SAND2024-05190C

Sandia National Laboratories

User-Level Threading for HPC Applications

CEA-NNSA Meeting – June 19-22, 2023

Jan Ciesko, SNL

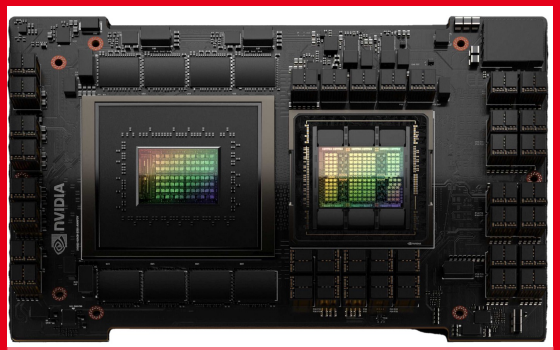
Adrien Roussel, CEA/DAM

Outline

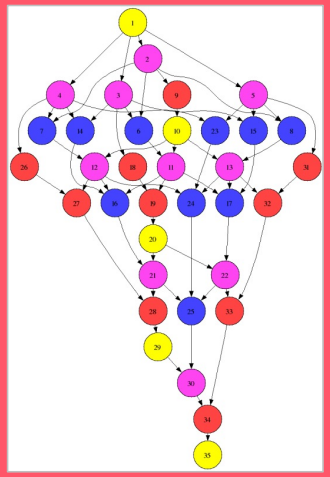
1. ULT for HPC Applications
2. Qthreads and MPC
3. Projects
4. Conclusion

ULT for HPC Applications

The opportunity:



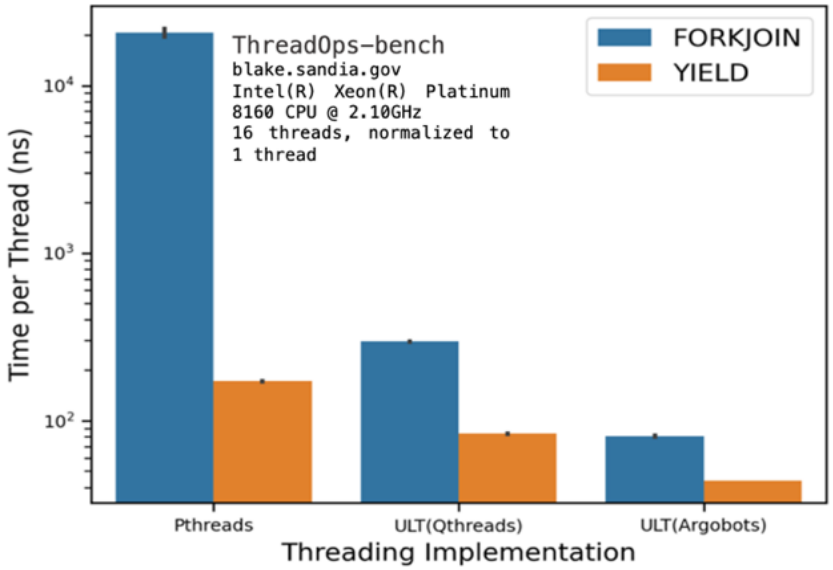
Matrix factorization (ILU, LU, CHOL), Direct Solvers, MD, Graph algorithms

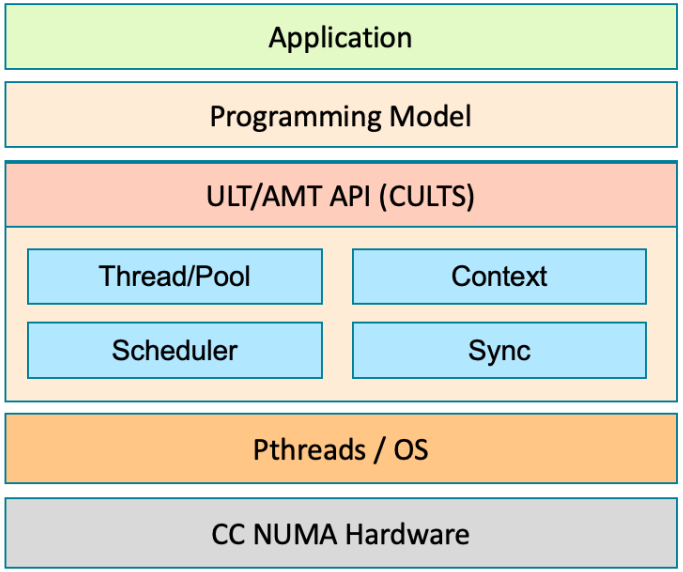


ULT for HPC Applications

Asyncs, jthreads, processes, futures, coroutines, pthreads, tasks, fibers, kernels, agents, actors, executors, senders-receivers

- We need domain-specific interfaces that target abstract machine models (a la Kokkos)
- Important: task-parallelism benefits from light-weight units of work (task)



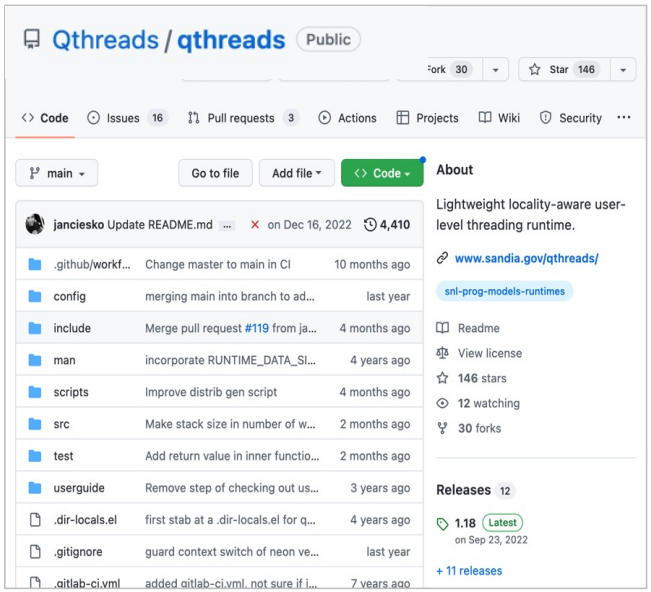


2. Multi-Processor Computing (MPC)

Qthreads

FunHPC: Functional HPC Programming

"An API for Programing with Millions of Lightweight Threads"



Publications

To cite Qthreads:

- Qthreads: An API for Programming with Millions of Lightweight Threads . Kyle Wheeler, Richard Murphy, Douglas Thain. In *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPS '08, in the MTAAP '08 workshop)*, IEEE Press, 2008.

