Argonne NATIONAL LABORATORY

User-Level Threads and OpenMP

Sangmin Seo

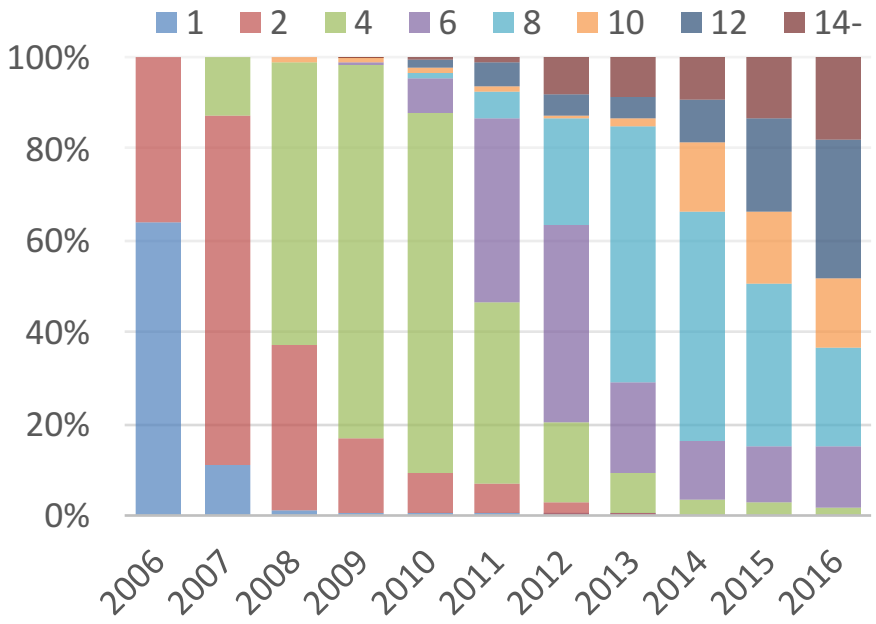
Assistant Computer Scientist
Argonne National Laboratory
sseo@anl.gov

November 7, 2016

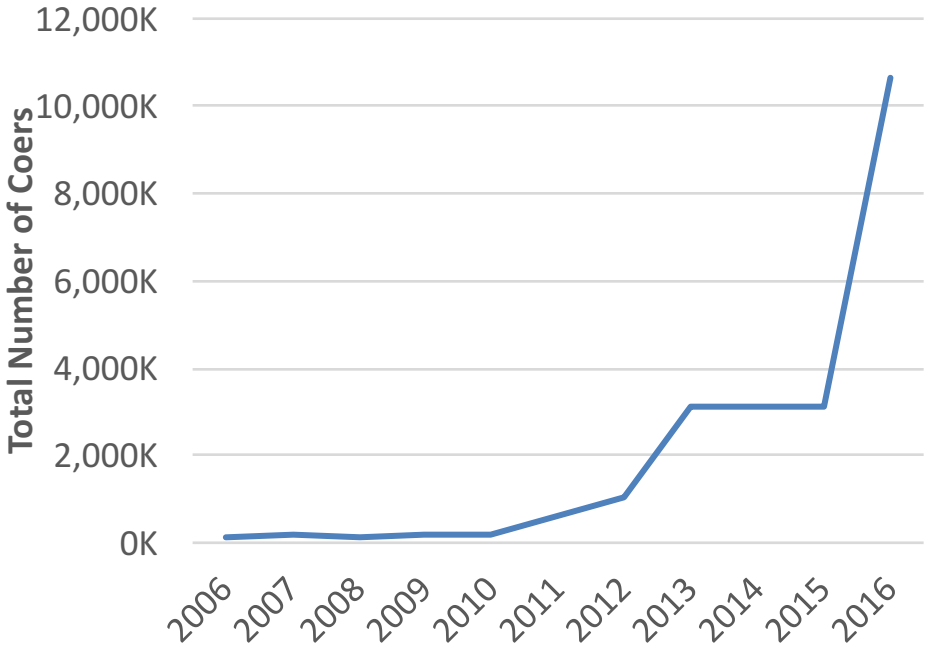
U.S. DEPARTMENT OF ENERGY

Top500 Supercomputers Today

1. Sunway TaihuLight (Sunway MPP): 10,646,600 cores
2. Tianhe-2 (Intel Xeon + Xeon Phi): 3,120,000 cores
3. Titan (Cray XK7 + Nvidia K20x): 560,640 cores
4. Sequoia (BlueGene/Q): 1,572,864 cores
5. K computer (SPARC64): 705,024 cores
6. Mira (BlueGene/Q): 786,432 cores



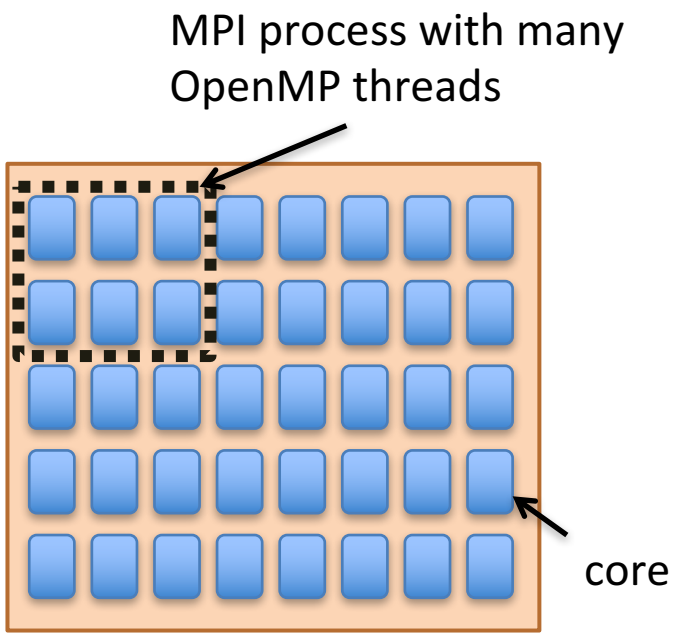
Number of cores per socket in Top500



Total number of cores in Top500 rank #1

Massive On-node Parallelism

- To address massive on-node parallelism, the number of work units (e.g., threads) must increase by 100X
- MPI+OpenMP is sufficient for many apps, but implementation is poor
  - Today **MPI+OpenMP == MPI+Pthreads**
- Pthread abstraction is too generic, not suitable for HPC
  - Lack of fine-grained scheduling, memory management, network management, signaling, etc.
- Better runtime can significantly improve MPI+OpenMP performance and support other emerging programming models

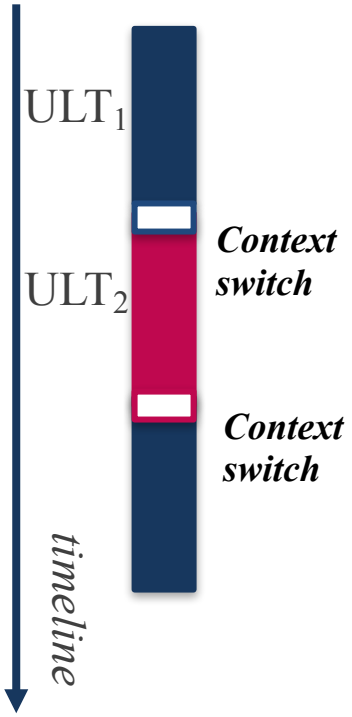


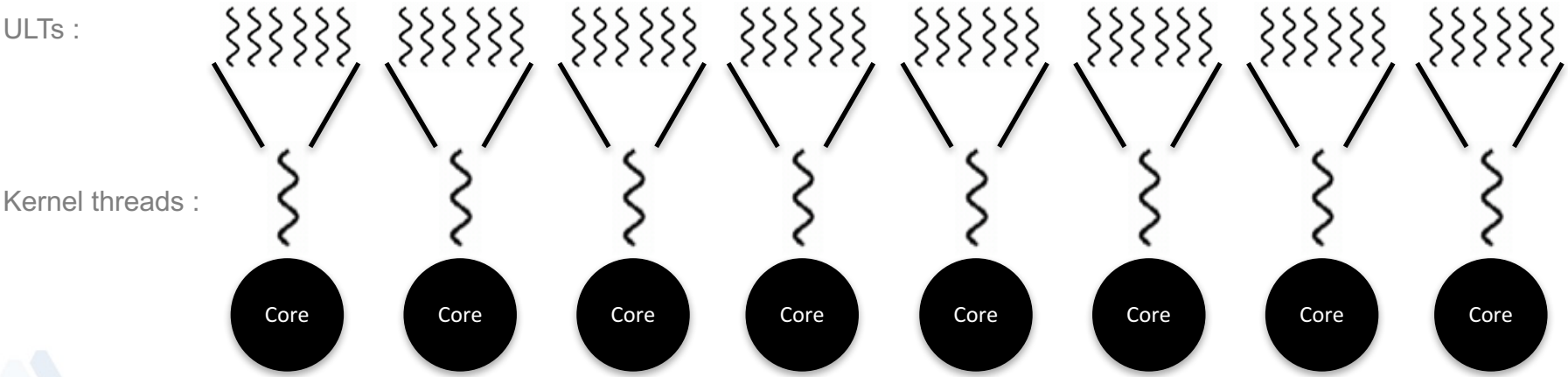
Outline

- Motivation
- User-Level Threads (ULTs)
- Argobots
- BOLT: OpenMP over Lightweight Threads
- Summary

