LLVM/OpenMP

A Brief Update

WITH J🤔HANNES

Why LLVM/Clang/Flang (for HPC)?

- open (source/community/...)

- extensible, "fixable"
- portable (GPUs😔, CPUs, ...) [😔 LLVM 13]
- OpenMP/C++/... feature complete😉 [😉 eventually]
- early access to *the coolest* features

- performant and correct ;)

LOT'S OF CONTENT, NO TIME, USE THE SLIDES AND GET IN TOUCH!

LLVM/OpenMP - A Community Effort

Weekly Meeting: https://bit.ly/2Zqt49v

"Academia"

★ Joseph Huber (ORNL)
★ Shilei Tian (SBU)
★ Giorgis Georgakoudis (LLNL)
★ Michael Kruse (ANL)
★ Joachim Protze (RWTH A.)
★ Joel Denny (ORNL)
★ Valentin Clement (ORNL, now NVIDIA)
★ Many, many, more

Industry

★ Alexey Bataev (Intel)
★ Jon Chesterfield (AMD)
★ George Rokos (Intel)
★ Pushpinder Singh (AMD)
★ Kiran Chandramohan (ARM)
★ Chi Chun Chen (HPE/Cray)
★ Andrey Churbanov (Intel)
★ Carlo Bertolli (AMD)
★ Many, many, more

Power Users

★ Ye Luo (ANL)
★ Christopher Daley (NERSC)
★ John Tramm (ANL)
★ Rahul Gayatri (NERSC)
★ Itaru Kitayama (RIKEN)
★ Wael Elwasif (ORNL)
★ More that I have forgotten

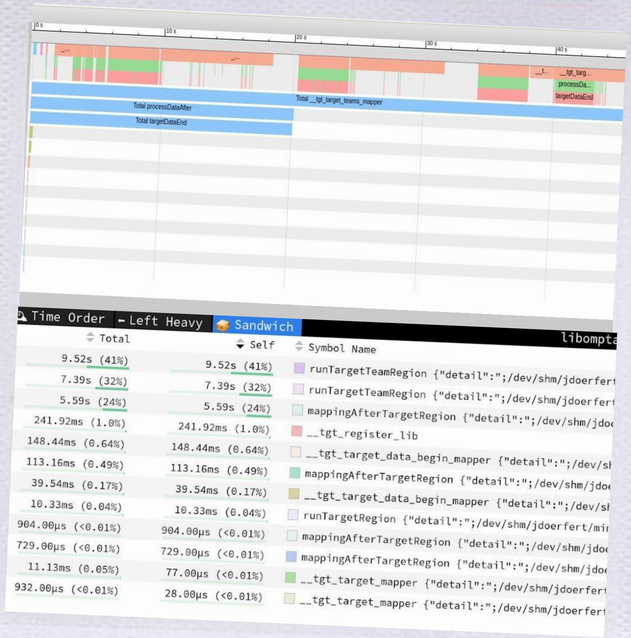
Simple Profiling Support (LLVM 12)

Use

LIBOMPTARGET_PROFILE=file.json

to portably profile target interactions.

Chrome tracing format, source line information, ...



Total	Self	Symbol Name
9.52s (41%)	9.52s (41%)	runTargetTeamRegion {"detail":";/dev/shm/jdoerfert
7.39s (32%)	7.39s (32%)	runTargetTeamRegion {"detail":";/dev/shm/jdoerfert
5.59s (24%)	5.59s (24%)	mappingAfterTargetRegion {"detail":";/dev/shm/jdoe
241.92ms (1.0%)	241.92ms (1.0%)	__tgt_register_lib
148.44ms (0.64%)	148.44ms (0.64%)	__tgt_target_data_begin_mapper {"detail":";/dev/sh
113.16ms (0.49%)	113.16ms (0.49%)	mappingAfterTargetRegion {"detail":";/dev/shm/jdoe
39.54ms (0.17%)	39.54ms (0.17%)	__tgt_target_data_begin_mapper {"detail":";/dev/sh
10.33ms (0.04%)	10.33ms (0.04%)	runTargetRegion {"detail":";/dev/shm/jdoerfert/mir
904.00μs (<0.01%)	904.00μs (<0.01%)	mappingAfterTargetRegion {"detail":";/dev/shm/jdoe
729.00μs (<0.01%)	729.00μs (<0.01%)	mappingAfterTargetRegion {"detail":";/dev/shm/jdoe
11.13ms (0.05%)	77.00μs (<0.01%)	__tgt_target_mapper {"detail":";/dev/shm/jdoerfert
932.00μs (<0.01%)	28.00μs (<0.01%)	__tgt_target_mapper {"detail":";/dev/shm/jdoerfert

