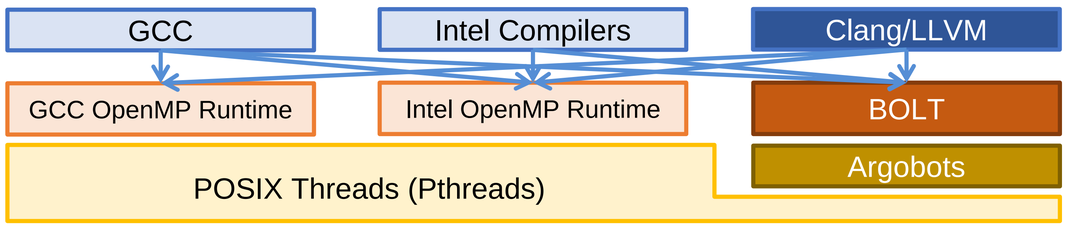
Exascale OpenMP Runtime

Part of the ECP Scaling OpenMP via LLVM for Exascale Performance and Portability (SOLLVE) Project

OpenMP Runtime Project Lead: Pavan Balaji, Argonne National Laboratory; SOLLVE Project PI: Barbara Chapman, Stony Brook University

BOLT: Lightning-Fast OpenMP Runtime

- BOLT is a recursive acronym: "BOLT is OpenMP over Lightweight Threads"
- OpenMP runtime that exploits lightweight threads and tasks
  - ***Won the Best Paper Award at PACT '19!!***
- Uses *Argobots* as the underlying threading and tasking mechanism
- Based on the LLVM OpenMP 9.0 runtime (http://openmp.llvm.org/)
  - Supports most of the OpenMP 4.5 features
- ABI compatibility with GCC, Intel compilers, and Clang/LLVM



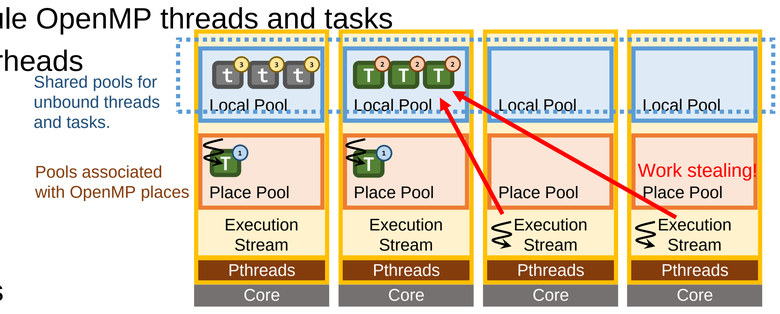
- **BOLT 1.0rc3 is available at http://www.bolt-omp.org**

```
# Install via Spack (only BOLT)
 $ spack install bolt
# Install via Spack (all SOLLVE packages)
 $ spack install sollve
# Install via Github
 $ git clone https://github.com/pmodels/bolt.git
 $ (CMake-based make & install)
```

- Key Institutions: Argonne National Laboratory, Universitat Jaume I, Barcelona Supercomputing Center, RIKEN AICS, the University of Tokyo
- Key contributors: Sangmin Seo (ANL), Halim Amer (ANL), Adrián Castelló (Universitat Jaume I), Phil Carns (ANL), and Shintaro Iwasaki (UTokyo, ANL)

Nested Parallelism over Argobots

- Uses Argobots lightweight threads as OpenMP threads and tasks
  - Execution streams schedule OpenMP threads and tasks
  - Tiny oversubscription overheads
- BOLT's scheduling model
  - Hierarchical affinity via `OMP_PROC_BIND`
  - User-level threads can be bound to specific cores by place pools
  - New affinity keyword `unset`: unset affinity and use random work stealing



