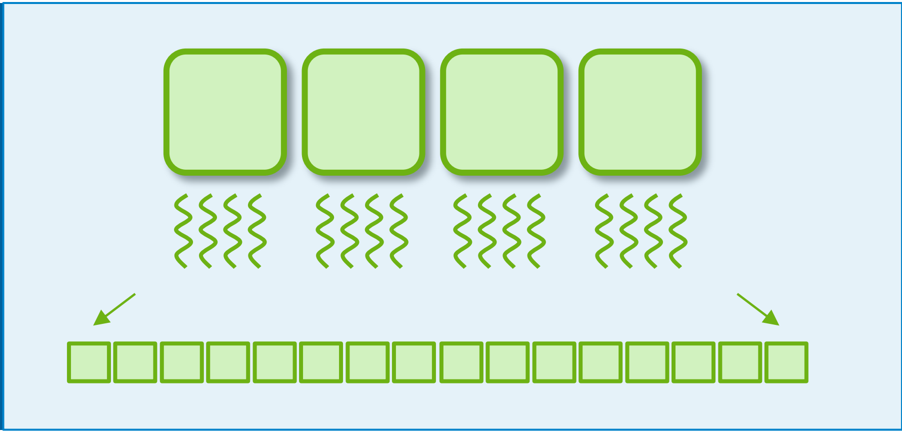
FRUSTRATED WITH MPI+THREADS? TRY MPI×THREADS!



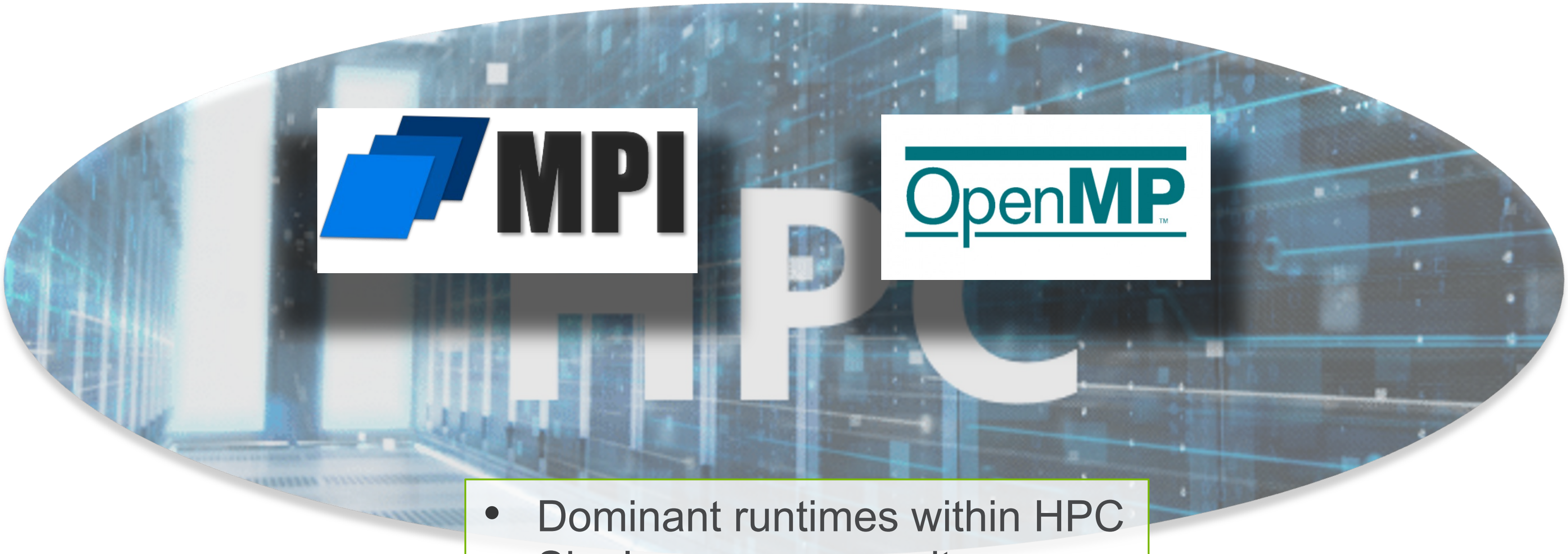
EUROMPI 23

11-13 Sept. 2023, Bristol, UK

Hui Zhou, Ken Raffenetti, Junchao Zhao, Yanfei Guo, and Rajeev Thakur
Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