Related Papers:

- Scheduling Chapel Tasks with Qthreads on Manycore: A Tale of Two Schedulers . Noah Evans, Stephen Olivier, Richard Barrett, George, Stelle. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2017)*, ACM Press, 2017.
- Kokkos/Qthreads task-parallel approach to linear algebra based graph analytics . Michael Wolf, H. Carter Edwards and Stephen Olivier. In *Proceedings of the 2016 IEEE High Performance Extreme Computing Conference (HPEC 2016)*, IEEE Press, 2016.
- Early Experiences Co-Scheduling Work and Communication Tasks for Hybrid MPI+X Applications . D. T. Stark, R. F. Barrett, R. E. Grant, S. L. Olivier, K. T. Pedretti and C. T. Vaughan. In *2014 Workshop on Exascale MPI (ExaMPI 2014) at SC14*, IEEE Press, 2014.
- Scheduling Task Parallelism on Multi-Socket Multicore Systems . Stephen Olivier, Allan Porterfield, Kyle Wheeler, and Jan Prins. In *Proceedings of the 25th International Conference on Supercomputing (ICS'11, in the ROSS'11 workshop)*, ACM Press, 2011.
- Implementing a Portable Multi-threaded Graph Library: the MTGL on Qthreads . Brian Barrett, Jonathan Berry, Richard Murphy, Kyle Wheeler. *In Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS '09, in the MTAAP '09 workshop)*, IEEE Press, 2009.

- 182 functions in 16 areas

- 6 concepts
  - Qthreads
  - Locality
  - FEB
  - Schedulers
  - Blocking Action
  - Context Swap

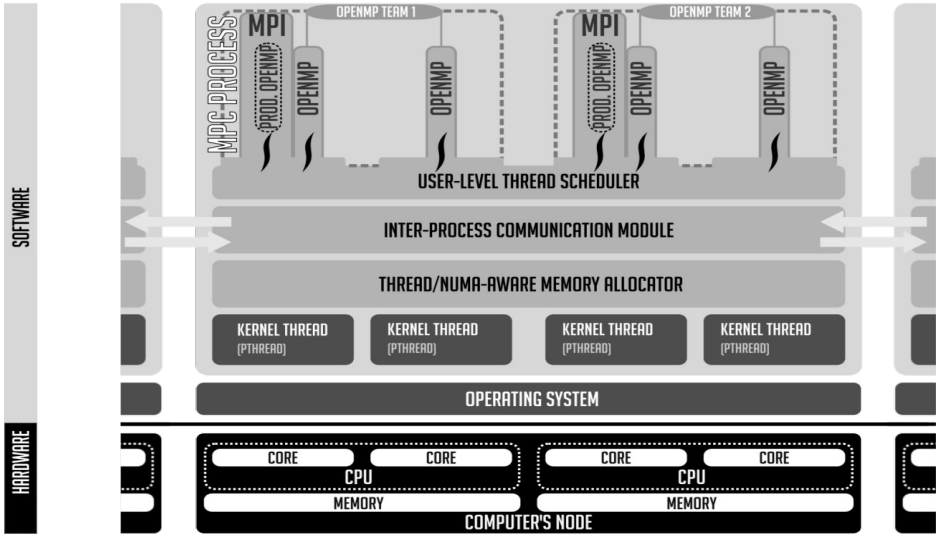
[1] https://github.com/Qthreads/qthreads
[2] https://join.slack.com/t/qthreads/

MPC Framework

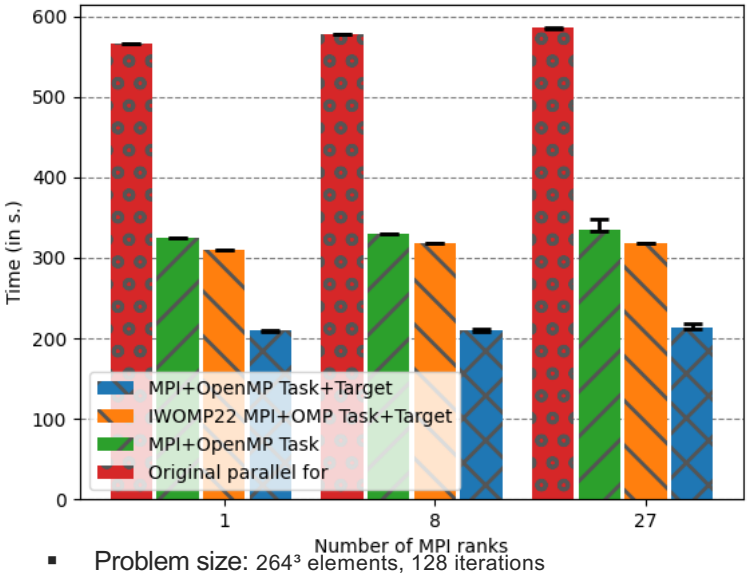
- MPC (Multi Processor Computing)
  - **Features MPI** & **OpenMP** Implementation
  - Implements user-level threading scheduler and interoperability

Multi-Processor Computing

- Architecture



- Lulesh (weak scaling)



- Problem size: 264^3 elements, 128 iterations

3. Projects

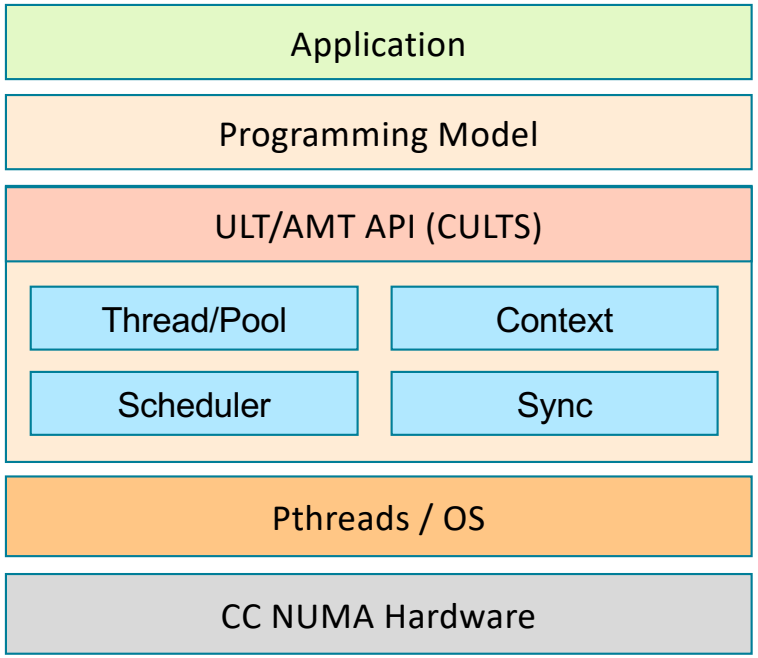
Better code: Common ULT Substrate (CULTS)