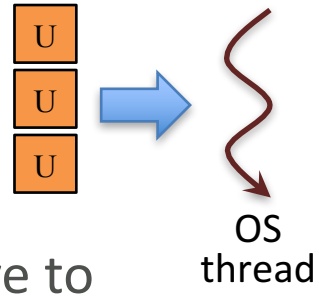
User-Level Threads (ULTs)

- What is user-level thread (ULT)?
  - Provides thread semantics in user space
  - Execution model: **cooperative timesharing**
    - More than one ULT can be mapped to a single kernel thread
    - ULTs on the same OS thread do not execute in parallel
  - Can be implemented with coroutines
    - Enable explicit suspend and resume of its progress by preserving execution state
    - Some languages such as Python and Go use coroutines for asynchronous I/O



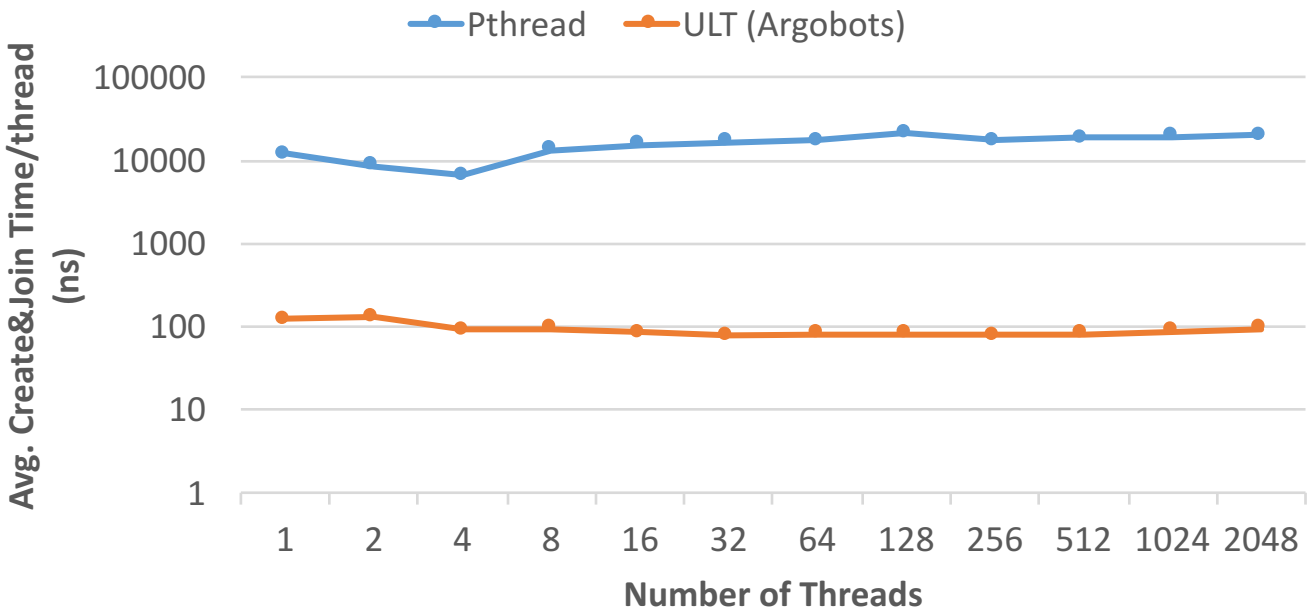


User-Level Threads (ULTs)



- Why ULTs?
  - Conventional threads (e.g., Pthreads) are too expensive to express massive parallelism
  - If we create Pthreads more than # of cores (i.e., oversubscription):
    - Context-switch and synchronization overhead will increase dramatically
  - ULTs can mitigate high overhead of Pthreads but need explicit control
- Where to use?
  - To better overlap computation and communication/IO
    - Low context-switching overhead of ULTs can give more opportunities to hide communication/IO latency
  - To exploit fine-grained task parallelism
    - Lightweight ULTs are more suitable to express massive task parallelism

Pthreads vs. ULTs



- Average time for creating and joining one thread
  - Pthread: 6.6us - 21.2us (avg. 34,953 cycles)
  - ULT (Argobots): 78ns - 130ns (avg. 191 cycles)
  - ULT is **64x - 233x faster** than Pthread
- How fast is ULT?
  - L1$ access: 1.112ns, L2$ access: 5.648ns, memory access: 18.4ns
  - Context switch (2 processes): 1.64us

* measured using LMbench3

Growing Interests in ULTs

- ULT and task libraries
  - Converse threads, Qthreads, MassiveThreads, Nanos++, Maestro, GnuPth, StackThreads/MP, Protothreads, Capriccio, StateThreads, TiNy-threads, etc.
- OS supports
  - Windows fibers, Solaris threads
- Language and programming models
  - Cilk, OpenMP task, C++11 task, C++17 coroutine proposal, Stackless Python, Go coroutines, etc.

- Pros
  - *Easy to use with Pthreads-like interface*
- Cons
  - *Runtime tries to do something smart (e.g., work-stealing)*
  - *This may conflict with the characteristics and demands of applications*

Argobots

A lightweight low-level threading and tasking framework

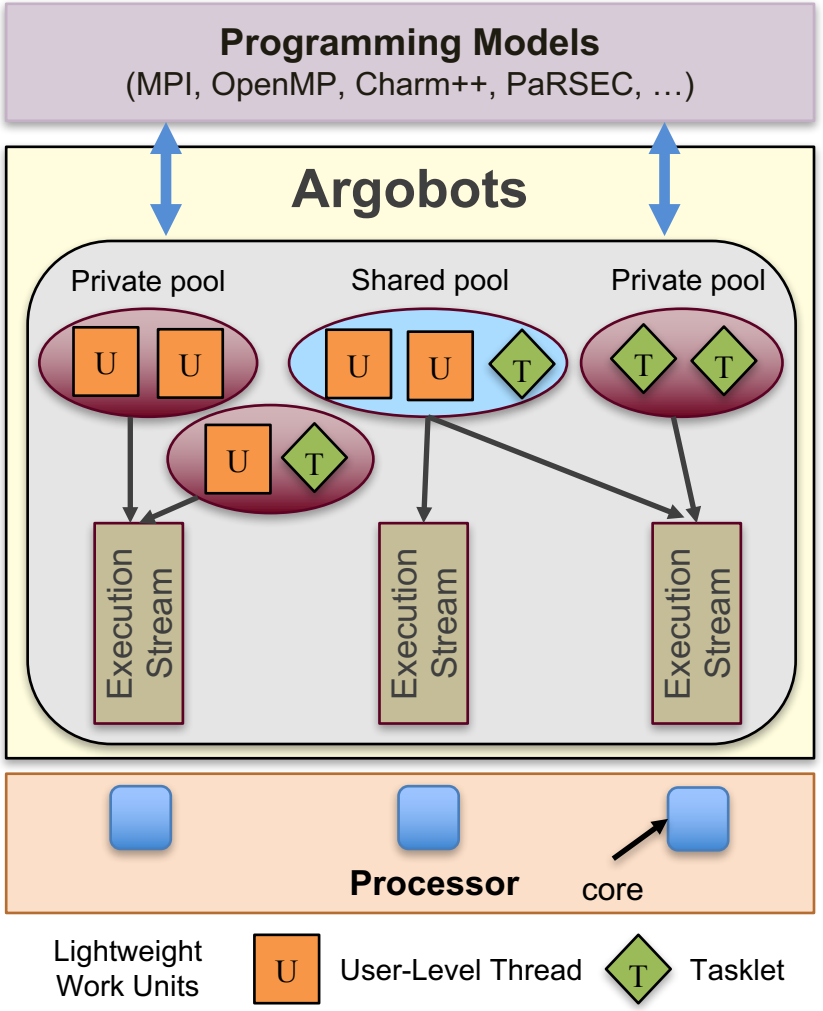
(http://www.mcs.anl.gov/argobots/)

Overview

- Separation of mechanisms and policies
- Massive parallelism
  - **Exec. Streams** guarantee progress
  - **Work Units** execute to completion
    - User-level threads (ULTs) vs. Tasklets
- Clearly defined memory semantics
  - Consistency domains
    - Provide Eventual Consistency
  - Software can manage consistency

Argobots Innovations

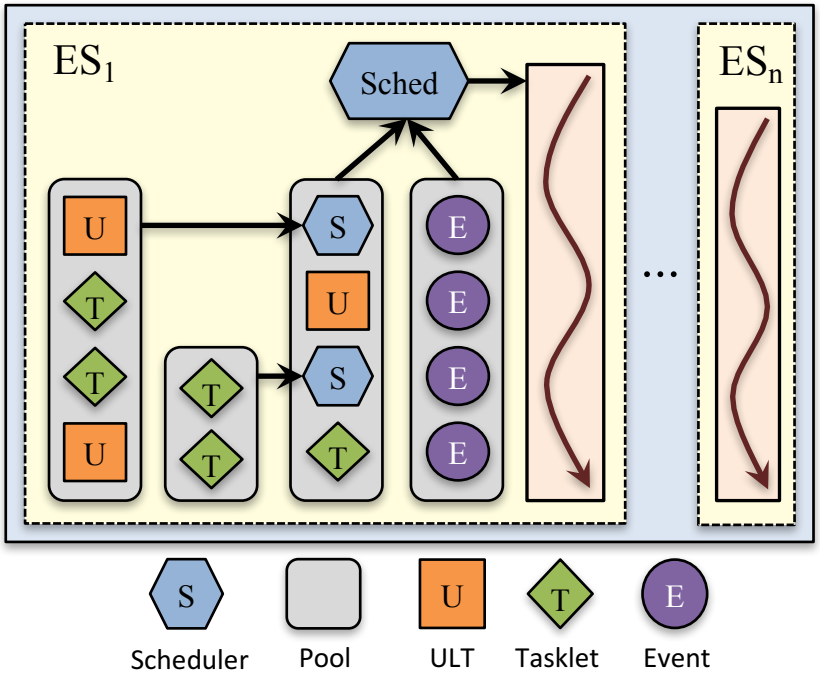
- **Enabling technology, but not a policy maker**
  - High-level languages/libraries such as OpenMP or Charm++ have more information about the user application (data locality, dependencies)
- **Explicit model**:
  - Enables dynamism, but always managed by high-level systems