INTRODUCTION



- Dominant runtimes within HPC
- Single user community
- Split research community

PARALLEL COMPUTING

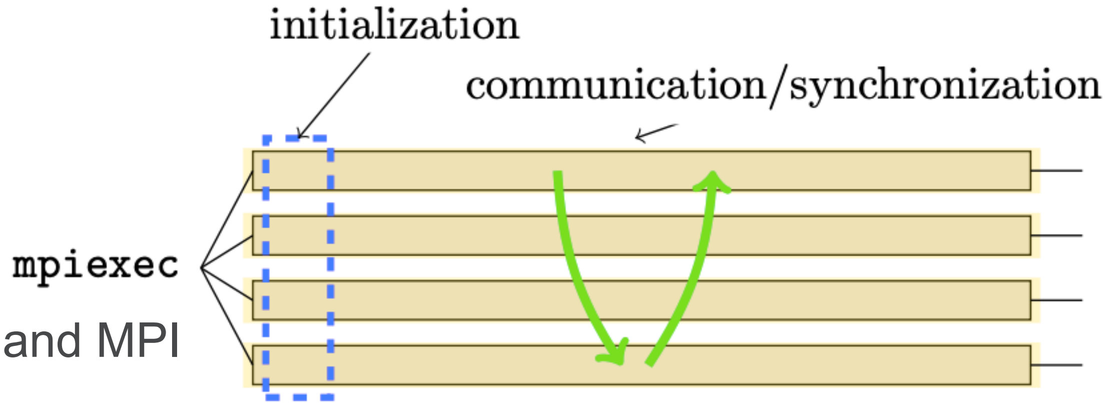
3 important aspects

- Programmability
- Environment
- Synchronization

SPMD

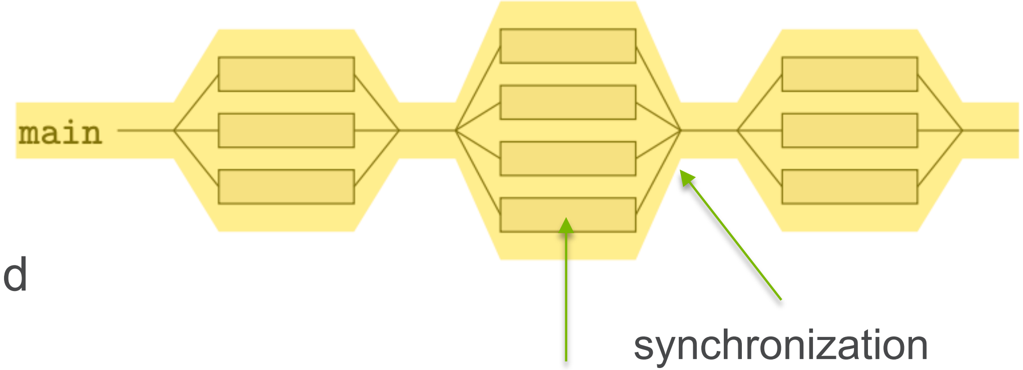
MPI

- External launcher
  - Barrier between users and MPI
  - Unspecified and specific
- Private variable space
  - Free of race conditions and false sharing
  - Message passing
  - Rich API, efficient and flexible synchronizations
- Point-to-point