Many common building blocks among ULT/AMT libraries

- Externalize common functionality like context swap
- Improve code quality
- Develop towards common prog model frontends (C++ stdlib, HPX, Chapel)

Collaborators:

Sandia National Laboratories, cea, Argonne National Laboratory, The University of Tennessee Knoxville, LSU



Better interop: LIBULT and MPI_TASK_MULTIPLE

OpenMPI and MPICH support ULT/AMT libs

- Qthreads and Argobots are supported
- Configure options
- Developer needs matching MPI for this app
- Pulls in ULT/AMT lib specific codes

Develop libult as a generic ULT/AMT interface so MPI can be configured for ULT support (vs. Ptheads)

- Which particular ULT is used is determined at application compile time
- Potentially add MPI_TASK_MULTIPLE

Collaborators: **Others:**

Sandia National Laboratories, cea, Argonne National Laboratory, The University of Tennessee Knoxville, LSU

MPICH

```
--with-thread-package=package
--with-argobots=[PATH]
--with-argobots-include=PATH
--with-argobots-lib=PATH
```

Open MPI

```
--with-threads=TYPE      Specify thread TYPE to use. default:pthreads. Ot
                         options are qthreads and argobots.
--with-argobots=DIR      Specify location of argobots installation. Erro
                         argobots support cannot be found.
--with-argobots-libdir=DIR
                         Search for argobots libraries in DIR
--with-qthreads=DIR      Specify location of qthreads installation. Erro
                         qthreads support cannot be found.
--with-qthreads-libdir=DIR
                         Search for qthreads libraries in DIR
```

Note: Use *ompi_info --config* to see your MPI configuration

Event-orientation in MPI: MPI Continuations

Allows to attach callback functions to MPI operations

- Spec proposal to be voted on in 2023. [1][2]
- Makes OpenMP interop work (sort of)
- MPI Cont' can replace task offload to communication theads in HPX and others
- Polling vs callback in interop?

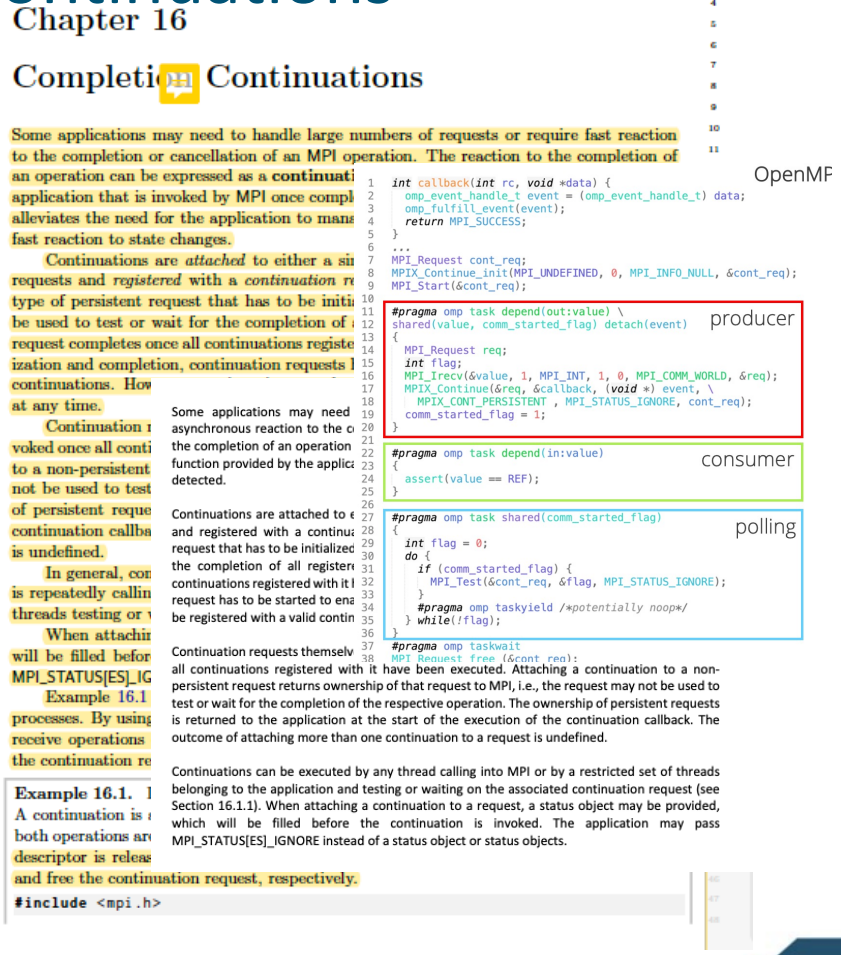
Collaborators:

Sandia National Laboratories · The University of Tennessee Knoxville · Argonne National Laboratory

ECP Deliverable 2022

https://github.com/devreal/ompi/tree/mpi-continue-master

https://github.com/mpiwg-hybrid/hybrid-issues/issues/6


Chapter 16

Completi Continuations

Some applications may need to handle large numbers of requests or require fast reaction to the completion or cancellation of an MPI operation. The reaction to the completion of an operation can be expressed as a continuati application that is invoked by MPI once compl alleviates the need for the application to mans fast reaction to state changes.

Continuations are *attached* to either a sir requests and *registered* with a *continuation* re type of persistent request that has to be initi be used to test or wait for the completion of request completes once all continuations registe ization and completion, continuation requests continuations. How at any time.

Continuation r voked once all conti to a non-persistent not be used to test of persistent reque continuation callba is undefined.

In general, con is repeatedly callin threads testing or

When attachir will be filled befor MPI_STATUS[ES]_IG

Example 16.1 processes. By using receive operations the continuation re

Example 16.1. A continuation is both operations ar descriptor is relea and free the continuation request, respectively.

```
#include <mpi.h>
```

Some applications may need asynchronous reaction to the c the completion of an operation function provided by the applic detected.

Continuations are attached to and registered with a continu request that has to be initialize the completion of all register continuations registered with it request has to be started to ena be registered with a valid contin

Continuation requests themselv all continuations registered with it have been executed. Attaching a continuation to a non-persistent request returns ownership of that request to MPI, i.e., the request may not be used to test or wait for the completion of the respective operation. The ownership of persistent requests is returned to the application at the start of the execution of the continuation callback. The outcome of attaching more than one continuation to a request is undefined.

Continuations can be executed by any thread calling into MPI or by a restricted set of threads belonging to the application and testing or waiting on the associated continuation request (see Section 16.1.1). When attaching a continuation to a request, a status object may be provided, which will be filled before the continuation is invoked. The application may pass MPI_STATUS[ES]_IGNORE instead of a status object or status objects.