* Current team members: Pavan Balaji, Sangmin Seo, Halim Amer (ANL), L. Kale, Nitin Bhat (UIUC)

Argobots Execution Model

- **Execution Streams (ES)**
  - Sequential instruction stream
    - Can consist of one or more work units
  - Mapped efficiently to a hardware resource
  - Implicitly managed progress semantics
    - One blocked ES cannot block other ESs
- **User-level Threads (ULTs)**
  - Independent execution units in user space
  - Associated with an ES when running
  - Yieldable and migratable
  - Can make blocking calls
- **Tasklets**
  - Atomic units of work
  - Asynchronous completion via notifications
  - Not yieldable, migratable before execution
  - Cannot make blocking calls

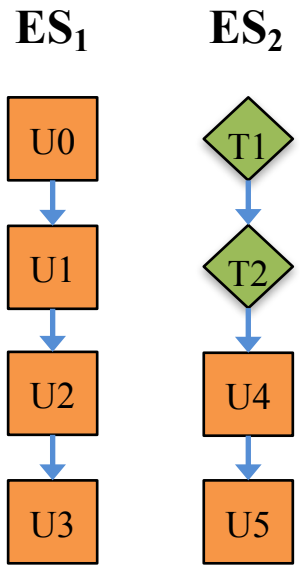


Argobots Execution Model

- **Scheduler**
  - Stackable scheduler with pluggable strategies
- **Synchronization primitives**
  - Mutex, condition variable, barrier, future
- **Events**
  - Communication triggers

Explicit Mapping ULT/Tasklet to ES

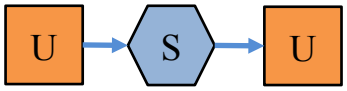
- The user needs to map work units to ESs
- No smart scheduling, no work-stealing unless the user wants to use



- Benefits
  - Allow locality optimization
    - Execute work units on the same ES
  - No expensive lock is needed between ULTs on the same ES
    - They do not run in parallel
    - A flag is enough

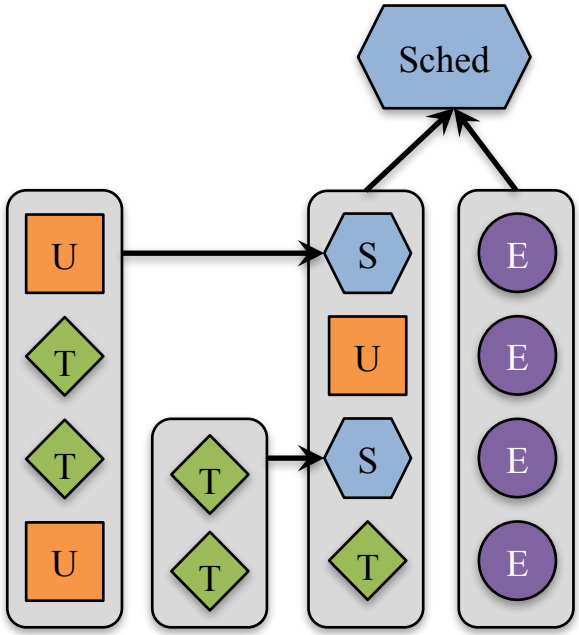
Stackable Scheduler with Pluggable Strategies

- Associated with an ES
- Can handle ULTs and tasklets
- *Can handle schedulers*
  - Allows to stack schedulers hierarchically
- Can handle asynchronous events
- *Users can write schedulers*
  - Provides **mechanisms**, not policies
  - Replace the default scheduler
    - E.g., FIFO, LIFO, Priority Queue, etc.
- ULT can explicitly *yield to* another ULT
  - Avoid scheduler overhead



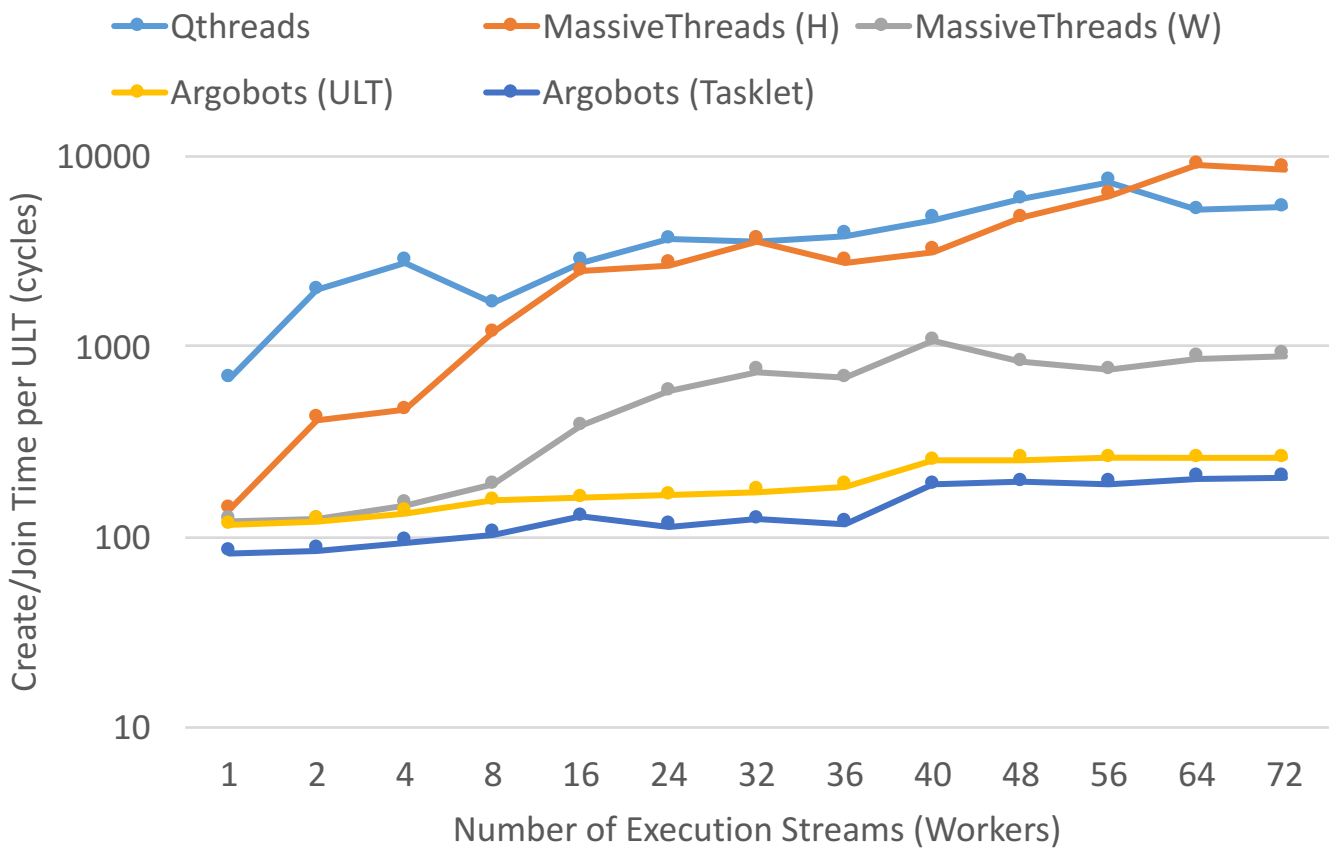
yield()

yield_to(target)