dominant style: point-to-point

OPENMP



load/store

dominant style: one-sided

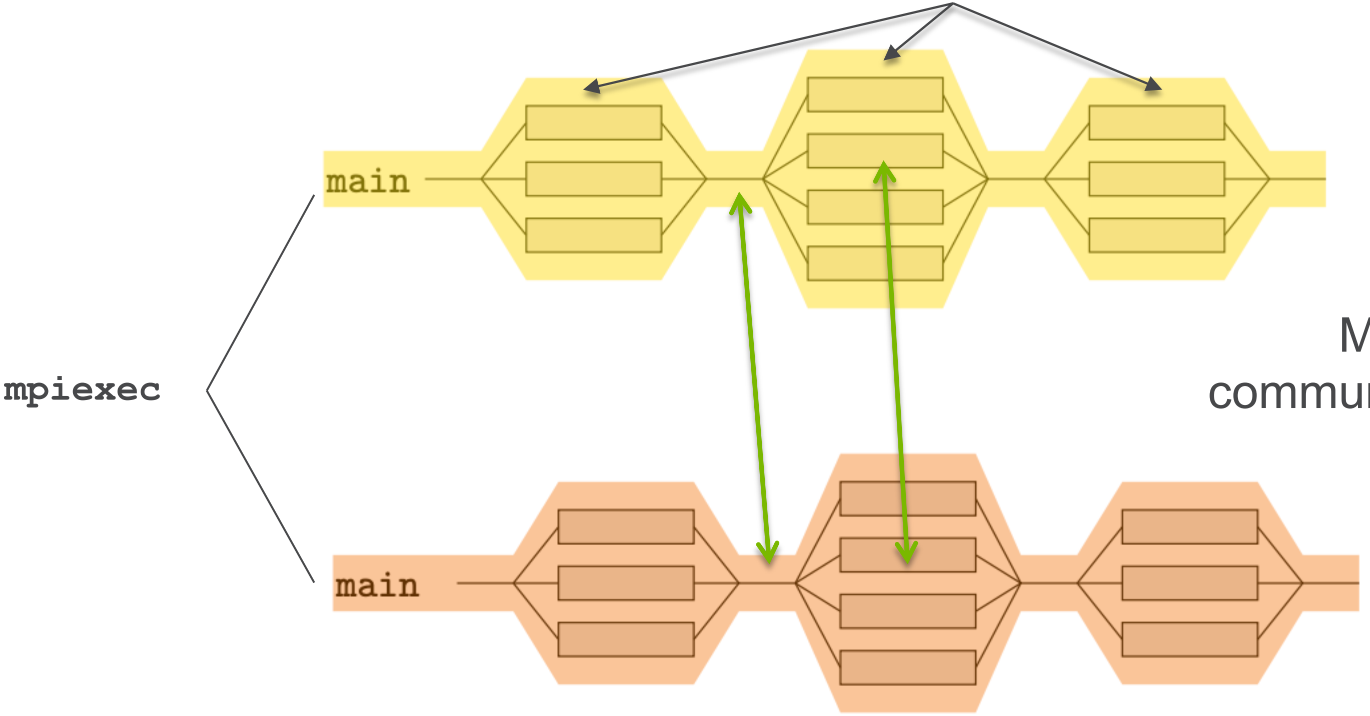
- Parallel regions on-demand
  - Lightweight, dynamic
  - Limited to on-node environment
- Shared variable space
  - Susceptible to data race / false sharing
  - Bulk synchronous pattern
- One-sided load/store
  - Resembles MPI's RMA w. fence synchronization

MPI & OPENMP IN A TABLE

	MPI	OpenMP
Programability	SPMD ✓	SPMD ✓
Environment	static, processes, cluster	dynamic, threads, on-node
Synchronization	rich patterns pt2pt, collective, rma Nonblocking, persistent	single pattern bulk sync + one-sided