OpenMP

```
int callback(int rc, void *data) {
  omp_event_handle_t event = (omp_event_handle_t) data;
  omp_fulfill_event(event);
  return MPI_SUCCESS;
}
...
MPI_Request cont_req;
MPIX_Continue_init(MPI_UNDEFINED, 0, MPI_INFO_NULL, &cont_req);
MPI_Start(&cont_req);
```

producer

```
#pragma omp task depend(out:value) \
shared(value, comm_started_flag) detach(event)
{
  MPI_Request req;
  int flag;
  MPI_Irecv(&value, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &req);
  MPIX_Continue(&req, &callback, (void *) event, \
    MPIX_CONT_PERSISTENT , MPI_STATUS_IGNORE, cont_req);
  comm_started_flag = 1;
}
```

consumer

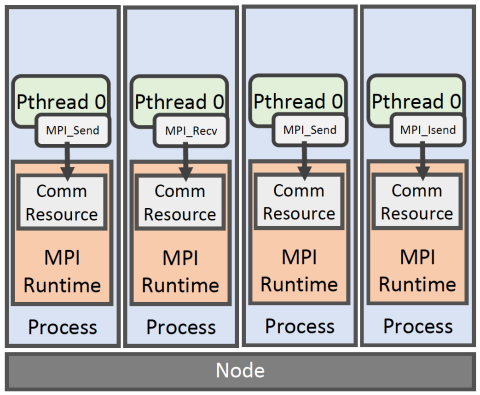
```
#pragma omp task depend(in:value)
{
  assert(value == REF);
}
```

polling

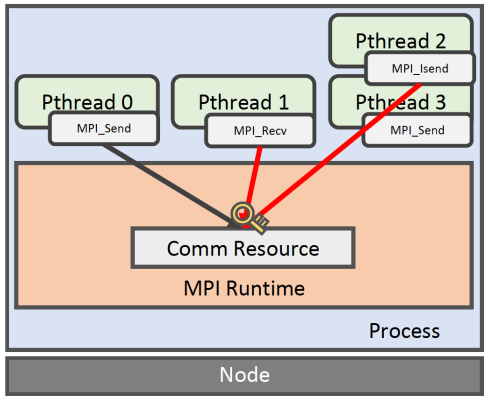
```
#pragma omp task shared(comm_started_flag)
{
  int flag = 0;
  do {
    if (comm_started_flag) {
      MPI_Test(&cont_req, &flag, MPI_STATUS_IGNORE);
    }
    #pragma omp taskyield /*potentially noop*/
  } while(!flag);
}
#pragma omp taskwait
MPI_Request_free (&cont_req);
```

VCI-(Resource-)Aware Task-scheduling

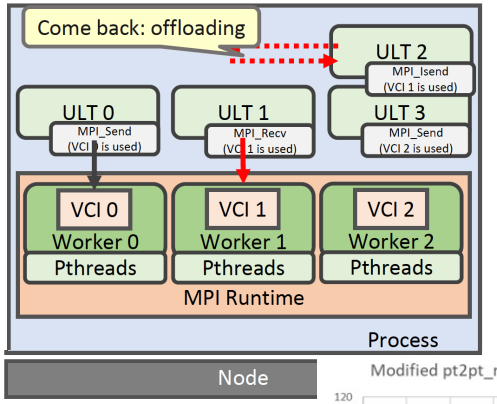
Match performance of MPI-everywhere if scheduler serialize contending tasks on VCI



MPI Everywhere (MPI_THREAD_SINGLE)



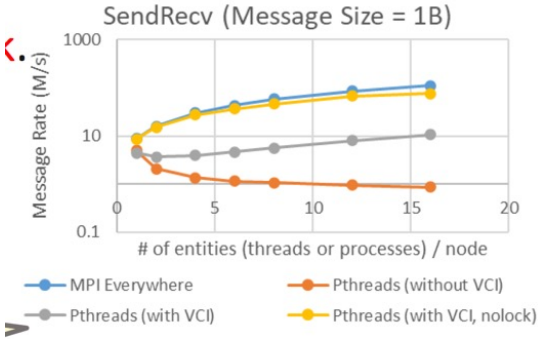
MPI_THREAD_MULTIPLE

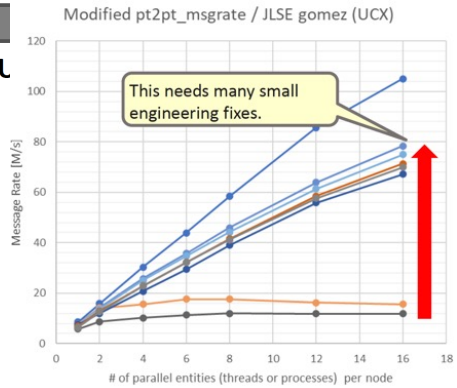


MPI_THREAD_MU

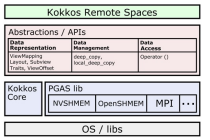
Collaborators:

Sandia National Laboratories · The University of Tennessee Knoxville · Argonne National Laboratory





Device-initiated Communication

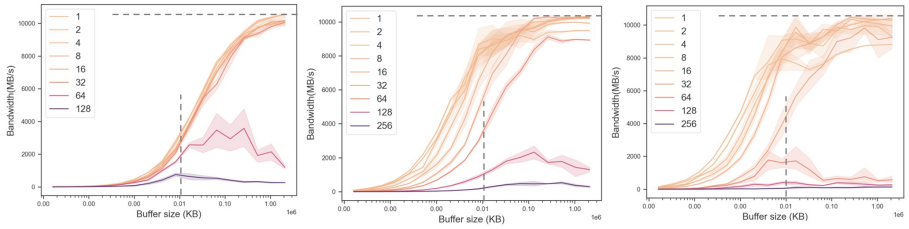


Two APIs: RMA or Partitioned Communication

- What is the usage model in a kernel?