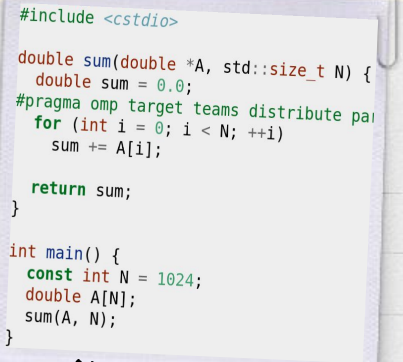
Profiling OpenMP offload with LIBOMPTARGET_PROFILE

https://openmp.llvm.org/design/Runtimes.html#libomptarget-profile

Debugging OpenMP (LLVM 12+)

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 -O3 -gline-tables-only sum.cpp -o sum
$ ./sum
```

```
CUDA error: Error when copying data from device to host.
CUDA error: an illegal memory access was encountered
Libomptarget error: Copying data from device failed.
Libomptarget error: Call to targetDataEnd failed, abort target.
Libomptarget error: Failed to process data after launching the kernel.
Libomptarget error: Run with LIBOMPTARGET_INFO=4 to dump host-target pointer mappings.
sum.cpp:5:1: Libomptarget error 1: failure of target construct while offloading is mandatory
```



```
#include <cstdio>

double sum(double *A, std::size_t N) {
  double sum = 0.0;
#pragma omp target teams distribute pa
  for (int i = 0; i < N; ++i)
    sum += A[i];