MPI+THREADS

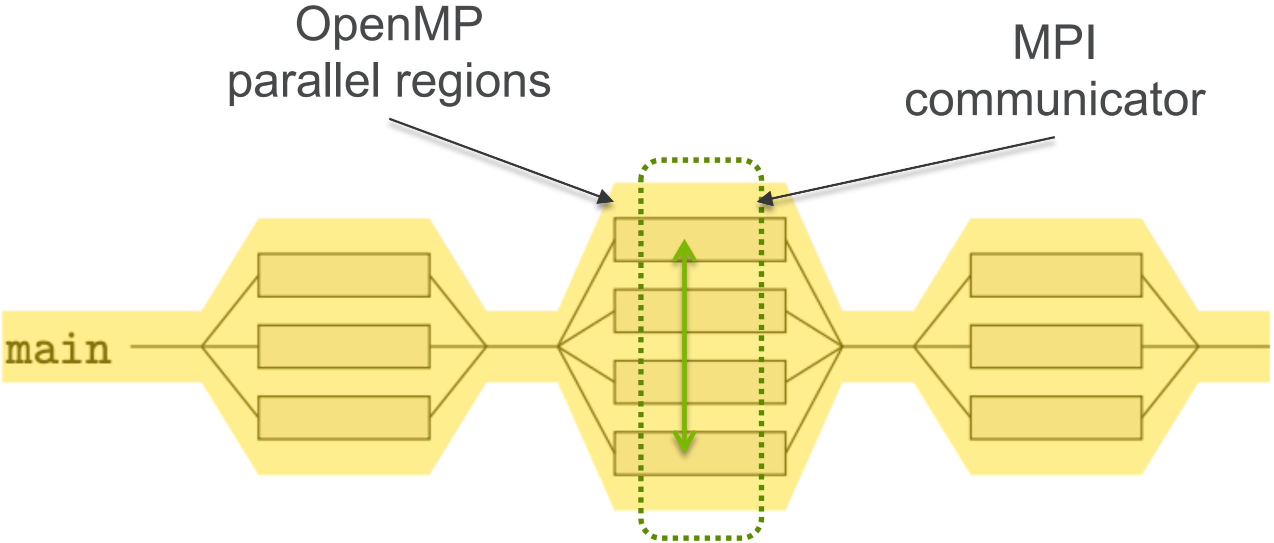
OpenMP
parallel regions



MPI
communications

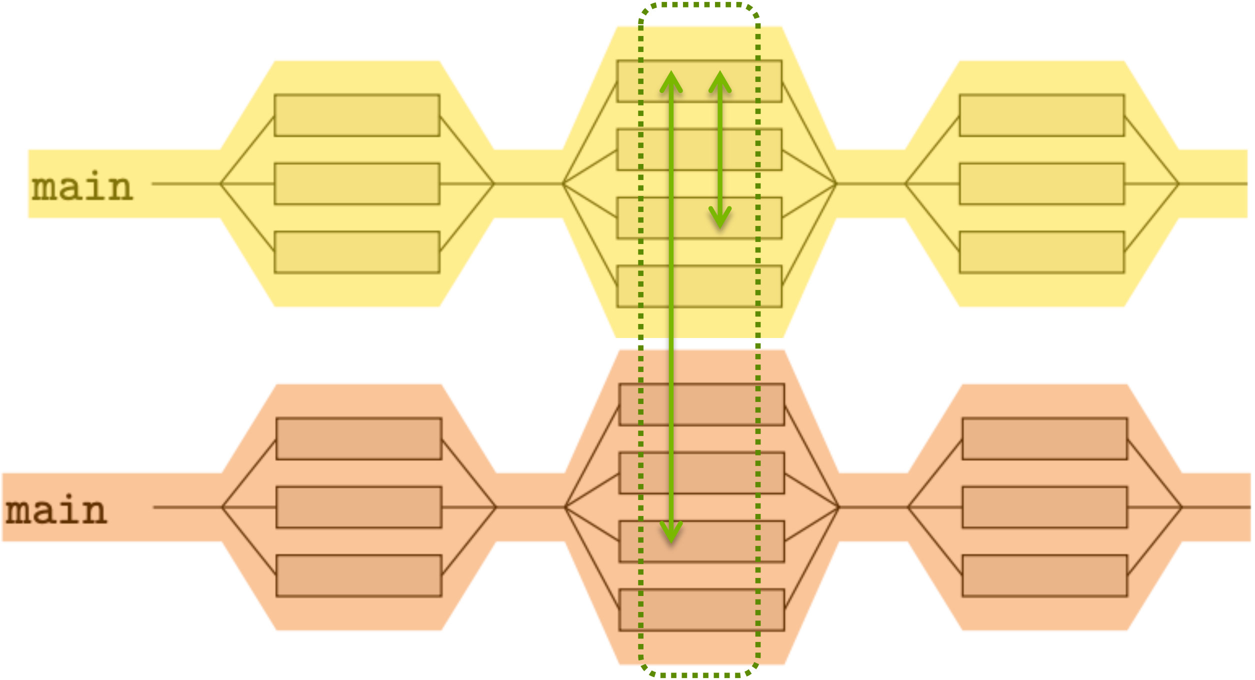
OPENMP'S PARALLEL REGIONS + MPI'S RICH COMMUNICATIONS

- Pick MPI's good parts and add to where OpenMP is lacking

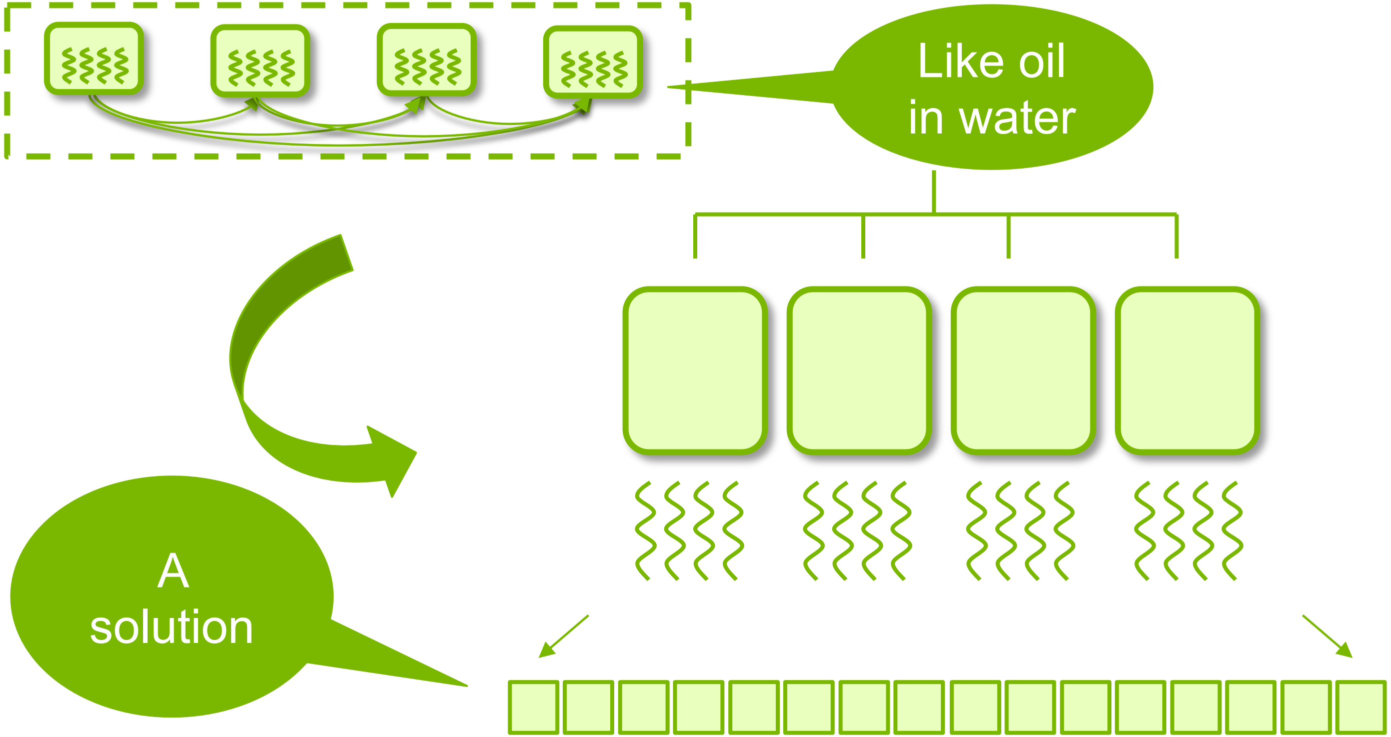


MPI × THREADS

- Pick MPI’s distributed parallel environment