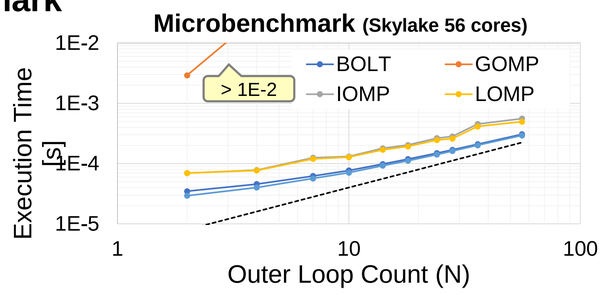

```
//OMP_PROC_BIND=close,unset
#pragma omp parallel for
for (int i = 0; i < 2; i++) {
  #pragma omp parallel for
  for (int j = 0; j < 3; j++) [...];
  #pragma omp taskloop
  for (int k = 0; k < 3; k++) [...];
}
```

① Threads (T) are put into place pools (close)
② Threads (T) are put into local pools (unset)
③ Tasks (t) are put into local pools

Nested Parallel Loop Microbenchmark

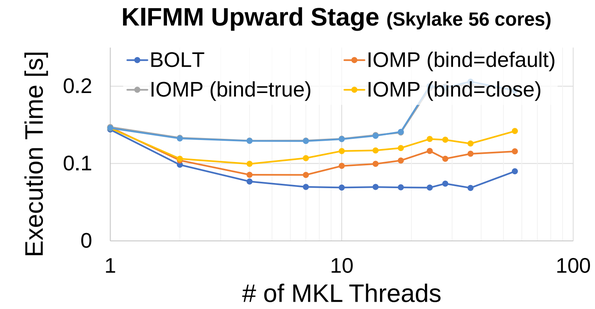
```
#pragma omp parallel for num_threads(N)
for (int i = 0; i < N; i++)
  #pragma omp parallel for num_threads(ncores/2)
  for (int j = 0; j < ncores/2; j++)
    comp(i, j); // 20000-cycle computation
```

- Nested parallel regions with different N.

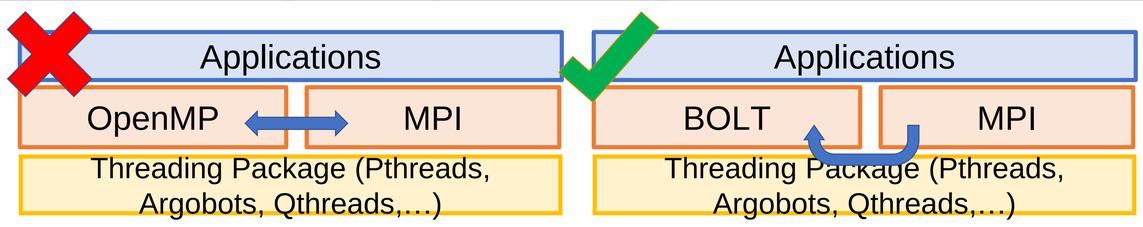


KIFMM with BLAS

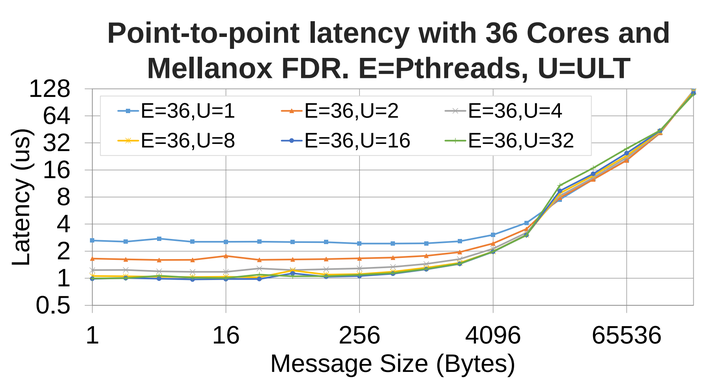
- Kernel-Independent FMM (KIFMM) with Intel MKL
  - OpenMP-parallelized `dgemv` is called in a parallel loop
  - Changed # of MKL threads



Interoperability Between OpenMP and MPI



- MPI Interoperation with Argobots allows
  - Better communication and computation overlapping
  - Latency hiding of blocking operations
- Supported by MPICH
  - Will be supported by Open MPI.