  return sum;
}

int main() {
  const int N = 1024;
  double A[N];
  sum(A, N);
}
```

My First OpenMP Offload Program

Debugging OpenMP (LLVM 12+)

LLVM 12 introduced

LIBOMPTARGET_INFO=<bitfield>

to portably and reliably debug offloading.

Supports OpenMP runtime debug messages as well as "plugin" debug messages.

Available in release mode!

https://openmp.llvm.org/design/Runtimes.html#libomptarget-info

IMPORTANT!

VISIT

openmp.llvm.org

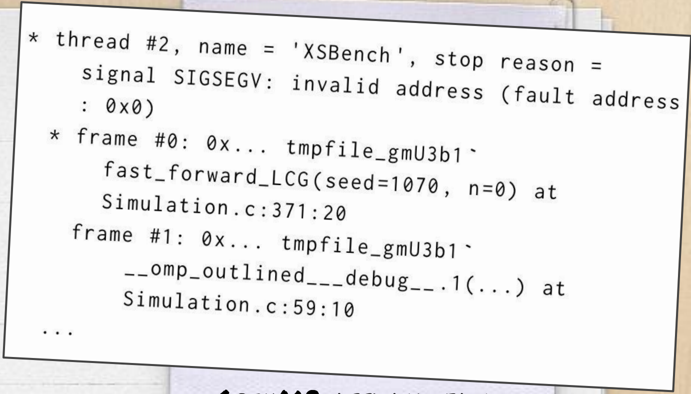
REGULARLY.

FEEL FREE TO SUGGEST CONTENT + FAQ ENTRIES!

Debugging OpenMP (LLVM 14)

A plugin to offload to a virtualized GPU (VGPU).

Device compilation and runtime executed on the host
=> host tooling (gdb, sanitizers, ...) works natively!



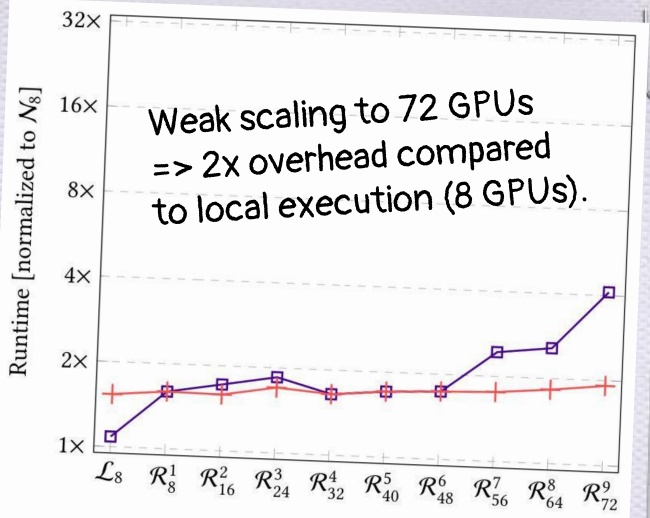
```
* thread #2, name = 'XSBench', stop reason =
    signal SIGSEGV: invalid address (fault address
    : 0x0)
  * frame #0: 0x... tmpfile_gmU3b1`
      fast_forward_LCG(seed=1070, n=0) at
      Simulation.c:371:20
    frame #1: 0x... tmpfile_gmU3b1`
      __omp_outlined___debug__.1(...) at
      Simulation.c:59:10
 ...
```

OpenMP offload to a virtual GPU, ... cool

Remote OpenMP offloading (LLVM 12+)

Utilize remote GPUs (and CPUs) as if they were local.

Also allows to debug memory mapping errors on a single host!



RSBench remote offloading performance

https://openmp.llvm.org/design/Runtimes.html#llvm-openmp-target-host-runtime-plugins-libomptarget-rtl-xxxx

OpenMP-Aware Optimizations (LLVM 12+)

Towards OpenMP-aware compiler optimizations

- LLVM "knows" about OpenMP API and (internal) runtime calls, incl. their potential effects (e.g., they won't throw exceptions)
- LLVM performs "high-level" optimizations, e.g., parallel region merging, and various GPU-specific optimizations late
- Some LLVM/Clang "optimizations" remain, but we are in the process of removing them: simple frontend, smart middle-end

interprocedural optimizations for host & device

run with -O2 and -O3 since LLVM 11 (-O1 with LLVM 13)

OpenMP-Opt

OpenMP-Aware Optimizations

Automatic SPMDzation + shared memory usage (LLVM 13+)

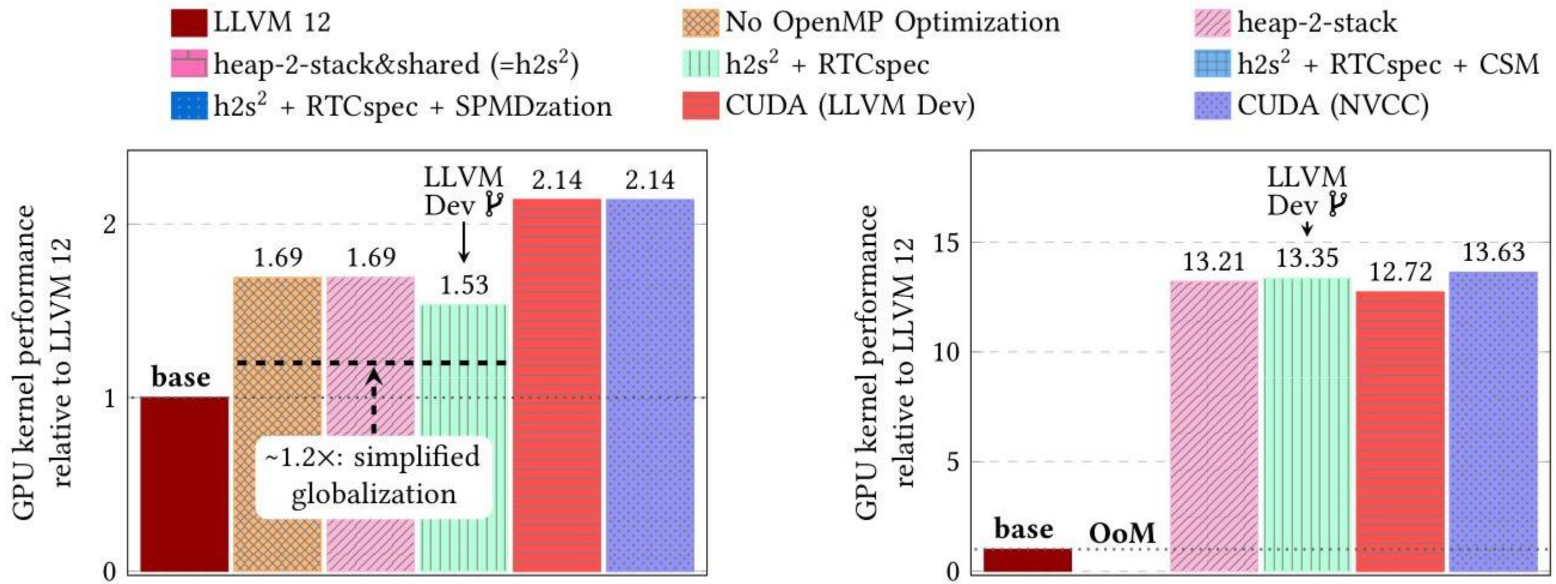
```
#pragma omp target teams
{
   double team_local_memory[M];
   team_main_thread_only();
   #pragma omp parallel
   every_thread(team_local_memory);
}
```

SPMDzation - “CUDA”-like execution mode

Shared memory usage for scratchpads

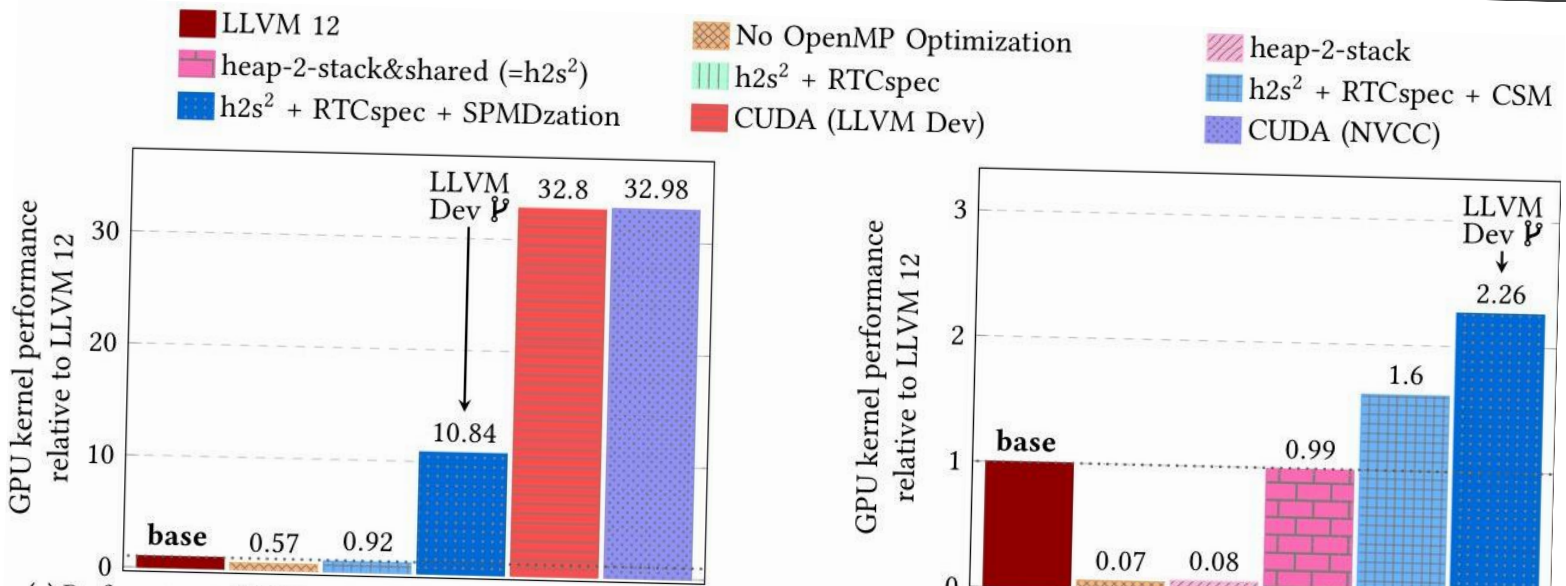
Automatic guarding and synchronization

