * red_sum
          16     return pi
          17
          18 print("pi =", calc_pi(1000000))
          19
```

Target Region & Data env { Loop {

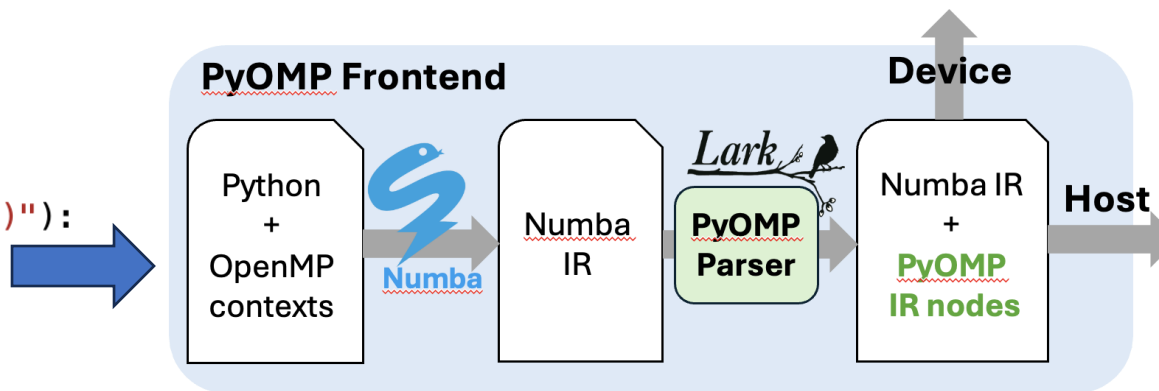
PyOMP extends Numba and bases on LLVM to support both CPU/GPU parallel programming



@njit

```
def inc(x):
    with openmp("target teams "
                "distribute parallel for map(tofrom:x)"):
        for i in range(len(x)):
            x[i] = x[i] + 1

    return x
```



Numba IR + PyOMP nodes

Directive

label 32.1:

```
openmp_region_start
openmp_tag(DIR.OMP.TARGET.TEAMS.DISTRIBUTE.PARALLEL.LOOP,0),
openmp_tag(QUAL.OMP.NUM_TEAMS,0), openmp_tag(QUAL.OMP.THREAD_LIMIT,0),
openmp_tag(QUAL.OMP.MAP.TOFROM,x), openmp_tag(QUAL.OMP.PRIVATE,i),
openmp_tag(QUAL.OMP.MAP.TOFROM.STRUCT,x*data, omp_slice(0,x_num_elements_var00)),
openmp_tag(QUAL.OMP.MAP.TO.STRUCT,x*shape, omp_slice(0,1)),
openmp_tag(QUAL.OMP.MAP.TO.STRUCT,x*strides, omp_slice(0,1)),
openmp_tag(OMP.DEVICE,None)
```

Region
markers

...

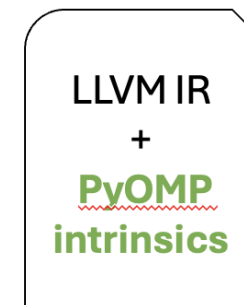
<region body>

label 126:

```
openmp_region_end openmp_tag(DIR.OMP.END.TARGET.TEAMS.DISTRIBUTE.PARALLEL.LOOP,0) []
```

Array metadata

PyOMP lowers the extended Numba IR to LLVM IR with PyOMP intrinsics



LLVM IR + PyOMP intrinsics

Intrinsics markers

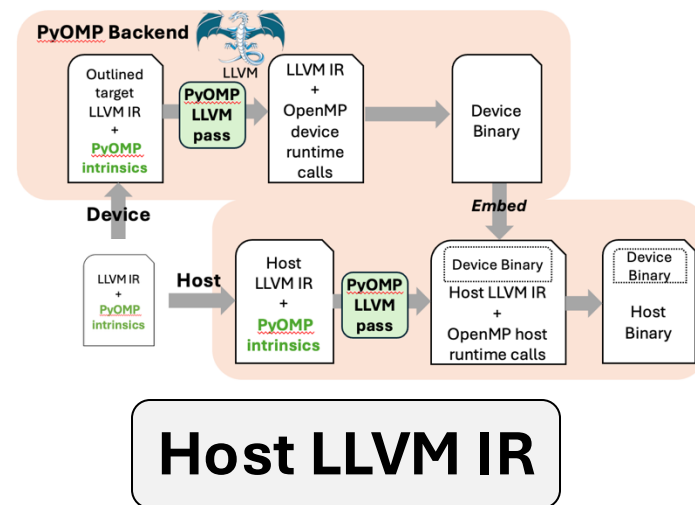
Directive & clause qualifiers
operand bundles