User-level threading (ULT) + Partitioned Communication (Basic)

- MPI Partix: **"Bench1"**, 16KB-2GB buffer size, 1:1 partitions to task mapping



Blake, x86, UCX, OMPI 5.0.X, **Pthreads**, 1-128 partitions, 1:1 P2T

Blake, x86, UCX, OMPI 5.0.X, **Qthreads**, 1-256 partitions, 1:1 P2T

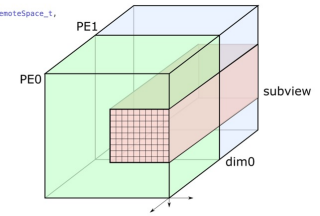
Blake, x86, UCX, OMPI 5.0.X, **OMP Task**, 1-256 partitions, 1:1 P2T

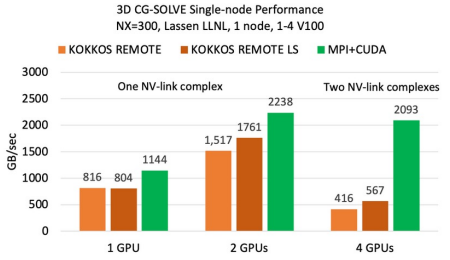
https://github.com/sandialabs/MPI-Partix

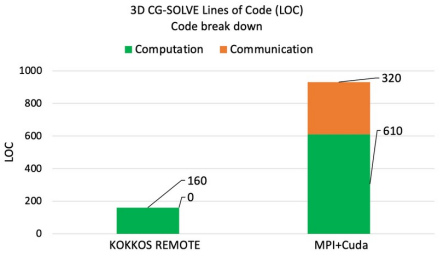
Kokkos + RMA/PC (Kokkos Remote Spaces Project)

- CGSolve

```
...
using ViewRemote_3D_t = Kokkos::View<Data_t ***, DefaultRemoteSpace_t,
  Kokkos::MemoryTraits<Kokkos::Atomic>>;

ViewRemote_3D_t view =
  ViewRemote_3D_t("RemoteView", 20, 20, 20);

auto range0                = Kokkos::ALL;
std::pair<int, int> range1 = std::make_pair(10, 20);
std::pair<int, int> range2 = std::make_pair(5, 15);

auto sub_view =
  Kokkos::subview(view, range0, range1, range2);
...

Kokkos::parallel_for("Increment",sub_view.extent(0),
KOKKOS_LAMBDA(const int j) {
  for (int k = 0; k < sub_view.extent(1); ++k)
    for (int l = 0; l < sub_view.extent(2); ++l) {
      sub_view(j, k, l)++;
    }
});
...
```







https://github.com/kokkos/kokkos-remote-spaces

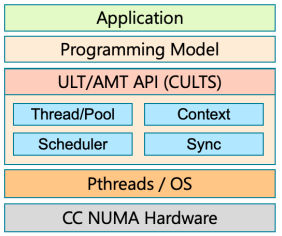
Collaborators:

Sandia National Laboratories

4. Conclusion & Future Works

Conclusion (brief)

- Roadmap ULT/AMT libs @ NNSA and CEA
  - Increase code quality and reduce maintenance
    - Common ULT Substrate (CULTS)
  - No API lock-in
    - Expose through established interfaces (OpenMP, Chapel, HPX, stdlib)
  - Play nice with others
    - Interoperability with MPI and non-cooperative APIs
  - Be relevant
    - Work on GPU stuff (RMA and PartCom on GPUs)
    - Could constrained task-execution (aka dataflow) be a thing?
    - 144 CCNUMA Cores on GH – how to use that?



MPICH OPEN MPI

CHAPEL

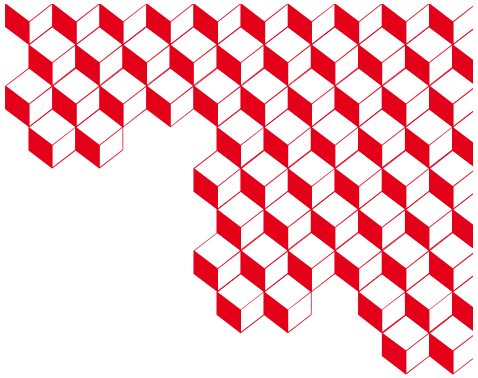
kokkos

LSU Sandia National Laboratories cea

Argonne NATIONAL LABORATORY THE UNIVERSITY OF TENNESSEE KNOXVILLE

Contact:

adrien.roussel@cea.fr, jciesko@sandia.gov



Thank you for your attention

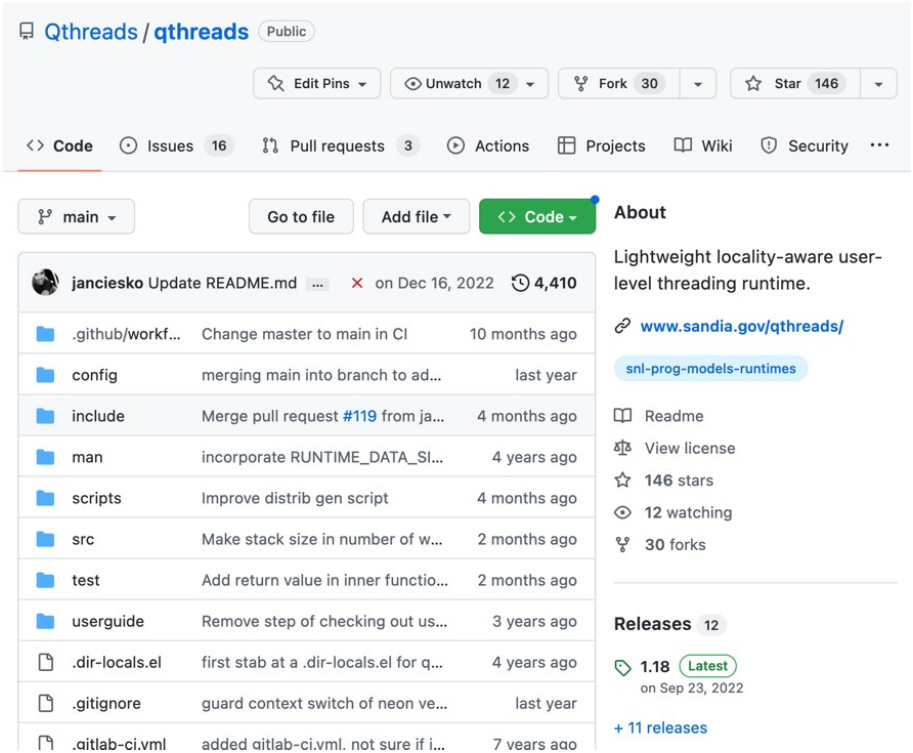
Questions?

Adrien Roussel, PhD
adrien.roussel@cea.fr
jciesko@sandia.gov
MPC Team

Backup

Qthreads

"An API for Programing with Millions of Lightweight Threads"



Publications

To cite Qthreads:

- Qthreads: An API for Programming with Millions of Lightweight Threads . Kyle Wheeler, Richard Murphy, Douglas Thain. In *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPS '08, in the MTAAP '08 workshop)*, IEEE Press, 2008.