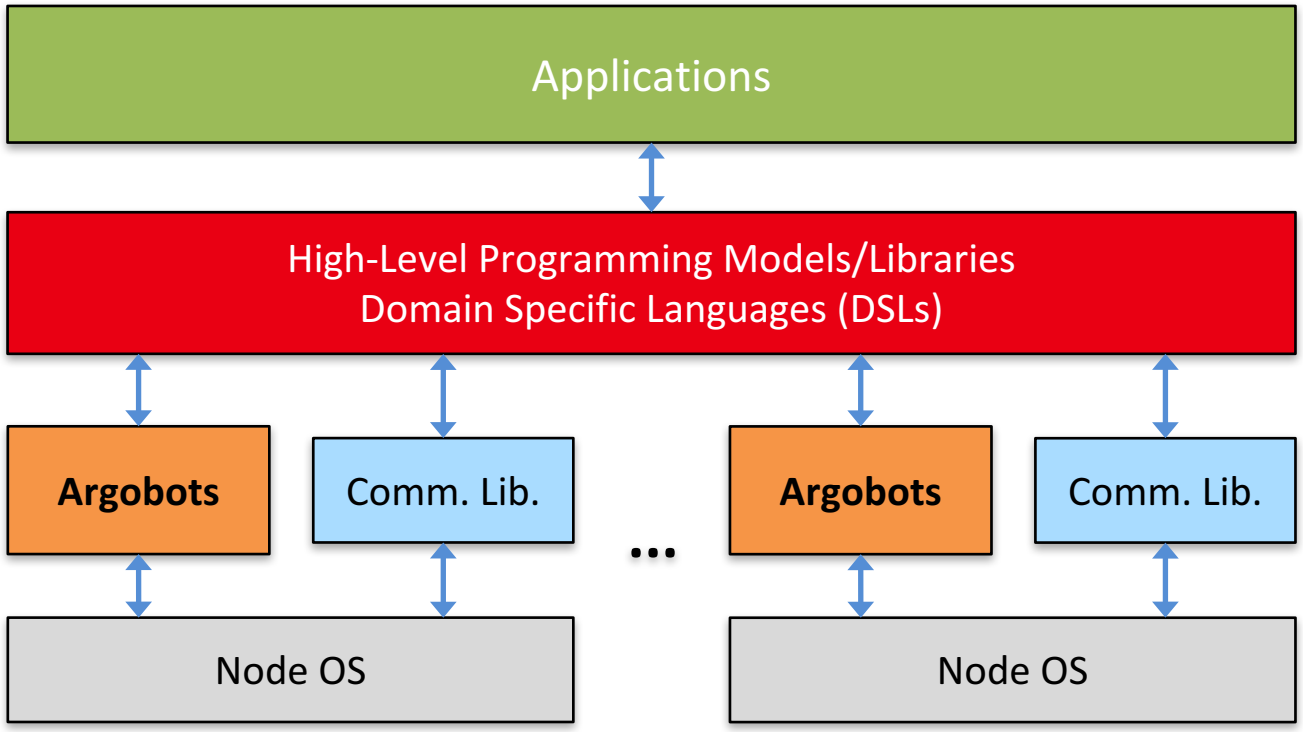


Performance: Create/Join Time

- Ideal scalability
  - If the ULT runtime is perfectly scalable, the time should be the same regardless of the number of ESs

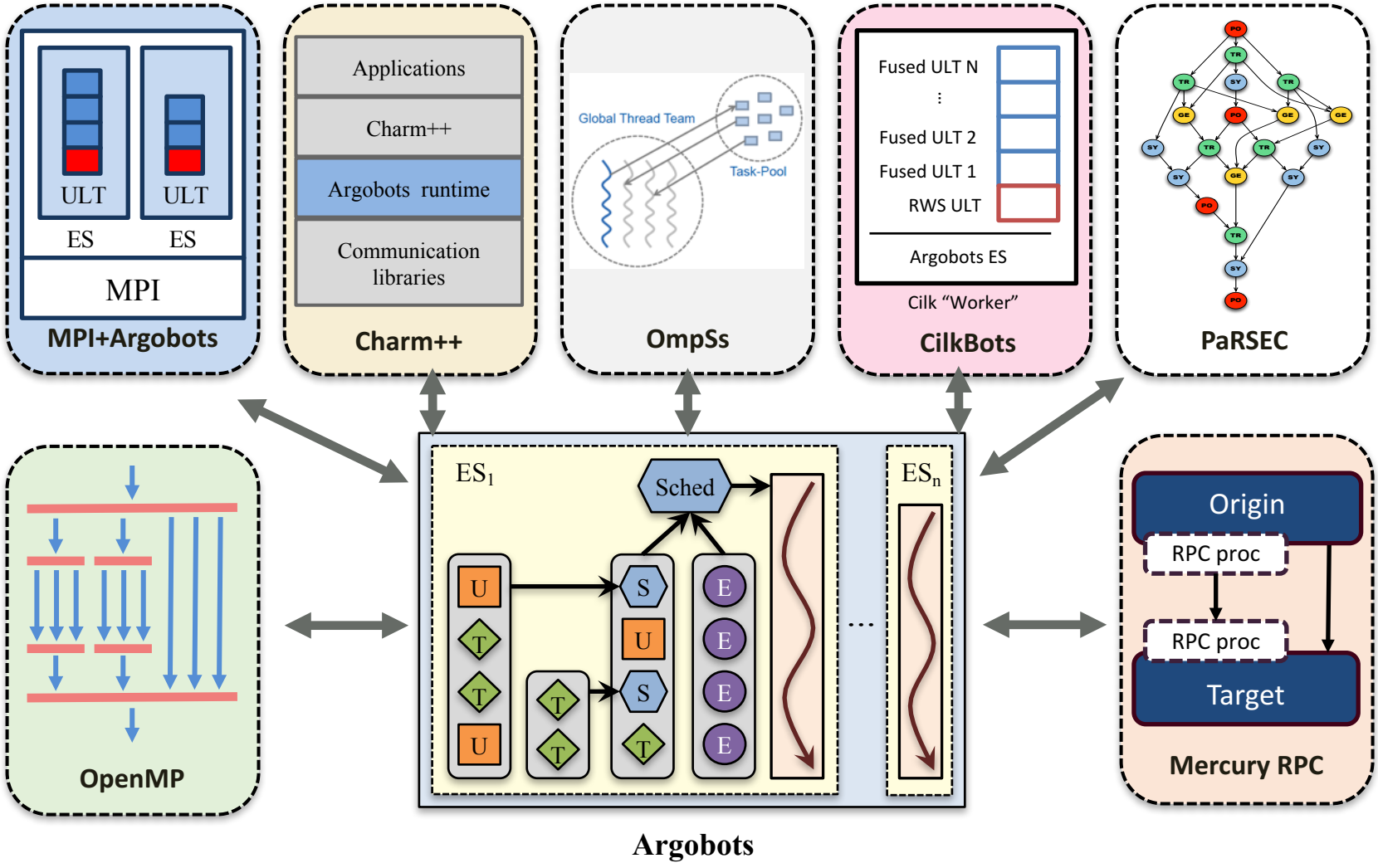


Argobots' Position



Argobots is a low-level threading/tasking runtime!

Argobots Ecosystem



OpenMP

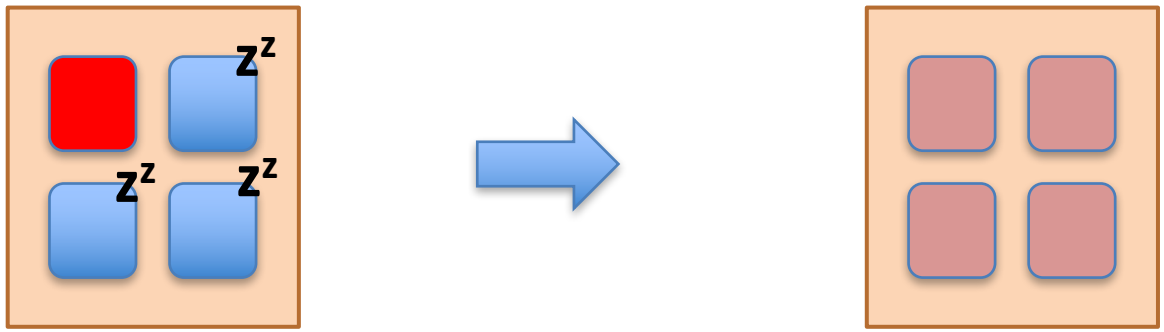
- Directive based programming model
- Commonly used for shared-memory programming in a node
- Many different implementations
  - Typically on top of Pthreads library
  - Intel, GCC, Clang, IBM, etc.

Sequential code

```
for (i = 0; i < N; i++) {
    do_something();
}
```

OpenMP code

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
    do_something();
}
```



Nested Parallel Loop: Microbenchmark

```
int in[1000][1000], out[1000][1000];