```
#pragma omp target teams
#pragma omp parallel
{
  double team_local_memory[M];
  #pragma omp allocate(team_local_memory) \
              allocator(omp_cgroup_mem_alloc)
  #pragma omp masked
  team_main_thread_only();
  #pragma omp barrier
  every_thread(team_local_memory);
}
```

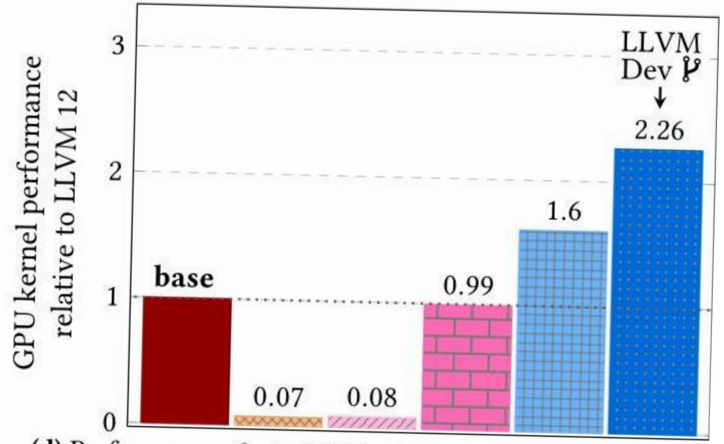


(a) Performance of XSBench relative to LLVM 12 (base).

(b) Performance of RSBench relative to LLVM 12 (base).



(c) Performance of SU3Bench relative to LLVM 12 (base).

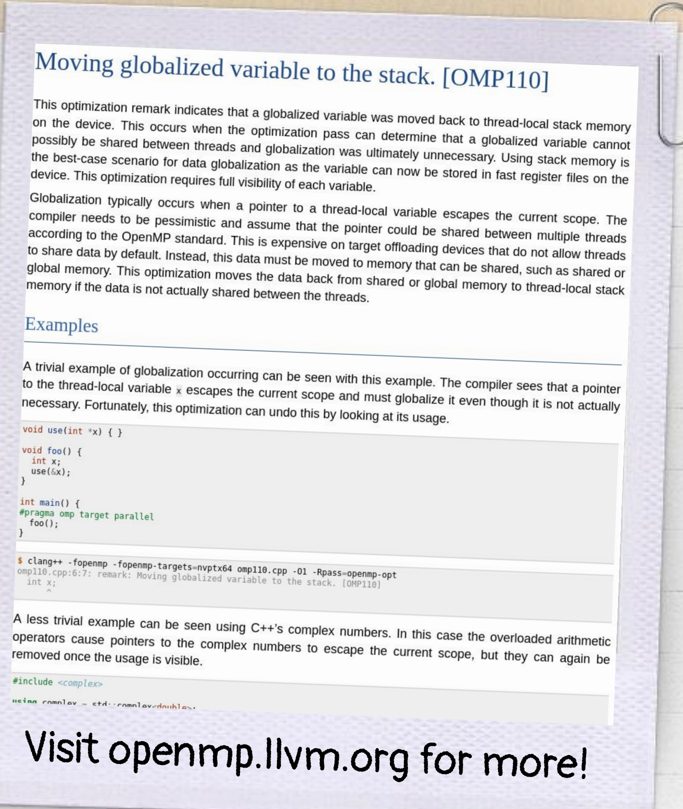


(d) Performance of miniQMC relative to LLVM 12 (base).

OpenMP-Optimization Remarks & Assumptions