Related Papers:

- Scheduling Chapel Tasks with Qthreads on Manycore: A Tale of Two Schedulers . Noah Evans, Stephen Olivier, Richard Barrett, George, Stelle. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2017)*, ACM Press, 2017.
- Kokkos/Qthreads task-parallel approach to linear algebra based graph analytics . Michael Wolf, H. Carter Edwards and Stephen Olivier. In *Proceedings of the 2016 IEEE High Performance Extreme Computing Conference (HPEC 2016)*, IEEE Press, 2016.
- Early Experiences Co-Scheduling Work and Communication Tasks for Hybrid MPI+X Applications . D. T. Stark, R. F. Barrett, R. E. Grant, S. L. Olivier, K. T. Pedretti and C. T. Vaughan. In *2014 Workshop on Exascale MPI (ExaMPI 2014) at SC14*, IEEE Press, 2014.
- Scheduling Task Parallelism on Multi-Socket Multicore Systems . Stephen Olivier, Allan Porterfield, Kyle Wheeler, and Jan Prins. In *Proceedings of the 25th International Conference on Supercomputing (ICS'11, in the ROSS'11 workshop)*, ACM Press, 2011.
- Implementing a Portable Multi-threaded Graph Library: the MTGL on Qthreads . Brian Barrett, Jonathan Berry, Richard Murphy, Kyle Wheeler. *In Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS '09, in the MTAAP '09 workshop)*, IEEE Press, 2009.

Qthreads API

- 182 functions in 16 areas
- 6 concepts
  - Qthreads
  - Locality
  - FEB
  - Schedulers
  - Blocking Action
  - Context Swap