#pragma omp parallel for
for (i = 0; i < 1000; i++) {
   lib_compute(i);
}

lib_compute(int x)
{
   #pragma omp parallel for
   for (j = 0; j < 1000; j++)
      out[x][j] = compute(in[x][j]);
}
```

A thread for each CPU is created by default

Each thread executes a portion

Each thread creates more threads for the second loop

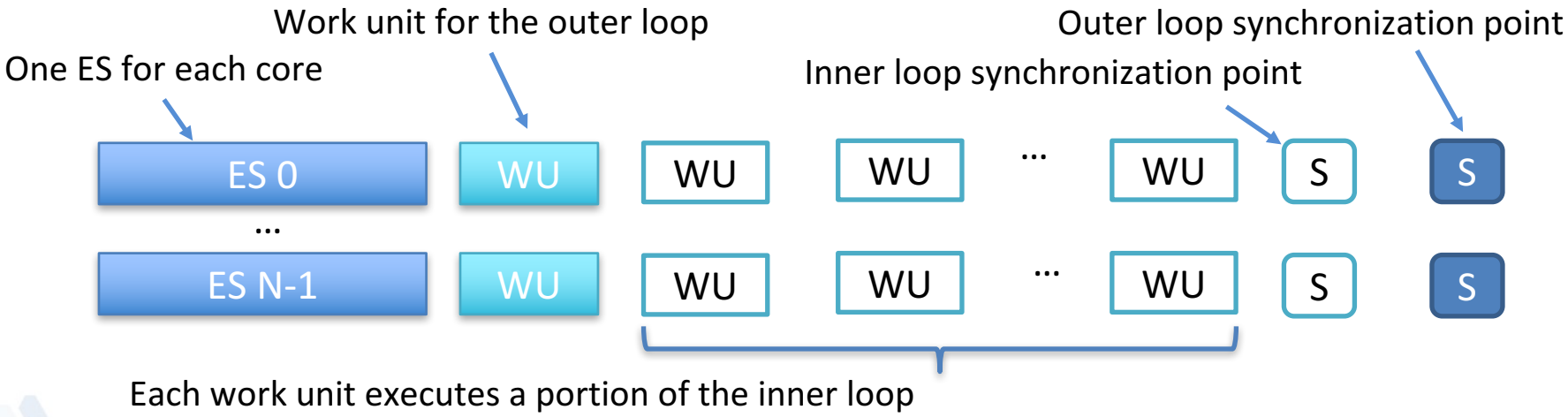
Each inner thread executes a portion

Contribution: Adrian Castello (Universitat Jaume I)

Nested Parallel Loop: Implementations

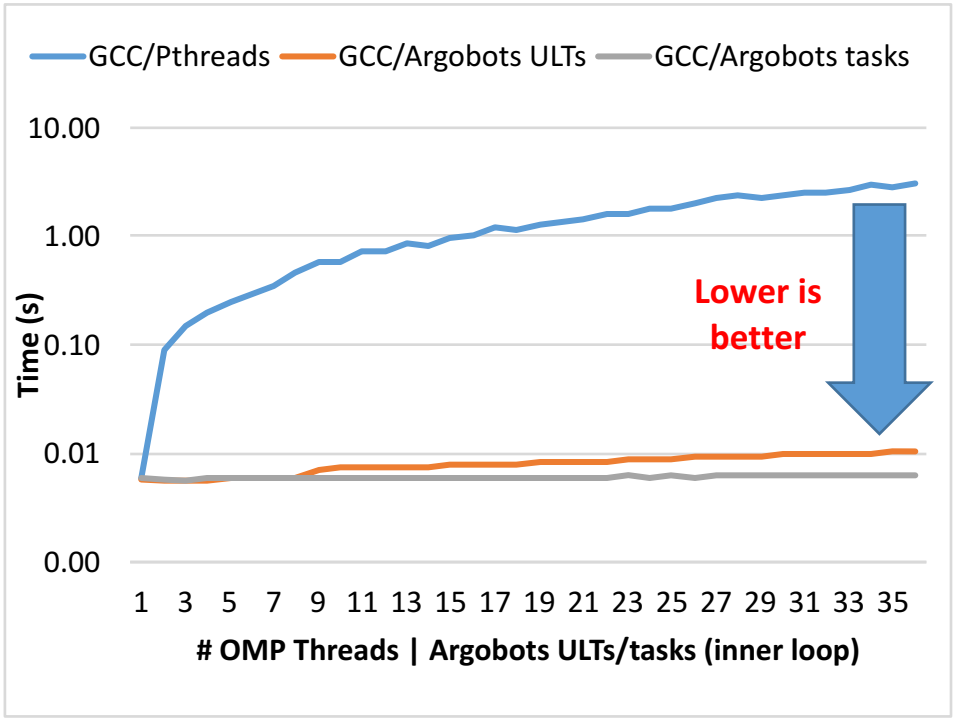
- GCC
  - Does not reuse the idle threads in nested parallel constructs
  - All thread teams inside a parallel region need to be created
- ICC
  - Reuse idle threads
    - If there are not more threads available, new threads are created
- *All created threads are OS threads and add overhead*

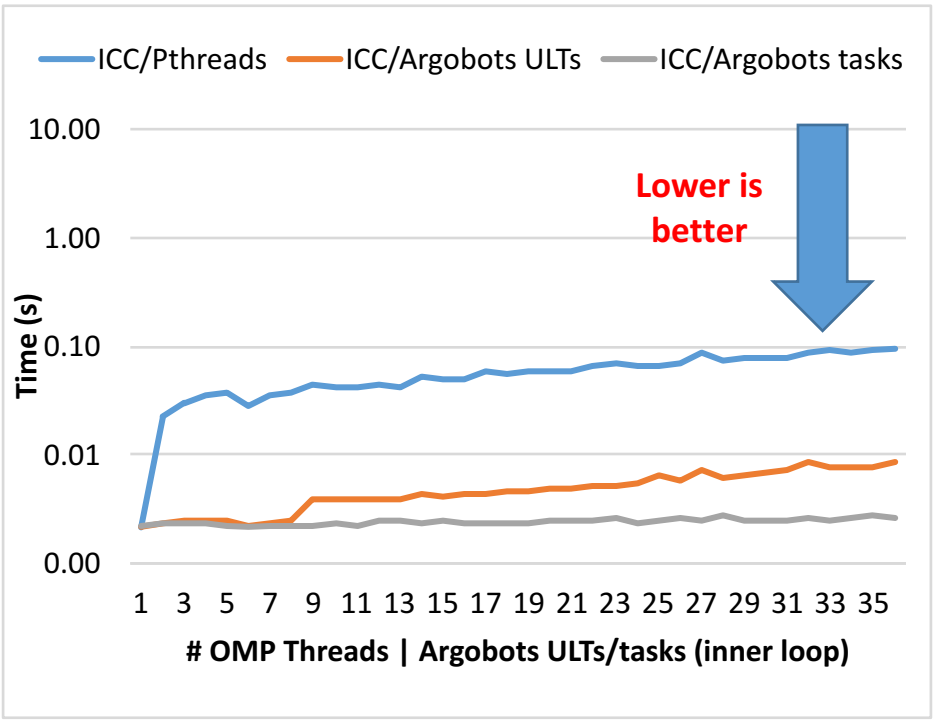
- Implementation using Argobots
  - Creates ULTs or tasklets for both outer loop and inner loop



Nested Parallel Loop: Performance

Execution time for 36 threads in the outer loop





GCC OpenMP implementation does not reuse idle threads in nested parallel regions, all the teams of threads need to be created in each iteration

Some overhead is added by creating ULTs instead of tasks

Nested Parallel Loop: Analysis

- How does each implementation manage the threads in nested parallel regions?
  - Parameters:
    - C: number of cores (or threads created by user at the beginning)
    - N: number of iterations of the outer loop
    - M: number of iterations of the inner loop
    - T: number of threads for the inner loop

Example -> C: 36, N: 1,000, M: 1,000, T:36

Imple.	# Created Threads	# Reused Threads	# Created ULTs	# Created Tasks
GCC	C + N*(T-1) = **35,036**	0	---	---
ICC	C + C*(T-1) = **1,296**	(N-C)*(T-1) = 33,740	---	---
Argobots tasks	C = 36	0	C = 36	N*T = **36,000**

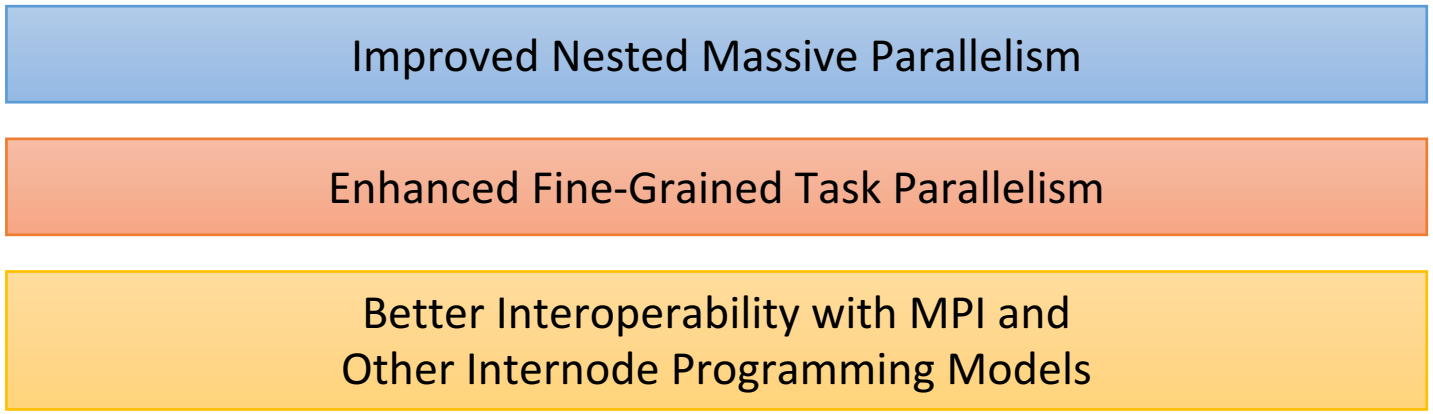
Sequential creation

Parallel creation

BOLT: A Lightning-Fast OpenMP Implementation