```
example.cpp:41:24: remark: Found thread data sharing on the
     GPU. Expect degraded performance due to data
     globalization. [OMP112] [-Rpass-missed=openmp-opt]
double device_function(float Arg) {
                             ^
example.cpp:42:3: remark: Moving globalized variable to the
     stack. [OMP110] [-Rpass=openmp-opt]
double Lcl;
       ^
```

1) OpenMP-Opt emits remarks (above)
2) The web provides explanations (right)
3) Users add OpenMP assumptions, e.g.,
   `#pragma omp assume ext_spmd_amenable`



Moving globalized variable to the stack. [OMP110]

This optimization remark indicates that a globalized variable was moved back to thread-local stack memory on the device. This occurs when the optimization pass can determine that a globalized variable cannot possibly be shared between threads and globalization was ultimately unnecessary. Using stack memory is the best-case scenario for data globalization as the variable can now be stored in fast register files on the device. This optimization requires full visibility of each variable.

Globalization typically occurs when a pointer to a thread-local variable escapes the current scope. The compiler needs to be pessimistic and assume that the pointer could be shared between multiple threads according to the OpenMP standard. This is expensive on target offloading devices that do not allow threads to share data by default. Instead, this data must be moved to memory that can be shared, such as shared or global memory. This optimization moves the data back from shared or global memory to thread-local stack memory if the data is not actually shared between the threads.

Examples

A trivial example of globalization occurring can be seen with this example. The compiler sees that a pointer to the thread-local variable `x` escapes the current scope and must globalize it even though it is not actually necessary. Fortunately, this optimization can undo this by looking at its usage.