FROM MPI + THREADS TO MPI × THREADS



Like oil in water

A solution

MPI

Threads

Threadcomm

MPIX THREAD COMMUNICATOR

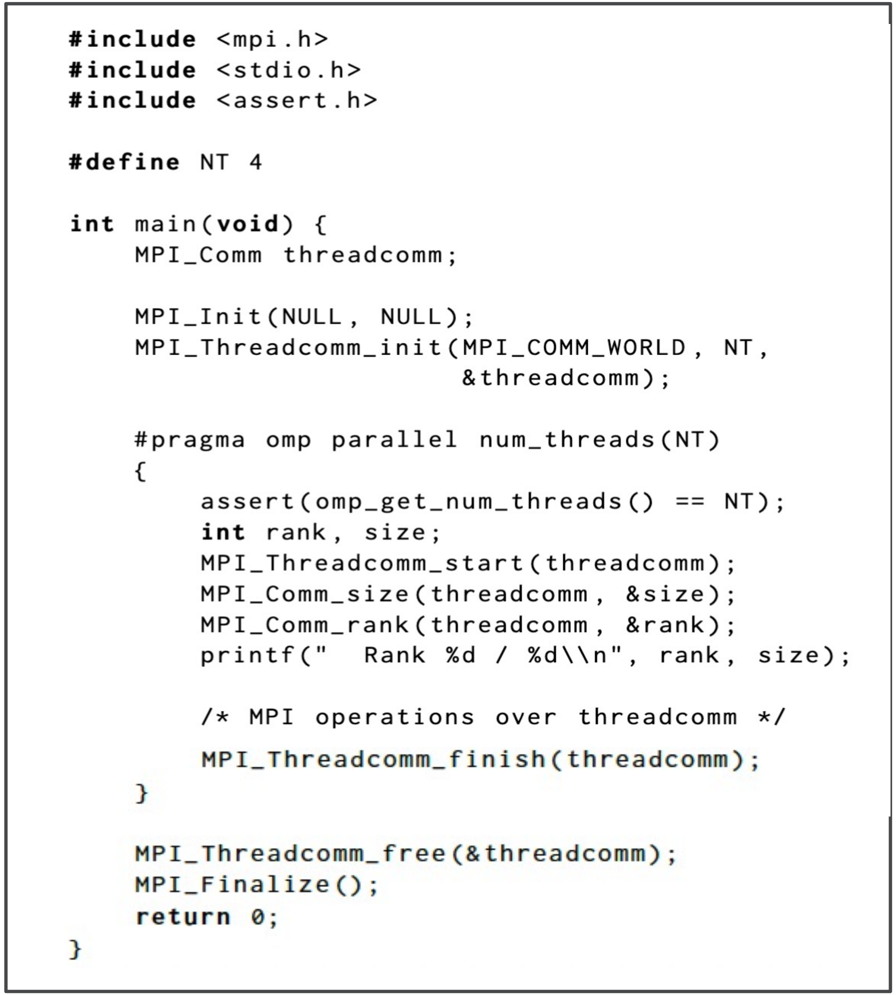
- Synopsis

```
int MPIX_Threadcomm_init(MPI_Comm comm, int num_threads,
                          MPI_Comm threadcomm)
```

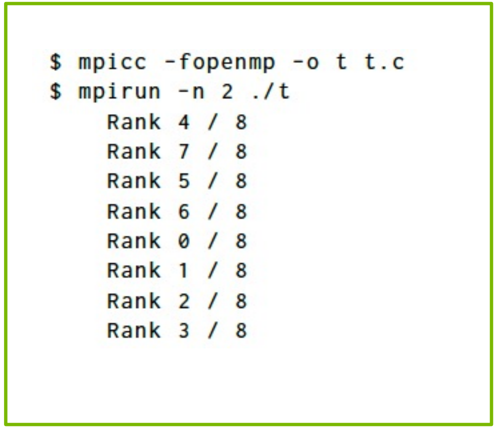
```
#pragma omp parallel {
    MPIX_Threadcomm_start(threadcomm);
    /* use threadcomm within parallel region */
    MPIX_Threadcomm_finish(threadcomm);
}
```

```
int MPIX_Threadcomm_free(MPI_Comm *threadcomm)
```

EXAMPLE



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

#define NT 4

int main(void) {
    MPI_Comm threadcomm;

    MPI_Init(NULL, NULL);
    MPI_Threadcomm_init(MPI_COMM_WORLD, NT,
                        &threadcomm);