- About BOLT
  - BOLT is a recursive acronym that stands for "BOLT is OpenMP over Lightweight Threads"
  - https://www.mcs.anl.gov/bolt/

- Objective
  - OpenMP framework that exploits *lightweight threads and tasks*

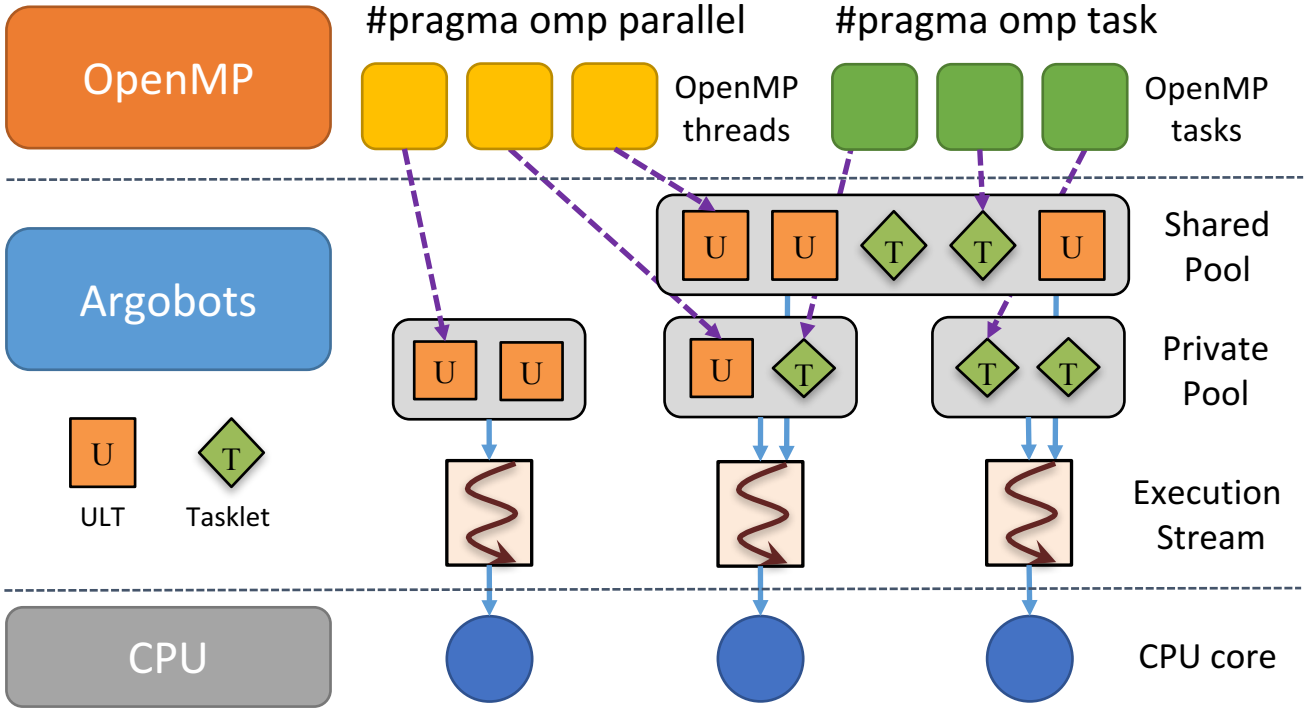


Approach & Development

- Basic approach
  - Compiler simply generates runtime API calls, while the runtime creates ULTs/tasklets and manages them over a fixed set of computational resources
  - Use ***Argobots*** as the underlying threading and tasking mechanism
  - ABI compatibility with Intel OpenMP compilers, LLVM/Clang, and GCC (i.e., can be used with these compilers)
- Development
  - Runtime
    - Based on Intel OpenMP Runtime API
    - Generates Argobots work units from OpenMP pragmas
    - Can generate ULTs or tasklets depending on code characteristics
  - Compiler (planned)
    - LLVM/Clang
    - Passes characteristics of parallel region or task (e.g., existence of blocking calls) to the runtime
    - Extends pragmas with the option “nonblocking”

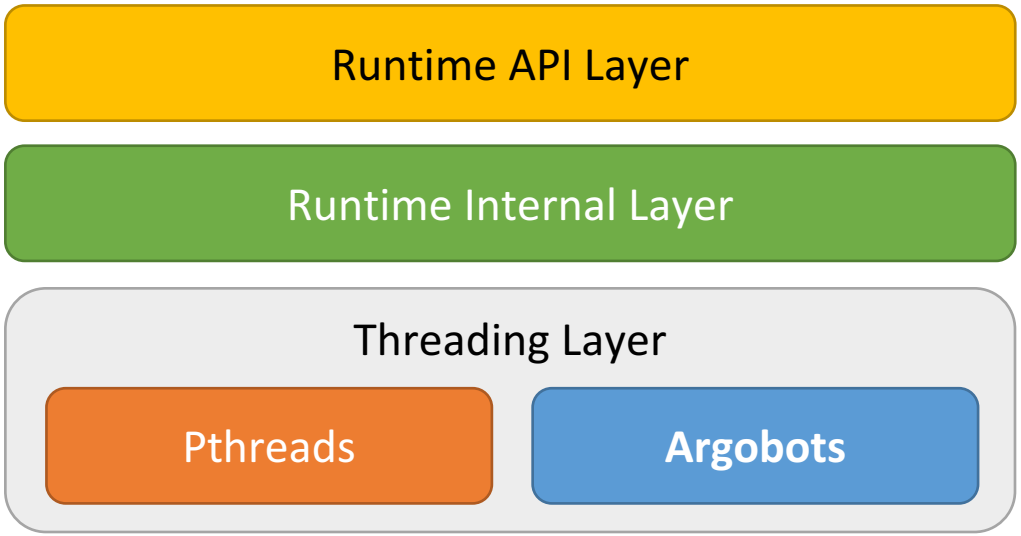
BOLT Execution Model

- OpenMP threads and tasks are translated into Argobots work units (i.e., ULTs and tasklets)
- Shared pools are utilized to handle nested parallelism
- A customized Argobots scheduler manages scheduling of work units across execution streams



Prototype Implementation of BOLT Runtime

- Based on Intel's open-source OpenMP runtime
  - http://openmp.llvm.org/
- Kept the original runtime API for the ABI compatibility
- Designed and implemented the threading layer using Argobots and modified the runtime internal layer



OpenMP Pragma Translation

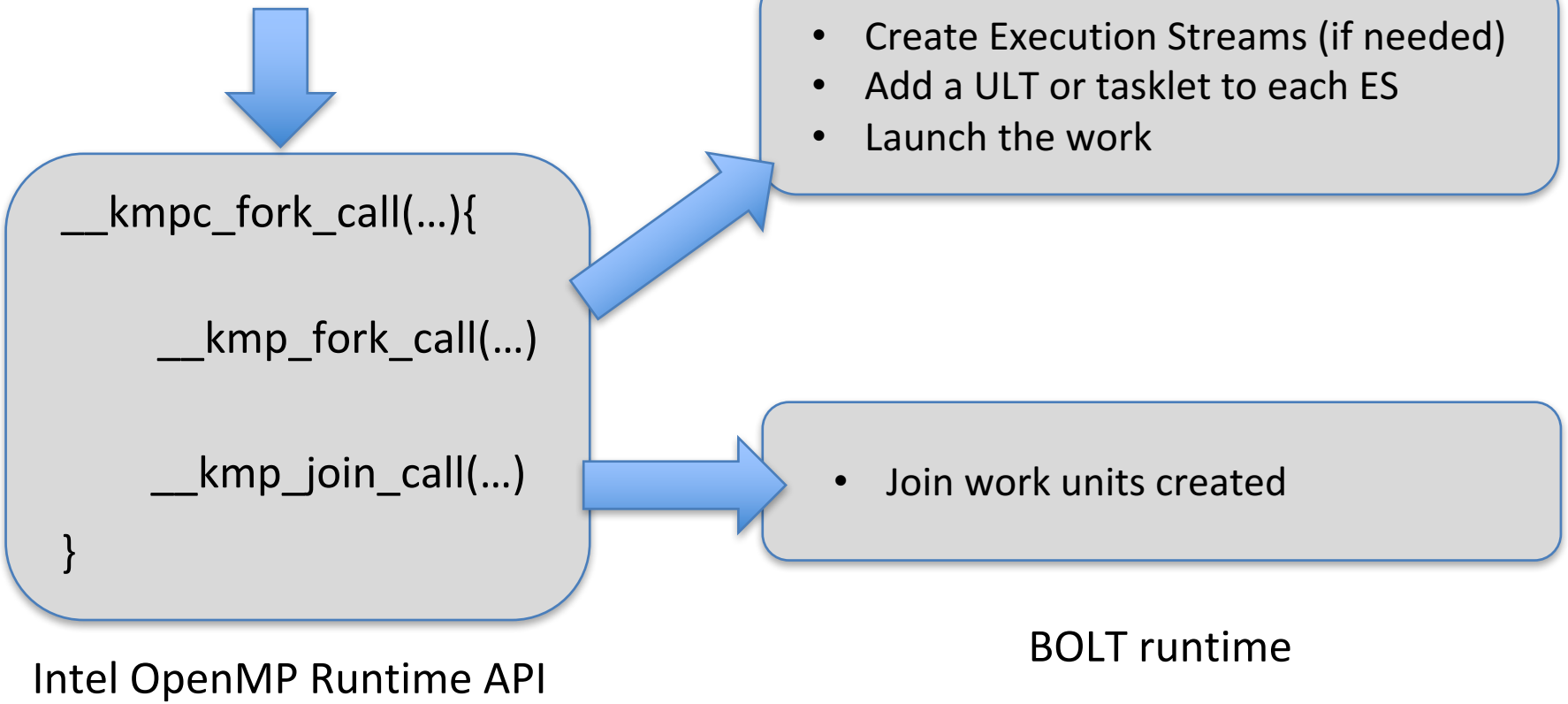
1. A set of *N* threads is created at run time
   - If they have not been created yet
   - Commonly as many as the number of CPU cores
2. The number of iterations is divided between all the threads
3. A synchronization point is added after the for loop
   - Implicit barrier at the end of parallel for