```
void use(int *x) { }

void foo() {
  int x;
  use(&x);
}

int main() {
#pragma omp target parallel
  foo();
}
```

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 omp110.cpp -O1 -Rpass=openmp-opt
omp110.cpp:6:7: remark: Moving globalized variable to the stack. [OMP110]
  int x;
      ^
```

A less trivial example can be seen using C++'s complex numbers. In this case the overloaded arithmetic operators cause pointers to the complex numbers to escape the current scope, but they can again be removed once the usage is visible.

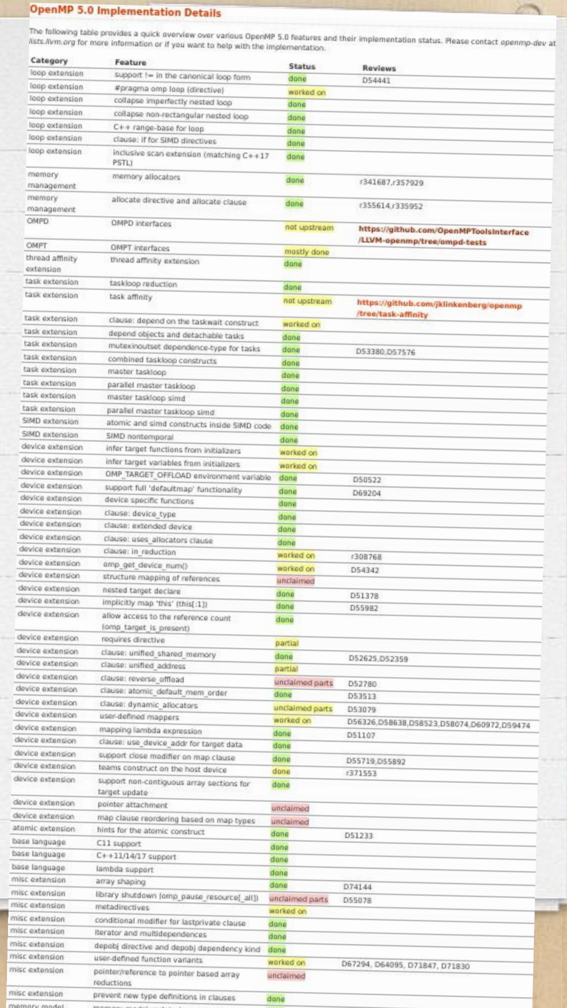
```
#include <complex>

using complex = std::complex<double>;
```

Visit openmp.llvm.org for more!

https://openmp.llvm.org/remarks/OptimizationRemarks.html

Near Future Development

- ★ Finishing last OpenMP 5.0 features (right)
- ★ Continue to work on OpenMP 5.1 features
- ★ Harden the AMD GPU offloading (LLVM 13+)
- ★ Enable the new GPU device runtime by default
  - ○ "SIMD" support for the GPU
  - ○ Memory and runtime overhead only for used features
  - ○ Better diagnostics, assertions, etc.
- ★ Proper linking support for device code
  - ○ Including Link-Time-Optimizations (LTO)!
- ★ Just-In-Time (JIT) compilation for device code
- ★ Many other cool things 😉, get involved!



OpenMP 5.0 Implementation Details

The following table provides a quick overview over various OpenMP 5.0 features and their implementation status. Please contact *openmp-dev* at *lists.llvm.org* for more information or if you want to help with the implementation.