Feature		API	Description
Array	1	qarray_create	allocate a runtime distributed array
	2	qarray_create_configured	allocate a runtime distributed array
	3	qarray_create_tight	allocate a runtime distributed array
	4	qarray_destroy	deallocate a distributed array
	5	qarray_dist_like	redistribute a qarray to match another qarray
	6	qarray_elem	return a pointer to a distributed array element
	7	qarray_elem_migrate	return a pointer to a distributed array element
	8	qarray_elem_nomigrate	return a pointer to a distributed array element
	9	qarray_iter	iterate through a distributed array
	10	qarray_iter_constloop	iterate through a distributed array
	11	qarray_iter_loop	iterate through a distributed array
	12	qarray_iter_loop_nb	iterate through a distributed array
	13	qarray_iter_loopaccum	iterate through a distributed array
	14	qarray_set_shepof	locate a distributed array element
	15	qarray_shepof	locate a distributed array element
Array Ops	16	qt_double_max	find the maximum value within an array in parallel
	17	qt_double_min	find the minimum value within an array in parallel
	18	qt_double_prod	multiplies an array in parallel
	19	qt_double_sum	add up an array in parallel
	20	qt_int_max	find the maximum value within an array in parallel
	21	qt_int_min	find the minimum value within an array in parallel
	22	qt_int_prod	multiplies an array in parallel
	23	qt_int_sum	add up an array in parallel
Utils	24	qutil_double_max	find the maximum value within an array in parallel
	25	qutil_double_min	find the minimum value within an array in parallel
	26	qutil_double_mult	multiply an array in parallel
	27	qutil_double_sum	add up an array in parallel
	28	qutil_int_max	find the maximum value within an array in parallel
	29	qutil_int_min	find the minimum value within an array in parallel
	30	qutil_int_mult	multiply an array in parallel
	31	qutil_int_sum	add up an array in parallel
	32	qutil_mergesort	sorts an array of doubles in parallel
	33	qutil_qsort	sorts an array of doubles in parallel
	34	qutil_uint_max	find the maximum value within an array in parallel
	35	qutil_uint_min	find the minimum value within an array in parallel
	36	qutil_uint_mult	multiply an array in parallel
	37	qutil_uint_sum	add up an array in parallel
	38	qthread_dincr	atomically increment a value
	39	qt_allpairs	computes a given function over all pairs of the input data
Queue	40	qdqueue_create	allocate a distributed queue
	41	qdqueue_dequeue	remove an element from a distributed queue
	42	qdqueue_destroy	deallocate a distributed queue
	43	qdqueue_empty	test whether a distributed queue is empty
	44	qdqueue_enqueue	append an element to a distributed queue
	45	qdqueue_enqueue_there	append an element to a distributed queue
	46	qlfqueue_create	allocate a lock-free queue
	47	qlfqueue_dequeue	remove an element from a lock-free queue
	48	qlfqueue_destroy	deallocate a lock-free queue
	49	qlfqueue_empty	test whether a lock-free queue is empty
	50	qlfqueue_enqueue	append an element to a lock-free queue
Mem pool	51	qpool_alloc	allocate memory from a distributed memory pool
	52	qpool_create	allocate a distributed memory pool
	53	qpool_create_aligned	allocate a distributed memory pool
	54	qpool_destroy	deallocate a distributed memory pool
	55	qpool_free	return a memory block to its distributed memory pool
Mmap	56	qalloc_checkpoint	locks an address, and may block
	57	qalloc_cleanup	unmaps all memory maps
	58	qalloc_dynfree	return an allocated block to the map
	59	qalloc_dynmalloc	allocates memory from the specified map
	60	qalloc_free	return an allocated block to the map
	61	qalloc_loadmap	creates or loads a map file
	62	qalloc_makedynmap	creates or loads a map file
	63	qalloc_makestatmap	creates or loads a map file
	64	qalloc_malloc	allocates memory from the specified map
	65	qalloc_statfree	return an allocated block to the map
	66	qalloc_statmalloc	allocates memory from the specified map
Sys call	67	qt_accept	
	68	qt_connect	
	69	qt_system	pass a command to the shell
	70	qt_begin_blocking_action	indicate blocking operations to the runtime
	71	qt_end_blocking_action	indicate blocking operations to the runtime
	72	qt_wait	wait for process termination
Dictionary	73	qt_dictionary_create	allocate a concurrent dictionary
	74	qt_dictionary_delete	remove a key/value pair from the dictionary
	75	qt_dictionary_destroy	deallocate a concurrent dictionary
	76	qt_dictionary_end	create an iterator pointing to the end of the dictionary
	77	qt_dictionary_get	retrieve a value from the dictionary
	78	qt_dictionary_iterator_copy	duplicate an iterator
	79	qt_dictionary_iterator_create	allocate a concurrent dictionary iterator
	80	qt_dictionary_iterator_destroy	deallocate a qt_dictionary_iterator
	81	qt_dictionary_iterator_equals	compare two iterators
	82	qt_dictionary_iterator_get	retrieves the indicated entry in the dictionary
	83	qt_dictionary_iterator_next	retrieve the current value and advance the iterator
	84	qt_dictionary_put	retrieve the current value and advance the iterator
	85	qt_dictionary_put_if_absent	insert a key/value pair into a dictionary
Timer	86	qtimer_create	allocate a qtimer
	87	qtimer_destroy	deallocate a qtimer
	88	qtimer_fastrand	generate a quick random number based on the high-resolution timer
	89	qtimer_secs	calculate the elapsed time in seconds
	90	qtimer_start	start/stop a qtimer timing
	91	qtimer_stop	start/stop a qtimer timing
Reductions	92	qt_sinc_create	allocate and/or initialize a sinc
	93	qt_sinc_destroy	deallocate and clean-up a sinc
	94	qt_sinc_expect	increase the number of expected submissions
	95	qt_sinc_fini	deallocate and clean-up a sinc
	96	qt_sinc_init	allocate and/or initialize a sinc
	97	qt_sinc_reset	restarts a sinc
	98	qt_sinc_submit	submit to a sinc
	99	qt_sinc_wait	block caller until all expected submissions are submitted
IO	100	qt_poll	synchronous I/O multiplexing
	101	qt_pread	read input
	102	qt_write	write output
	103	qt_read	read input
	104	qt_select	synchronous I/O multiplexing
Loop	105	qt_loop	implementation of a threaded loop
	106	qt_loop_balance	implementation of a threaded loop that returns values
	107	qt_loop_balance_simple	implementation of a threaded loop
	108	qt_loop_queue_addworker	adds a worker to an ongoing loop
	109	qt_loop_queue_create	allocate a loop queue handle
	110	qt_loop_queue_run	allocate a loop queue handle
	111	qt_loop_queue_run_there	implementation of a threaded loop
	112	qt_loop_queue_setchunk	a function to specify a loop chunk size
	113	qt_loop_step	implementation of a threaded loop
	114	qt_loopaccum_balance	implementation of a threaded loop that returns values
Team	115	qt_team_critical_section	marks a region of code that must not be interrupted by a eureka
	116	qt_team_eureka	signals a eureka moment
	117	qt_team_id	returns the current team's unique ID number
	118	qt_team_parent_id	returns the current team's parent's unique ID number
Thread	119	qthread_finalize	terminate all qthreads and free library data stuctures
	120	qthread_disable_shepherd	enable or disable a shepherd for thread scheduling
	121	qthread_disable_worker	enable or disable a worker for thread scheduling
	122	qthread_distance	returns the distance between shepherds
	123	qthread_debuglevel	returns status information from the runtime
	124	qthread_cacheline	returns the cache line size
	125	qthread_init	initialize the qthread library
	126	qthread_initialize	initialize the qthread library
	127	qthread_migrate_to	relocate a qthread to the target shepherd
	128	qthread_num_shepherds	returns the number of shepherds
	129	qthread_num_workers	returns the number of workers
	130	qt_system	pass a command to the shell
	131	qthread_yield	release the CPU, allow other qthreads to run
	132	qthread_worker	returns the qthread's current worker's stable ID
	133	qthread_worker_unique	returns the qthread's current worker ID
	134	qthread_stackleft	returns the amount of space left in the thread's allocated stack
	135	qthread_spawn	spawn a qthread (task)
	136	qthread_readstate	returns status information from the runtime
	137	qthread_retloc	returns pointer to qthread's return location
	138	qthread_shep	returns the qthread's shepherd ID
	139	qthread_shep_ok	check on whether a thread's shepherd is disabled
	140	qthread_size_tasklocal	returns the number of bytes allocated for task-local data
	141	qthread_sorted_sheps	returns a list of other shepherds, sorted by their distance
	142	qthread_sorted_sheps_remote	returns a list of other shepherds, sorted by their distance
	143	qthread_fork	spawn a qthread
	144	qthread_fork_precond	spawn a qthread
	145	qthread_fork_precond_to	spawn a qthread
	146	qthread_fork_syncvar	spawn a qthread
	147	qthread_fork_syncvar_to	spawn a qthread
	148	qthread_fork_to	spawn a qthread
	149	qthread_enable_shepherd	enable or disable a shepherd for thread scheduling
	150	qthread_enable_worker	enable or disable a worker for thread scheduling
FEB	151	qthread_fill	sets the given address to either "empty" or "full"
	152	qthread_readFE	waits for the source to be full, then copies and empties it
	153	qthread_readFF	waits for the source to be full, then copies it
	154	qthread_writeEF	waits for the dest to be empty, then fills it
	155	qthread_writeEF_const	waits for the dest to be empty, then fills it
	156	qthread_writeF	fills an address with data
	157	qthread_writeF_const	fills an address with data
	158	qthread_syncvar_empty	sets the given syncvar to either "empty" or "full"
	159	qthread_syncvar_fill	sets the given syncvar to either "empty" or "full"
	160	qthread_syncvar_readFE	waits for the syncvar to be full, then copies its data and empties
	161	qthread_syncvar_readFF	waits for the syncvar to be full, then copies its data
	162	qthread_syncvar_status	return the full/empty status of a syncvar_t
	163	qthread_syncvar_writeEF	waits for the dest syncvar to be empty, then fills it
	164	qthread_syncvar_writeEF_const	waits for the dest syncvar to be empty, then fills it
	165	qthread_syncvar_writeF	fills a syncvar_t with data
	166	qthread_syncvar_writeF_const	fills a syncvar_t with data
	167	qthread_feb_barrier_create	manipulates FEB-based barriers
	168	qthread_feb_barrier_destroy	manipulates FEB-based barriers
	169	qthread_feb_barrier_enter	manipulates FEB-based barriers
	170	qthread_feb_barrier_resize	manipulates FEB-based barriers
	171	qthread_feb_status	return the full/empty status of an address
	172	qthread_lock	lock or unlock an address
	173	qthread_unlock	lock or unlock an address
	174	qthread_cas	atomically compare-and-swap a value
	175	qthread_cas_ptr	atomically compare-and-swap a value
	176	qthread_empty	sets the given address to either "empty" or "full"
Queue	177	qthread_queue_create	allocate and/or initialize a synchronization queue
	178	qthread_queue_destroy	deallocate a synchronization queue
	179	qthread_queue_join	enter a queue and block until the queue is released
	180	qthread_queue_length	return the length of the queue
	181	qthread_queue_release_all	activate the tasks in the queue for scheduling and execution
	182	qthread_queue_release_one	activate the tasks in the queue for scheduling and execution