```

B32.1:
%50 = call token @llvm.directive.region.entry() [
  "DIR.OMP.TARGET.TEAMS.DISTRIBUTE.PARALLEL.LOOP"(i32 0),
  "QUAL.OMP.NUM_TEAMS"(i32 0), "QUAL.OMP.THREAD_LIMIT"(i32 0),
  "QUAL.OMP.MAP.TOFROM"({ i8*, i8*, i64, i64, double*, [1 x i64], [1 x i64] }* %arg.x),
  "QUAL.OMP.PRIVATE"(i64* %i),
  "QUAL.OMP.MAP.TOFROM.STRUCT"({ i8*, %i8*, i64, i64, double*, [1 x i64], [1 x i64] }* %arg.x, i32 4, i64 0, i64 %%.47),
  "QUAL.OMP.MAP.TO.STRUCT"({ i8*, i8*, i64, i64, double*, [1 x i64], [1 x i64] }* %arg.x, i32 5, i64 0, i64 1),
  "QUAL.OMP.MAP.TO.STRUCT"({ i8*, i8*, %i64, i64, double*, [1 x i64], [1 x i64] }* %arg.x, i32 6, i64 0, i64 1),
  %"OMP.DEVICE"() ]
br label %B92
...
<region body>
...
B126:
tail call void @llvm.directive.region.exit(token %50) [ "DIR.OMP.END.TARGET.TEAMS.DISTRIBUTE.PARALLEL.LOOP"(i32 0) ]

```

Split compilation with the PyOMP LLVM pass lowers intrinsics to the OpenMP API



Device LLVM IR

Kernel entry point

```
define void @__omp_offload_numba_uid(..., numba.array.ty* %arg.x, ...) {
  ...
  %3 = call i32 @__kmpc_target_init(...)
  %4 = call i32 @device_func(...)
  call void @__kmpc_target_deinit(...)
  ...
}

define internal i32 @device_func(..., numba.array.ty* %arg.x, ...) {
  call void @device.teams(...)
}

define internal void @device.teams(..., numba.array.ty* %arg.x, ...) {
  call void @__kmpc_distribute_static_init_8u(...)
  ...
  call void @__kmpc_parallel_51(..., device.parallel, ...)
  ...
  call void @__kmpc_distribute_static_fini(...)
}

define internal void @device.parallel(..., numba.array.ty* %arg.x, ...) {
  call void @__kmpc_for_static_init_8u(...)
  ...
  call void @__kmpc_for_static_fini(...)
}
```

OpenMP runtime API

C
O
M
P
I
L
E

Host LLVM IR

```
@.omp_offloading.entry = ...
@.omp_offloading.entries = ...
@.omp_offloading.device_image = internal constant [14128 x i8] c"<binary image>"
@.omp_offloading.device_images = ...
@.omp_offloading.descriptor = { <images>, <entries> }

define i32 @ZN8__main_inc(...) {
  ...
  call void @__kmpc_push_target_tripcount_mapper(..., i64 %tripcount)
  %54 = call i32 @__tgt_target_teams_mapper(...i, i8* @__omp_offload_numba_uid, ...)
  ...
}