Category	Feature	Status	Reviews
loop extension	support != in the canonical loop form	done	D54441
loop extension	#pragma omp loop (directive)	worked on	
loop extension	collapse imperfectly nested loop	done	
loop extension	collapse non-rectangular nested loop	done	
loop extension	C++ range-base for loop	done	
loop extension	clause: if for SIMD directives	done	
loop extension	inclusive scan extension (matching C++17 PSTL)	done	
memory management	memory allocators	done	r341687,r357929
memory management	allocate directive and allocate clause	done	r355614,r335952
OMPD	OMPD interfaces	not upstream	https://github.com/OpenMPToolsInterface/LLVM-openmp/tree/ompd-tests
OMPT	OMPT interfaces	mostly done	
thread affinity extension	thread affinity extension	done	
task extension	taskloop reduction	done	
task extension	task affinity	not upstream	https://github.com/jklinkenberg/openmp/tree/task-affinity
task extension	clause: depend on the taskwait construct	worked on	
task extension	depend objects and detachable tasks	done	
task extension	mutexinoutset dependence-type for tasks	done	D53380,D57576
task extension	combined taskloop constructs	done	
task extension	master taskloop	done	
task extension	parallel master taskloop	done	
task extension	master taskloop simd	done	
task extension	parallel master taskloop simd	done	
SIMD extension	atomic and simd constructs inside SIMD code	done	
SIMD extension	SIMD nontemporal	done	
device extension	infer target functions from initializers	worked on	
device extension	infer target variables from initializers	worked on	
device extension	OMP_TARGET_OFFLOAD environment variable	done	D50522
device extension	support full 'defaultmap' functionality	done	D69204
device extension	device specific functions	done	
device extension	clause: device_type	done	
device extension	clause: extended device	done	
device extension	clause: uses_allocators clause	done	
device extension	clause: in_reduction	worked on	r308768
device extension	omp_get_device_num()	worked on	D54342
device extension	structure mapping of references	unclaimed	
device extension	nested target declare	done	D51378
device extension	implicitly map 'this' (this[:1])	done	D55982
device extension	allow access to the reference count (omp_target_is_present)	done	
device extension	requires directive	partial	
device extension	clause: unified_shared_memory	done	D52625,D52359
device extension	clause: unified_address	partial	
device extension	clause: reverse_offload	unclaimed parts	D52780
device extension	clause: atomic_default_mem_order	done	D53513
device extension	clause: dynamic_allocators	unclaimed parts	D53079
device extension	user-defined mappers	worked on	D56326,D58638,D58523,D58074,D60972,D59474
device extension	mapping lambda expression	done	D51107
device extension	clause: use_device_addr for target data	done	
device extension	support close modifier on map clause	done	D55719,D55892
device extension	teams construct on the host device	done	r371553
device extension	support non-contiguous array sections for target update	done	
device extension	pointer attachment	unclaimed	
device extension	map clause reordering based on map types	unclaimed	
atomic extension	hints for the atomic construct	done	D51233
base language	C11 support	done	
base language	C++11/14/17 support	done	
base language	lambda support	done	
misc extension	array shaping	done	D74144
misc extension	library shutdown (omp_pause_resource[_all])	unclaimed parts	D55078
misc extension	metadirectives	worked on	
misc extension	conditional modifier for lastprivate clause	done	
misc extension	iterator and multidependences	done	
misc extension	depobj directive and depobj dependency kind	done	
misc extension	user-defined function variants	worked on	D67294, D64095, D71847, D71830
misc extension	pointer/reference to pointer based array reductions	unclaimed	
misc extension	prevent new type definitions in clauses	done	
memory model	[illegible]		

Cool, count me in, what next?