```
#pragma omp parallel for (1,2)
for (i = 0; i < N; i++) {
    do_something();
} (3)
```

OpenMP Compiler & BOLT Runtime

#pragma omp parallel

Clang and Intel compiler



parallel for

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
    do_something();
}
```

Creates threads

Divides all iterations among threads

Synchronization point

Implementation using Argobots

One Execution Stream
for each CPU core
(or hardware thread)

Each work unit executes
a portion of the for loop

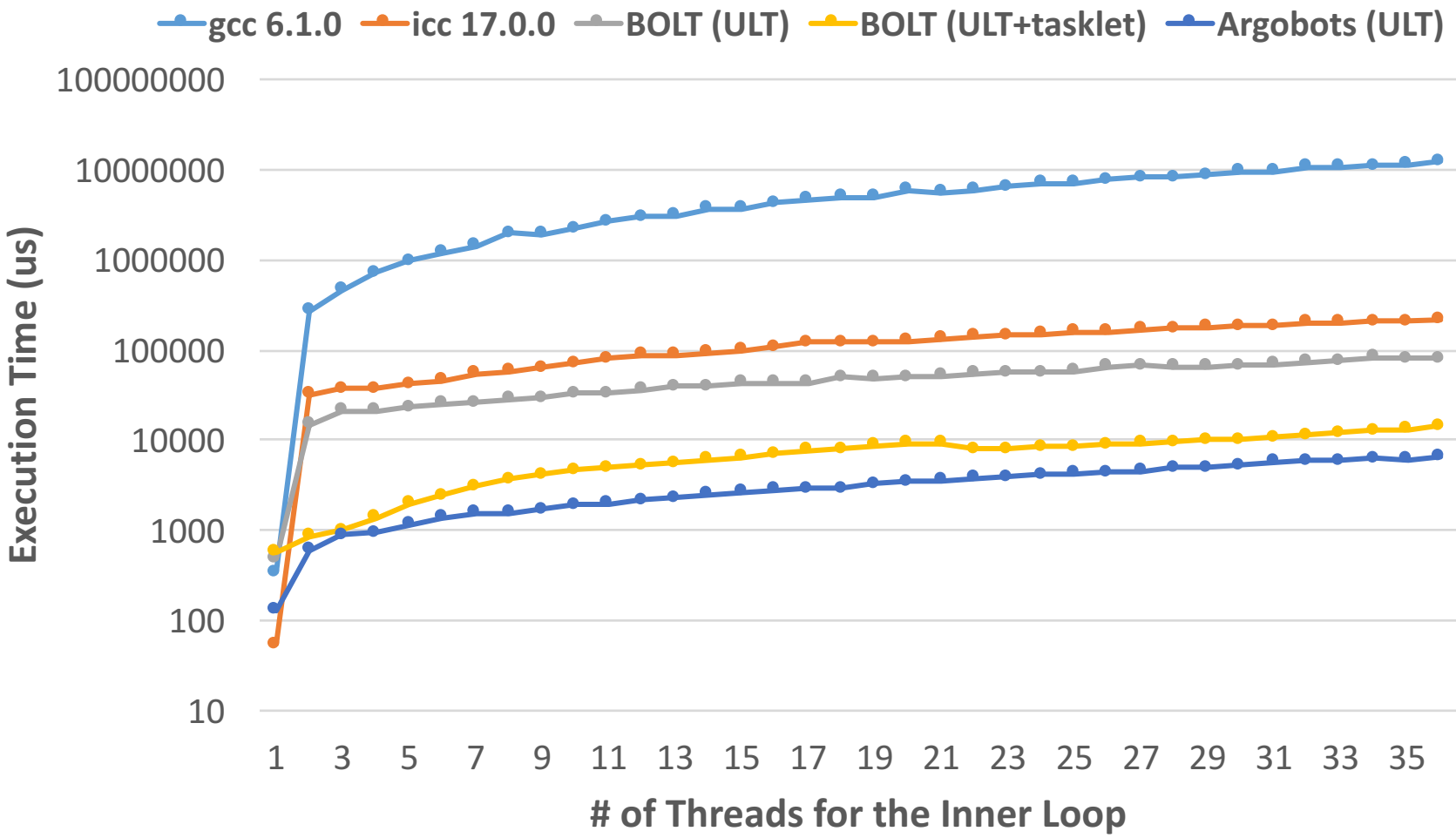
A synchronization point is added

OpenUH OpenMP Validation Suite 3.1

	GCC 6.1	ICC 17.0.0 + Intel OpenMP	ICC 17.0.0 + BOLT runtime (Argobots)	BOLT (clang + Argobots)
# of tested OpenMP constructs	62	62	62	62
# of used tests	123	123	123	123
# of successful tests	**118**	**118**	**122**	**112**
# of failed tests	5	5	1	1
Pass rate (%)	**95.9**	**95.9**	**99.2**	**99.2**

- The BOLT prototype functionally works well!

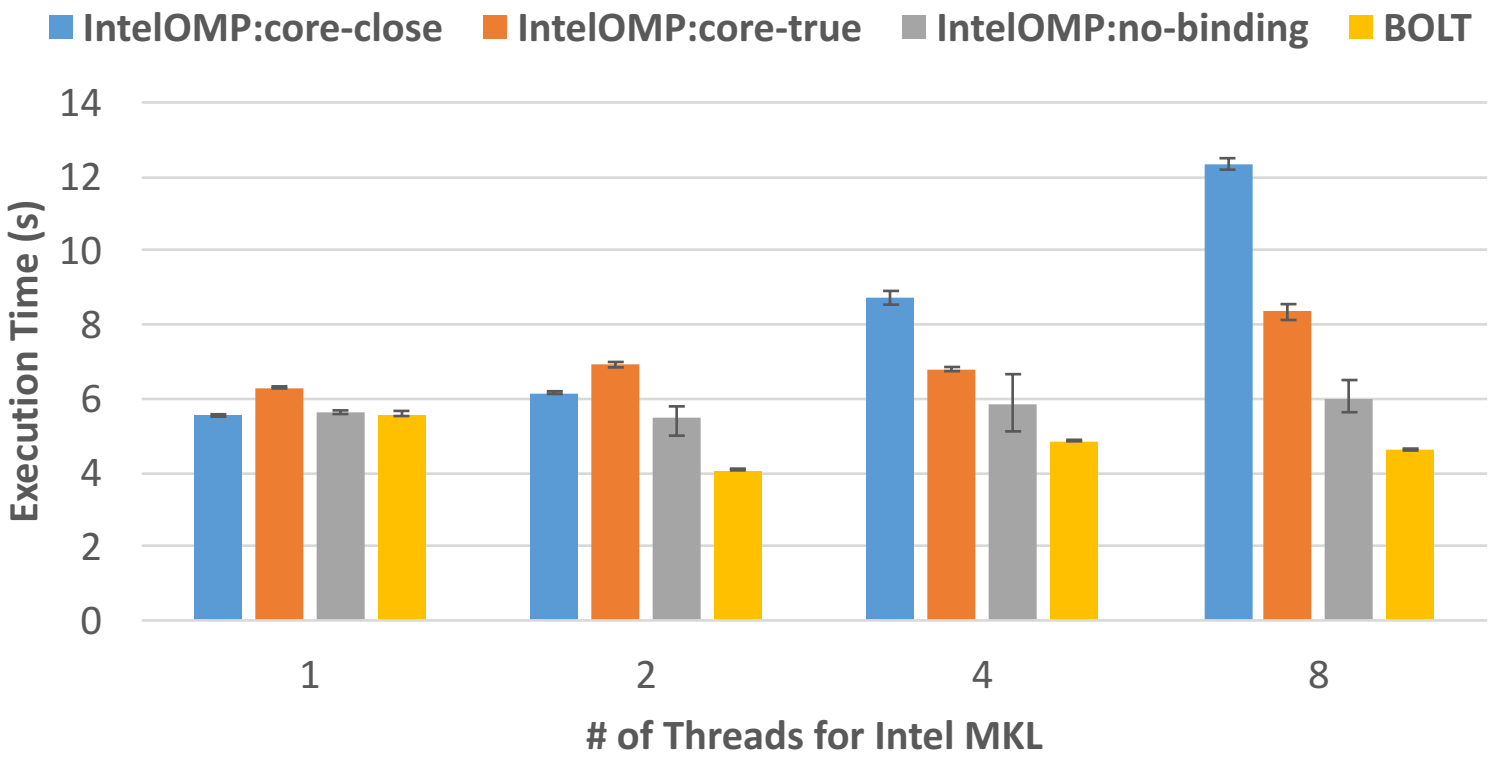
Nested Parallel Loop Microbenchmark



* The number of threads for the outer loop was fixed at 36.