define internal void @.omp_offloading.descriptor_reg() section ".text.startup" {
entry:
  call void @__tgt_register_lib(%struct.__tgt_bin_desc* @.omp_offloading.descriptor)
  ret void
}

define internal void @.omp_offloading.descriptor_unreg() section ".text.startup" {
entry:
  call void @__tgt_unregister_lib(%struct.__tgt_bin_desc* @.omp_offloading.descriptor)
  ret void
}
```

Evaluating performance

Georgakoudis, G., Anderson, T.A., Archibald, S., Supinski, B.R.d., Mattson, T.G. (2026). Programming GPUs with OpenMP and Python. In: Yan, Y., Klemm, M., de Supinski, B.R., Saule, E., Klinkenberg, J., Pophale, S. (eds) OpenMP: Balancing Productivity and Performance Portability. IWOMP 2025. Lecture Notes in Computer Science, vol 16123. Springer, Cham. https://doi.org/10.1007/978-3-032-06343-4_14

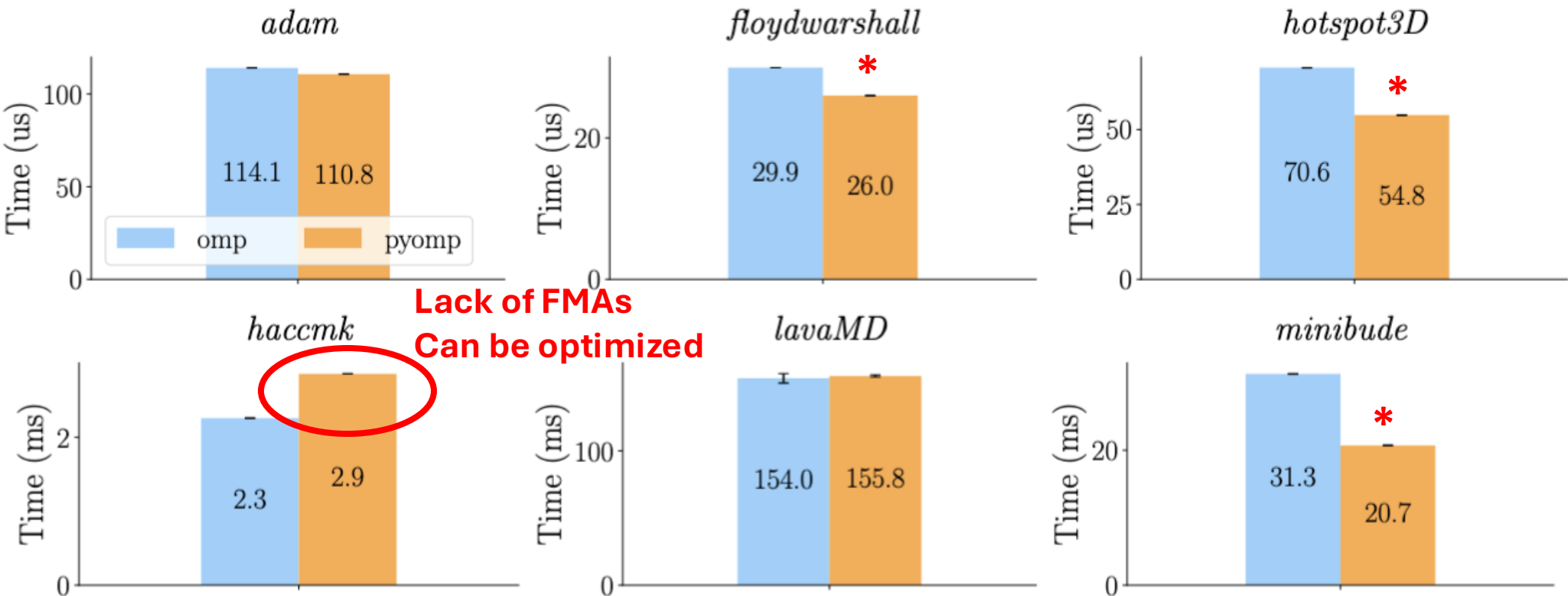
- HeCBench benchmarks
 - C/C++ OpenMP offload
 - PyOMP
- Hardware
 - AMD EPYC 7763 CPU + NVIDIA A100 GPU with 80 GB of memory
- Software
 - Python 3.9
 - Numba 0.57.1
 - Clang/LLVM 14.0.6
 - CUDA 12.2

- Metrics

- Kernel execution time