Qthreads Usage

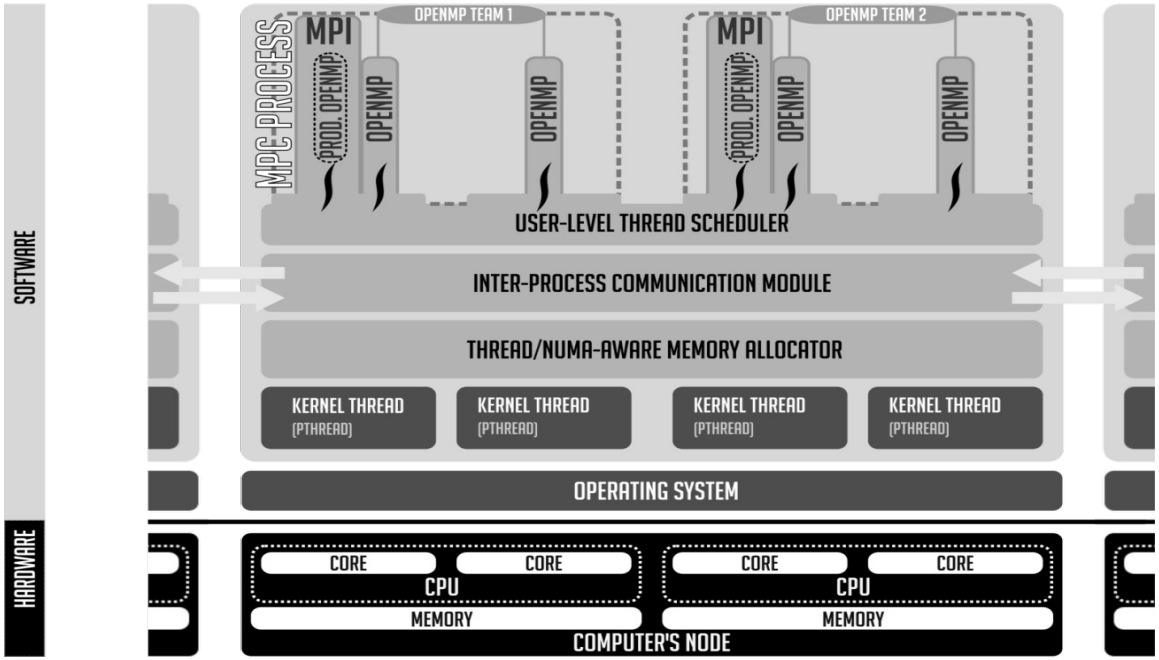
- Supported on most platforms
- One sub release per year (v1.18 latest)
- 10 papers published
- GitHub Repo [1]: 146 Stars, 30 clones pM, 100 views pM
- Package includes man pages, unit tests, examples and benchmarks,
- Slack channel [2]

Prominent use cases:

[1] https://github.com/Qthreads/qthreads
[2] https://join.slack.com/t/qthreads/

MPC Framework

- MPC (Multi Processor Computing)
  - **Features MPI** & **OpenMP** Implementation
  - Implements user-level threading scheduler and interoperability
- Task-based parallelism
  - Low overhead
  - Better interoperability
    - MPI
    - GPU
  - Scheduling flexibility



Code Example: LULESH

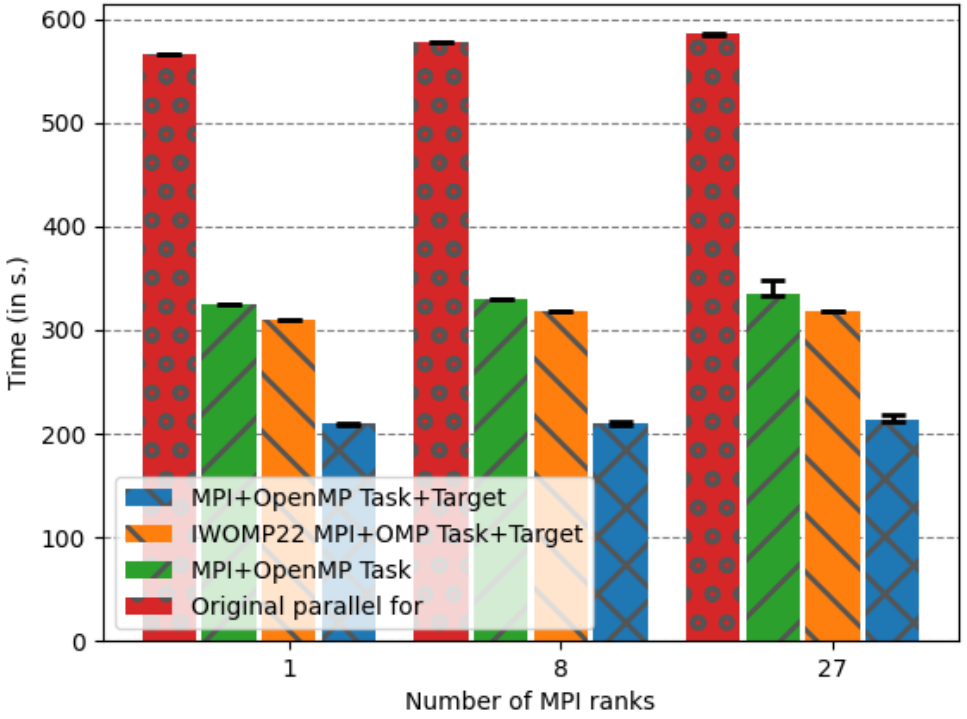
- MPI+OpenMP Implementation
  - MPI Encapsulated within OpenMP Tasks
  - CPU ←→ GPU Transfers through “target data update”
- Cooperative Communication Progression
  - Task switching until communication is completed
  - Avoid busy waiting in tasks

```
1  # pragma omp task depend(inout: x)
2  {
3      /* some work that may write x   */
4      MPI_Isend(x, ..., req);
5      /* some work independent from x */
6      MPI_Wait(req);
7      /* some work that may write x   */
8  }
9
10 # pragma omp task
11 { /* some independent work */ }
```

```
/* Internal MPI request progression */
int MPIX_Progress(MPI_Request * request)
{
    int completed;
    MPI_Test(request, &completed);
    if (completed)
    {
        omp_fulfill_event(request->omp_event);
    }
}

int MPI_Wait(MPI_Request * request, MPI_Status * status)
{
    if (omp_in_explicit_task())
    {
        /* notifies OMP that we are within a send-task */*
        omp_in_send_task();

        /* block current task until the event is fulfilled */
        omp_task_block(request->omp_event);
    }
    else
    {
        /* Standard MPI_Wait implementation */
        [...]
    }
}
```

LULESH Performance Evaluation

- Problem size
  - 264³ elements
    - 80% of the NUMA domain memory capacity
    - 20% of the GPU memory capacity
  - 128 iterations
- Software environment
  - MPC-OpenMP with LLVM 16.x for GPU offloading
    + OpenMPI 4.0.5
- Placement
  - 16 cores + 1 GPU per MPI rank
- Weak scaling
  - Median runtime of 10 runs
  - Each version scale very well
  - OpenMP tasks version significantly improves performances due to cache reuse
  - Significant performance gain with GPU



	Median time (in s.)			
ranks	parallel-for	no offloading	IWOMP22	offloading
1	565.84	325.02	309.14	209.31
8	572.45	331.09	317.91	209.43
27	581.26	333.98	318.83	212.78