Custom Loop and Task Scheduling

User-defined Loop Scheduling (UDS)

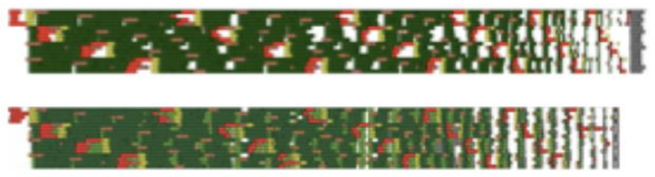
```
typedef my_type {...};
void my_init(...) {}; void my_next(...) {}; void my_fini(...) {};
size_t get_loopID(), get_usr_loopID(); void *get_usr_data(); // OpenMP RT provided functions.
#pragma omp declare schedule(my_sched) init(my_init) next(my_next) fini(my_fini) \
          {stackalloc(my_type)} [monotonic | non-monotonic] [usrLoopID(size_t)]

void example(int n) {
  int chunk = 4, user_val = 1337;
  #pragma omp parallel for schedule(my_sched, chunk, userdata(&user_val)) usrLoopID("IDstring")
  for (int i = 0; i < n; i++)
    compute(i);
}
```

UDS identifier | UDS parameters | UDS-related parameters

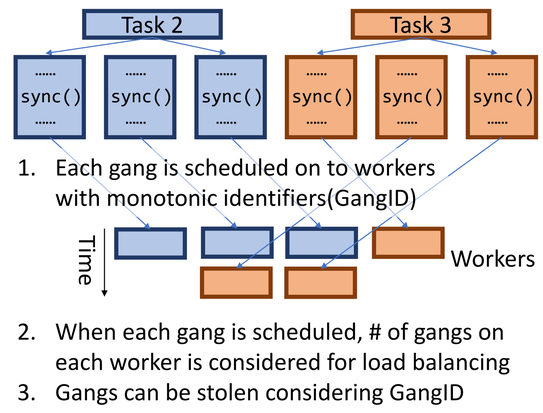
- The "declare schedule" directive is used to connect a schedule with a set of functions to initialize the schedule and hand out the next chunk of iterations.
  - The syntax of the "schedule" clause is extended to also accept an identifier of UDS.
- Instead of calling into the RTL for loop scheduling, the compiler will invoke the functions of the UDS.
- Visibility and namespaces of these identifiers will be borrowed from UDRs.

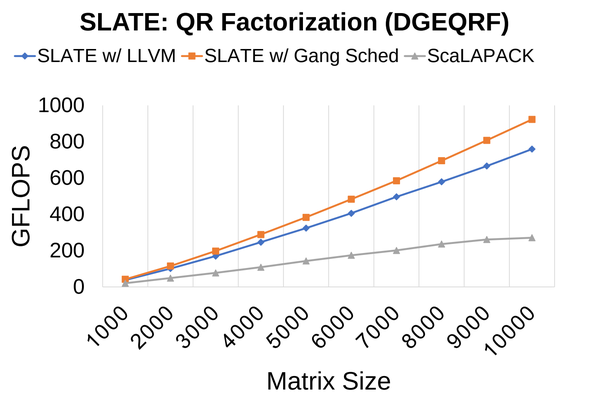
CALU using static scheduling (top) and fd=0.1 (bottom) with 2-level blocking layer run on AMD Opteron (16-core)



Task Scheduling

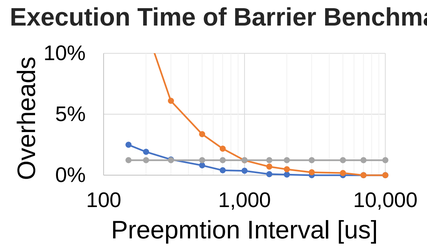
- Gang-scheduling for blocking tasks in task dependency graph of OpenMP
  - Adoption of gang-scheduling for tasks with blocking operations and data-parallelism in task-dependency graph
    → Data locality and deadlock-avoidance for user-level threads