- Data transfer size, time
- Compilation time

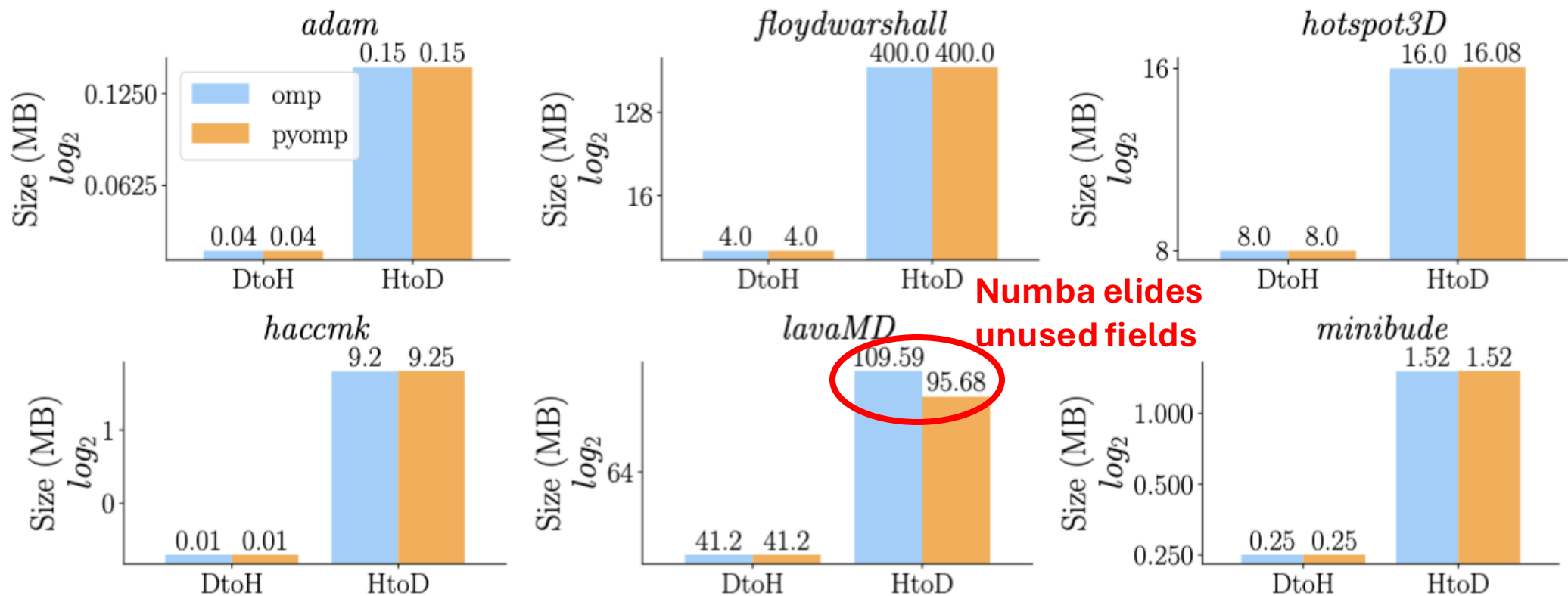
Program	Description
<i>adam</i>	ML
<i>floydwarshall</i>	Graph analysis
<i>haccmk</i>	Cosmology
<i>hotspot3D</i>	Thermal stencil
<i>lavaMD</i>	Molecular dynamics
<i>miniBUDE</i>	Protein docking

Kernel execution time: PyOMP is competitive with the C/C++ version

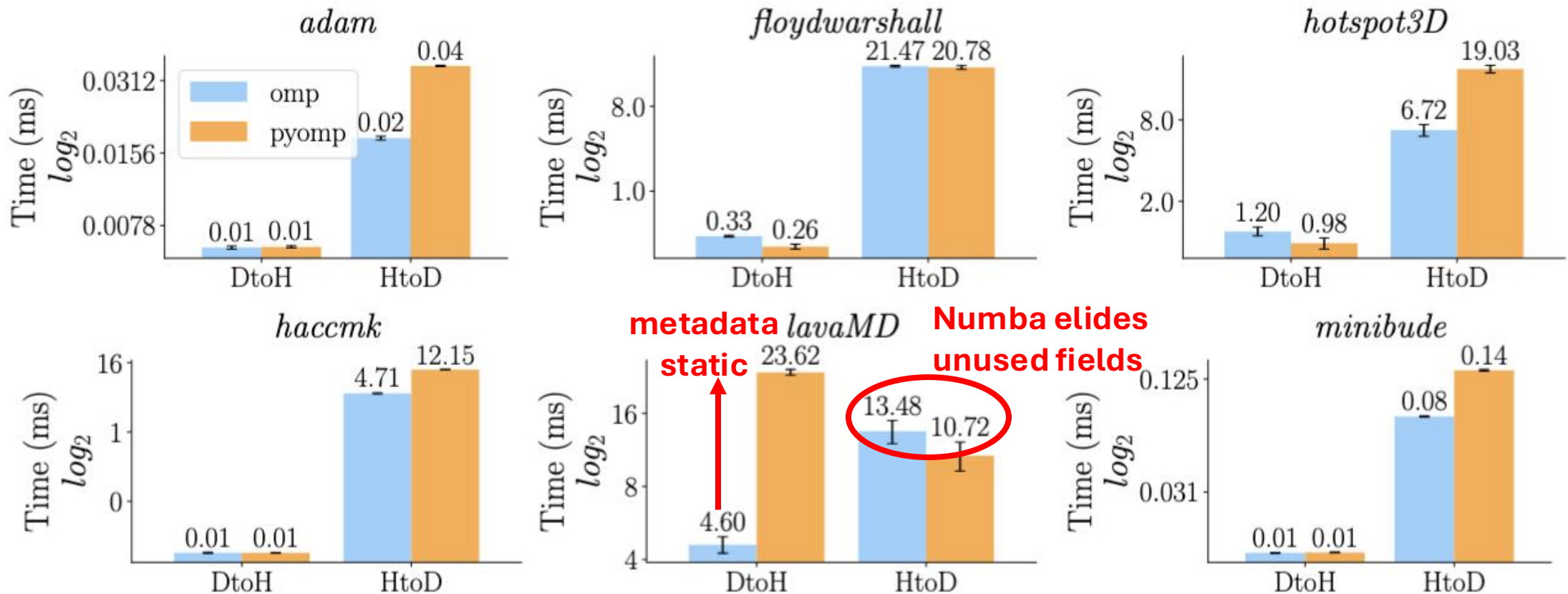


***Toolchain differences, we expect on par performance with LTO, LLVM upgrades**

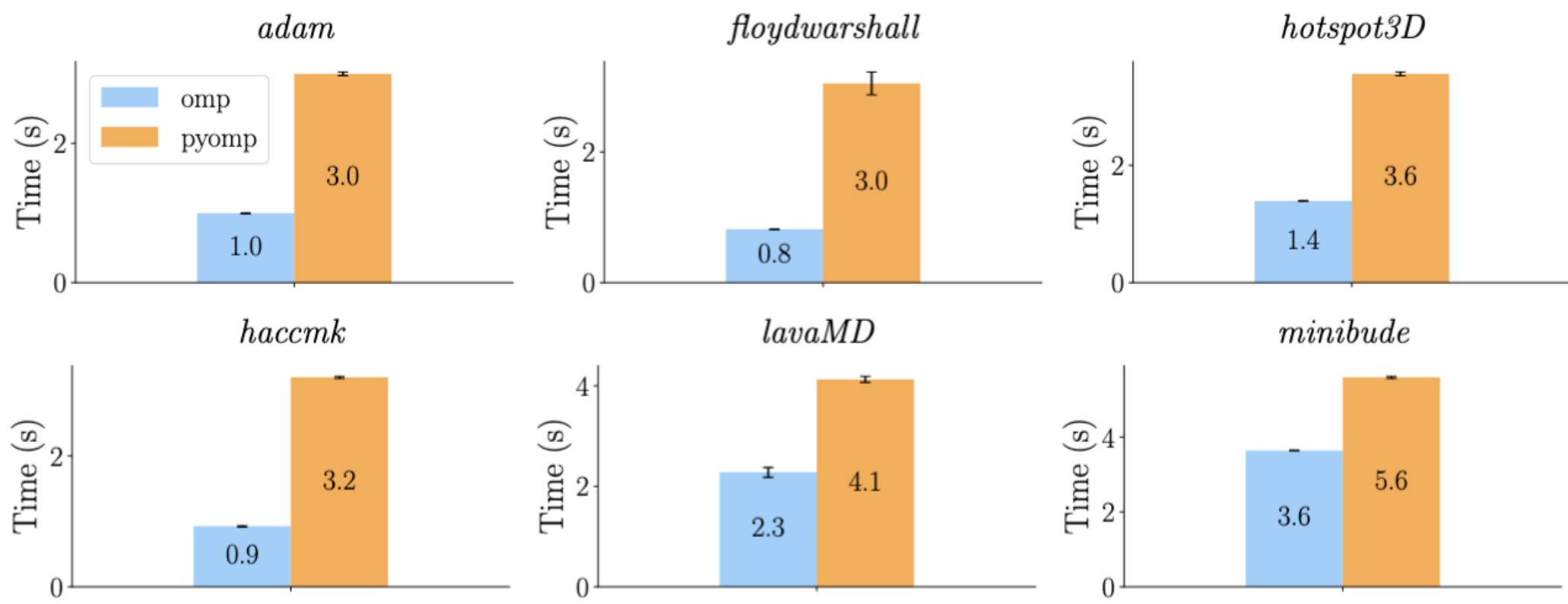
Data transfer size: PyOMP is comparable with the C/C++ version



Data transfer time: PyOMP is slower but data are re-used in compute (also impl. can be optimized)



Compilation: PyOMP JIT compilation (first time) is slower than AOT C/C++ but caching mitigates



PyOMP is open source and available to use

- <https://github.com/Python-for-HPC/PyOMP>



- Installation
 - PyPI: `pip install pyomp`
 - Conda: `conda install -c python-for-hpc -c conda-forge pyomp`

We  Users!

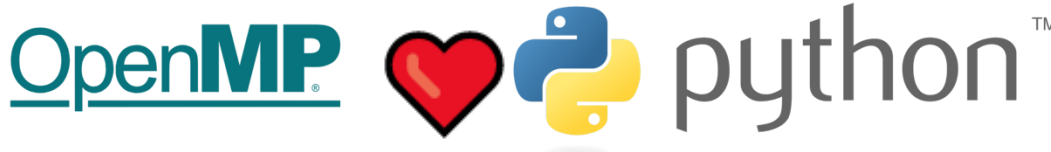
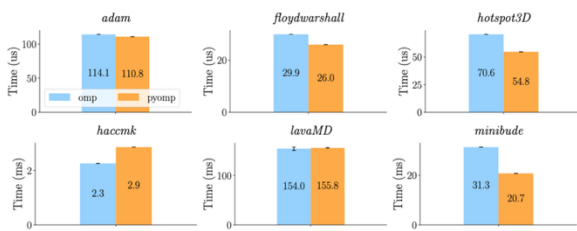