1) Get LLVM/Clang 13⚽ with offloading support ☕
2) clang++ -fopenmp -fopenmp-targets=nvptx64 ...
3) Check out https://openmp.llvm.org (FAQ!) and https://clang.llvm.org/docs/OpenMPSupport.html
4) Subscribe to https://llvm-gpu-news.github.io/
5) Talk to us! Join our meetings, report bugs, request cool features, ask questions, ... openmp-dev@llvm.lists.org

Thanks 2 quickmem.com

⚽ or even a recent development version from github!
☕ available on *all the cool* HPC machines

AMA

(Ask Me Anything)

The End

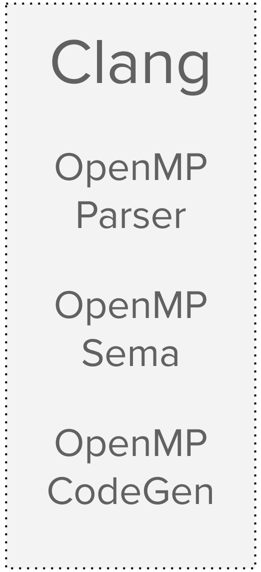
OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Johannes Doerfert
johannesdoerfert@gmail.com
Argonne National Lab

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v



OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

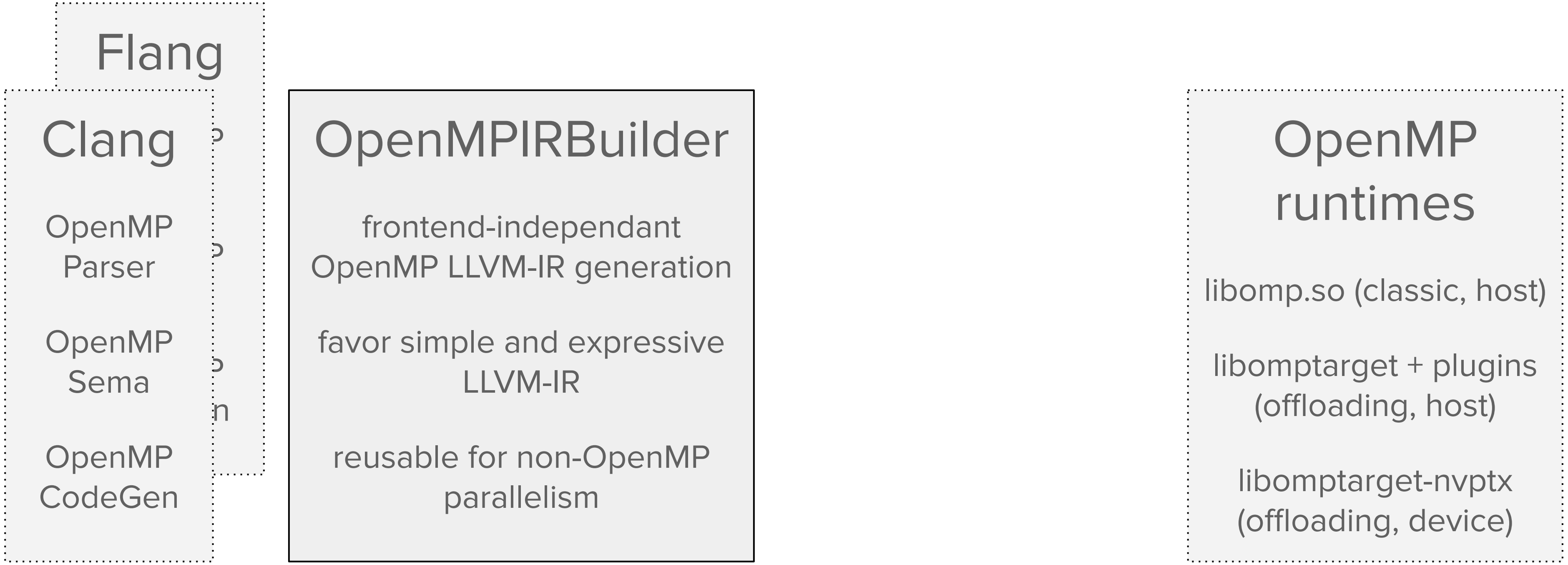
libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v


Flang
P
P
P
n
Clang
OpenMP Parser
OpenMP Sema
OpenMP CodeGen
OpenMPIRBuilder
frontend-independant OpenMP LLVM-IR generation
favor simple and expressive LLVM-IR
reusable for non-OpenMP parallelism
OpenMP runtimes
libomp.so (classic, host)
libomptarget + plugins (offloading, host)
libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMPIRBuilder

frontend-independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMPOpt

interprocedural optimization pass

contains host & device optimizations

run with `-O2` and `-O3` since LLVM 11

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)