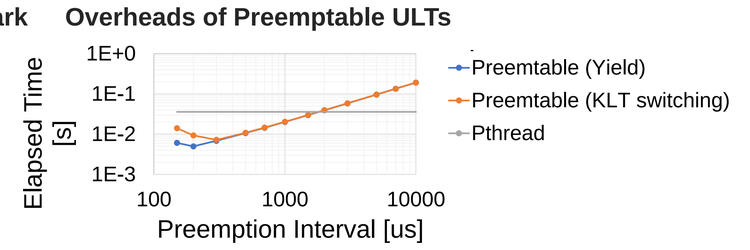




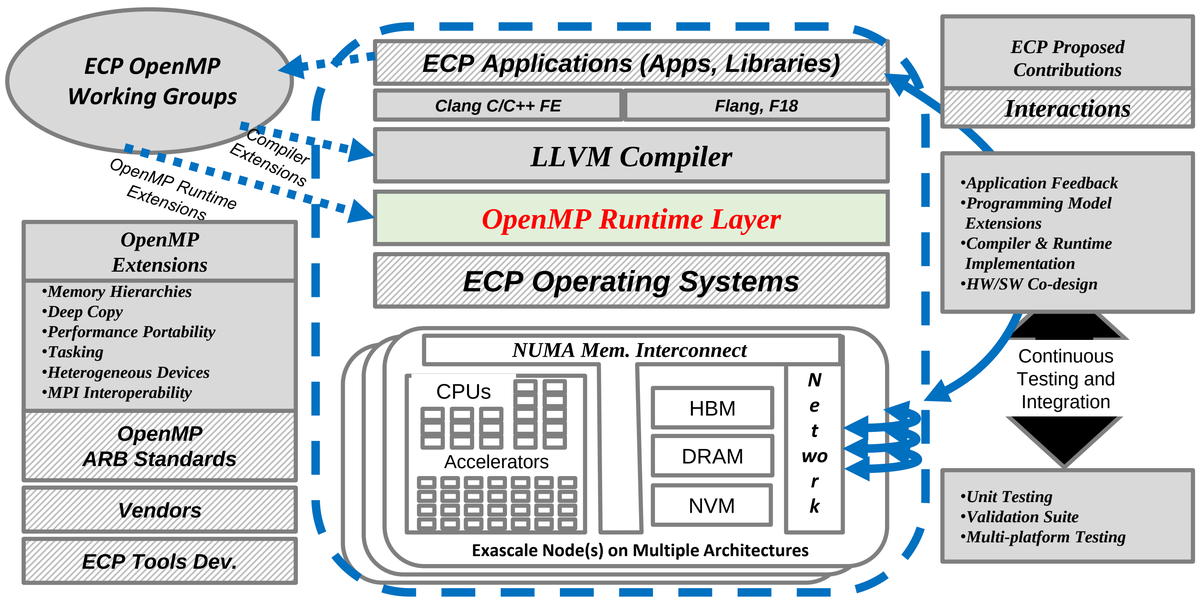
Preemptable Lightweight Threads in BOLT

- Some existing OpenMP programs that assuming Pthreads for underlying OpenMP threads may cause a deadlock.
  - These programs rely on *preemption*, which is not supported by lightweight Argobots threads used in BOLT.
- Exploring implementations of a *preemptable* lightweight thread.
  - A thread is interrupted by a signal and yield to a newly created scheduler.





OpenMP Runtime within ECP and the SOLLVE Project



Connections with Other ECP Projects

- 2.3.1.07 STPM09 Exascale MPI
- 2.3.1.11 STPM13 OMPI-X: Open MPI for Exascale
- 2.3.3.09 STMS10 SLATE: Software for Linear Algebra Targeting Exascale

POC: Vivek Kale vkale@bnl.gov and Seonmyeong Bak sbak5@gatech.edu (loop and task scheduling), Shintaro Iwasaki siwasaki@anl.gov (BOLT)

EXASCALE COMPUTING PROJECT