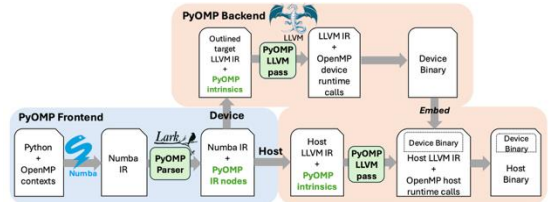
Conclusion

- PyOMP brings OpenMP to Python
- PyOMP is implemented on top of Numba and LLVM
- Competitive performance vs. C/C++ OpenMP
- We are working in the new OpenMP Python subcommittee to make Python officially supported by OpenMP

<code>with openmp("parallel"):</code>	Create a team of threads. Execute a parallel region
<code>with openmp("for"):</code>	Use inside a parallel region. Split up a loop across the team.
<code>with openmp("parallel for"):</code>	A combined construct. Same as <code>parallel</code> followed by a <code>for</code> .
<code>with openmp("single"):</code>	One thread does the work. Others wait for it to finish
<code>with openmp("task"):</code>	Create an explicit task for work within the construct.
<code>with openmp("taskwait"):</code>	Wait for all tasks in the current task to complete.
<code>with openmp("barrier"):</code>	All threads arrive at a barrier before any proceed.
<code>with openmp("critical"):</code>	Mutual exclusion. One thread at a time executes code
<code>schedule(static, 1, chunk)</code>	Map blocks of loop iterations across the team. Use with <code>for</code> .
<code>reduction(op: list)</code>	Combine values with <code>op</code> across the team. Used with <code>for</code> .
<code>private(list)</code>	Make a local copy of variables for each thread. Use with <code>parallel</code> , <code>for</code> or <code>task</code> .
<code>firstprivate(list)</code>	private, but initialize with original value. Use with <code>parallel</code> , <code>for</code> or <code>task</code> .
<code>shared(list)</code>	Variables shared between threads. Use with <code>parallel</code> , <code>for</code> or <code>task</code> .
<code>default(none)</code>	Force definition of variables as <code>private</code> or <code>shared</code> .
<code>omp_get_num_threads()</code>	Return the number of threads in a team
<code>omp_get_thread_num()</code>	Return an ID from 0 to the number of threads minus one
<code>omp_get_num_threads(int)</code>	Set the number of threads to request for parallel regions
<code>omp_get_wtime()</code>	Return a snapshot of the wall clock time.