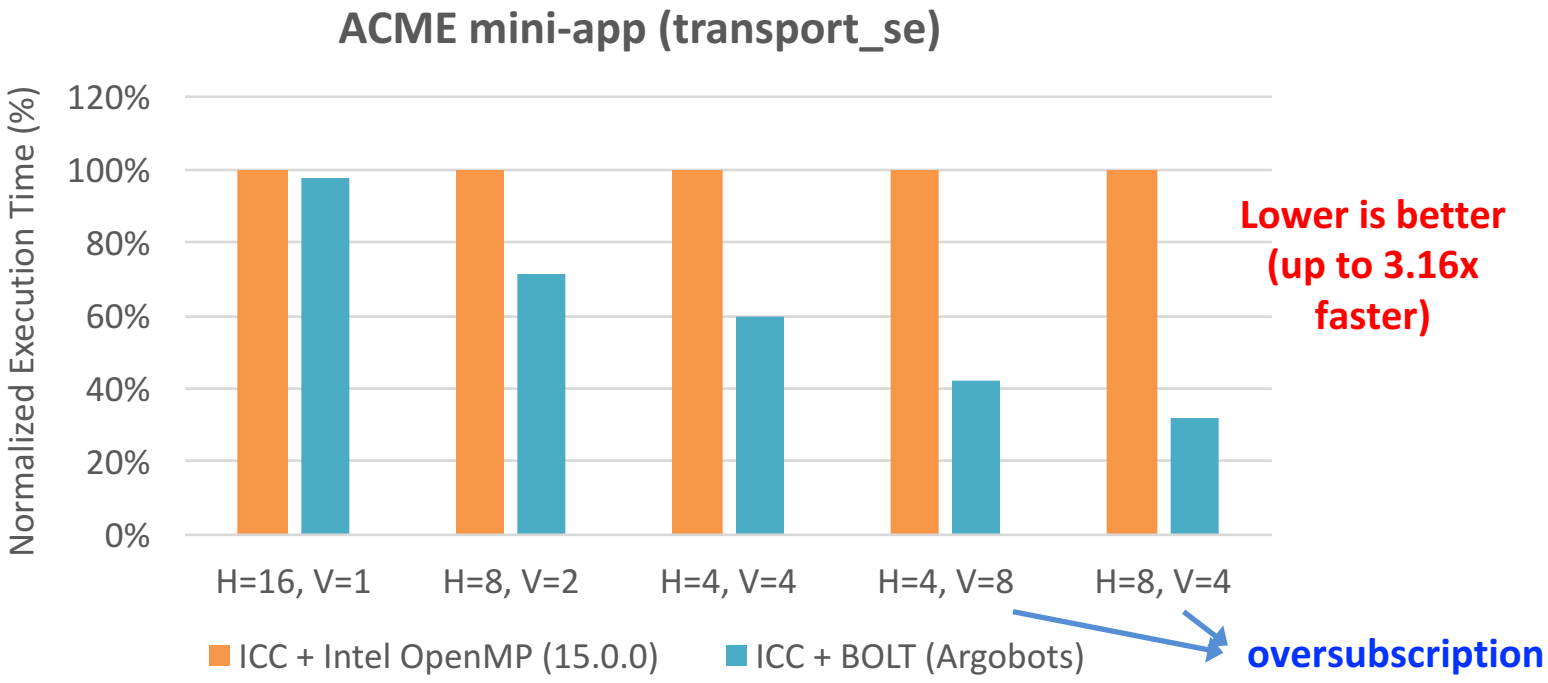
Application Study: KIFMM

- Kernel-Independent Fast Multipole Method (KIFMM)
  - Offload dgemv operations to Intel MKL
- Evaluated the efficiency of the nested parallelism support in Intel OpenMP and BOLT during the Downward stage
  - 9 threads for the application (outer parallel region)



Application Study: ACME mini-app

- ACME (Accelerated Climate Modeling for Energy)
  - Implementing additional levels of parallelism through OpenMP nested parallel loops for upcoming many-core machines
- Preliminary results of testing the `transport_se` mini-app version of HOMME (ACME's CAM-SE dycore)



Summary

- Massive on-node parallelism is inevitable
  - Need runtime systems utilizing such parallelism
- User-level threads (ULTs)
  - Lightweight threads more suitable for fine-grained dynamic parallelism and computation-communication overlap
- Argobots
  - A lightweight low-level threading/tasking framework
  - Provides efficient mechanisms, not policies, to users (library developers or compilers)
    - They can build their own solutions
- BOLT: OpenMP over Lightweight Threads
  - More efficient support of nested parallelism with Argobots ULTs and tasklets
  - Preliminary results show that BOLT is promising

Argo Concurrency Team

- Argonne National Laboratory (ANL)
  - Pavan Balaji (co-lead)
  - Sangmin Seo
  - Abdelhalim Amer
  - Pete Beckman (PI)
- University of Illinois at Urbana-Champaign (UIUC)
  - Laxmikant Kale (co-lead)
  - Marc Snir
  - Nitin Kundapur Bhat
- University of Tennessee, Knoxville (UTK)
  - George Bosilca
  - Thomas Herault
  - Damien Genet
- Pacific Northwest National Laboratory (PNNL)
  - Sriram Krishnamoorthy

Past Team Members:

- Cyril Bordage (UIUC)
- Prateek Jindal (UIUC)
- Jonathan Lifflander (UIUC)
- Esteban Meneses (University of Pittsburgh)
- Huiwei Lu (ANL)
- Yanhua Sun (UIUC)

BOLT Collaborations

- **Maintainers**
  - Argonne National Laboratory
    - Sangmin Seo
    - Abdelhalim Amer
    - Pavan Balaji
- **Contributors**
  - Universitat Jaume I de Castelló
    - Adrián Castelló
    - Rafael Mayo
    - Enrique S. Quintana-Ortí
  - Barcelona Supercomputing Center (BSC)
    - Antonio J. Peña
    - Jesus Labarta
  - RIKEN
    - Jinpil Lee
    - Mitsuhisa Sato

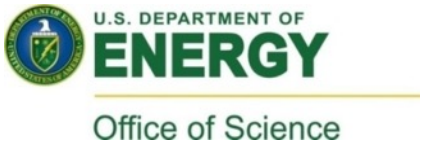
Try Argobots & BOLT

- **Argobots**
  - http://www.mcs.anl.gov/argobots/
  - git repository
    - https://github.com/pmodels/argobots
  - Wiki
    - https://github.com/pmodels/argobots/wiki
  - Doxygen
    - http://www.mcs.anl.gov/~sseo/public/argobots/

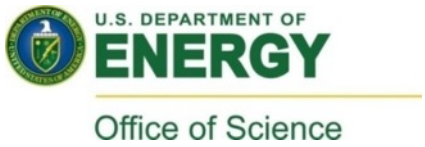
- **BOLT**
  - http://www.mcs.anl.gov/bolt/
  - git repository
    - https://github.com/pmodels/bolt-runtime

Funding Acknowledgments

Funding Grant Providers



Infrastructure Providers



Programming Models and Runtime Systems Group

Group Lead

- Pavan Balaji (computer scientist and group lead)

Current Staff Members

- Abdelhalim Amer (postdoc)
- Yanfei Guo (postdoc)
- Rob Latham (developer)
- Lena Oden (postdoc)
- Ken Raffenetti (developer)
- Sangmin Seo (assistant computer scientist)
- Min Si (postdoc)

Past Staff Members

- Antonio Pena (postdoc)
- Wesley Bland (postdoc)
- Darius T. Buntinas (developer)
- James S. Dinan (postdoc)
- David J. Goodell (developer)
- Huiwei Lu (postdoc)
- Min Tian (visiting scholar)
- Yanjie Wei (visiting scholar)
- Yuqing Xiong (visiting scholar)
- Jian Yu (visiting scholar)
- Junchao Zhang (postdoc)
- Xiaomin Zhu (visiting scholar)

Current and Recent Students

- Ashwin Aji (Ph.D.)
- Abdelhalim Amer (Ph.D.)
- Md. Humayun Arafat (Ph.D.)
- Alex Brooks (Ph.D.)
- Adrian Castello (Ph.D.)
- Dazhao Cheng (Ph.D.)
- Hoang-Vu Dang (Ph.D.)
- James S. Dinan (Ph.D.)
- Piotr Fidkowski (Ph.D.)
- Priyanka Ghosh (Ph.D.)
- Sayan Ghosh (Ph.D.)
- Ralf Gunter (B.S.)
- Jichi Guo (Ph.D.)
- Yanfei Guo (Ph.D.)
- Marius Horga (M.S.)
- John Jenkins (Ph.D.)
- Feng Ji (Ph.D.)
- Ping Lai (Ph.D.)
- Palden Lama (Ph.D.)
- Yan Li (Ph.D.)
- Huiwei Lu (Ph.D.)
- Jintao Meng (Ph.D.)
- Ganesh Narayanaswamy (M.S.)
- Qingpeng Niu (Ph.D.)
- Ziaul Haque Olive (Ph.D.)
- David Ozog (Ph.D.)
- Renbo Pang (Ph.D.)
- Nikela Papadopoulou (Ph.D)
- Sreeram Potluri (Ph.D.)
- Sarunya Pumma (Ph.D)
- Li Rao (M.S.)
- Gopal Santhanaraman (Ph.D.)
- Thomas Scogland (Ph.D.)
- Min Si (Ph.D.)
- Brian Skjerven (Ph.D.)
- Rajesh Sudarsan (Ph.D.)
- Lukasz Wesolowski (Ph.D.)
- Shucai Xiao (Ph.D.)
- Chaoran Yang (Ph.D.)
- Boyu Zhang (Ph.D.)
- Xiuxia Zhang (Ph.D.)
- Xin Zhao (Ph.D.)

Advisory Board

- Pete Beckman (senior scientist)
- Rusty Lusk (retired, STA)
- Marc Snir (division director)
- Rajeev Thakur (deputy director)

Q&A

- Thank you for your attention!

Questions?