```
with openmp("target data map(to: A,B) map(tofrom: C)"):
    with openmp("target teams distribute parallel for thread_limit(256)"):
        for x in range(Nx):
            for y in range(Ny):
                # operations with vectors A and B with results written to C
```

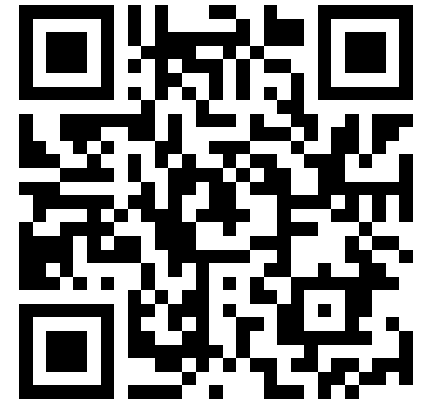
Construct/Clause	Description
<code>target data</code>	Create a data region on the device.
<code>map(to:A,B)</code>	Map arrays A and B to the device without copying back to host
<code>map(tofrom:C)</code>	Map array C to the device and copy back to the host
<code>target teams</code>	Offload to device and launch a league of teams.
<code>distribute</code>	Distribute loop iterations among the league of teams.
<code>parallel for</code>	Create a team of threads to execute loop iterations in parallel.
<code>thread_limit(256)</code>	Default number of threads (256) per team.



Thank you!

georgakoudis1@llnl.gov

<https://github.com/Python-for-HPC/PyOMP>





SC25 OpenMP Tech Talk

openmp.org

OpenMP API specs, forum,
reference guides,

and more

link.openmp.org/sc25talks

SC25 OpenMP Tech Talk
videos

and presentations