    #pragma omp parallel num_threads(NT)
    {
        assert(omp_get_num_threads() == NT);
        int rank, size;
        MPI_Threadcomm_start(threadcomm);
        MPI_Comm_size(threadcomm, &size);
        MPI_Comm_rank(threadcomm, &rank);
        printf("  Rank %d / %d\\n", rank, size);

        /* MPI operations over threadcomm */

        MPI_Threadcomm_finish(threadcomm);
    }

    MPI_Threadcomm_free(&threadcomm);
    MPI_Finalize();
    return 0;
}
```



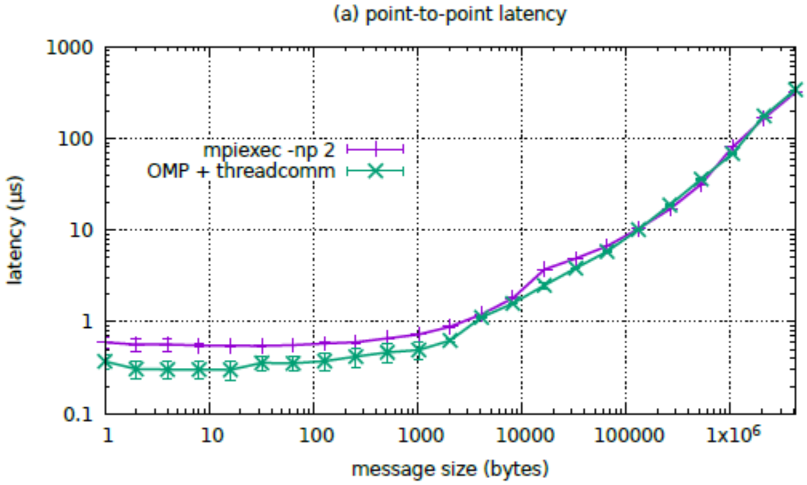
```
$ mpicc -fopenmp -o t t.c
$ mpirun -n 2 ./t
   Rank 4 / 8
   Rank 7 / 8
   Rank 5 / 8
   Rank 6 / 8
   Rank 0 / 8
   Rank 1 / 8
   Rank 2 / 8
   Rank 3 / 8
```

MPI THREAD LEVEL

- MPI_THREAD_SINGLE
- MPI_THREAD_FUNNELED
- MPI_THREAD_SERIALIZED
- MPI_THREAD_MULTIPLE

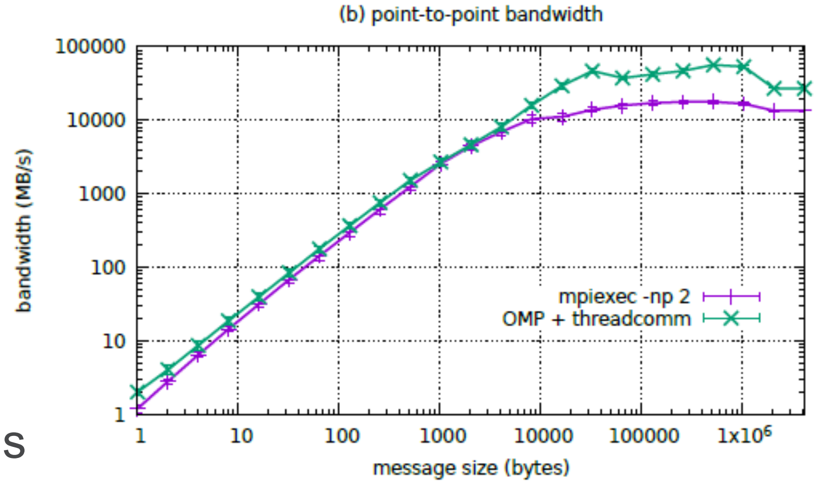
- What thread level should threadcomm use?
  - Uses thread – obviously
  - But a single execution context per assigned rank
- Why do we need MPI thread level?
  - MPI can’t tell thread contexts in MPI+Threads
- Why threadcomm does not need MPI thread level?
  - Threadcomm always can tell about the thread context!

LATENCY AND BANDWIDTH



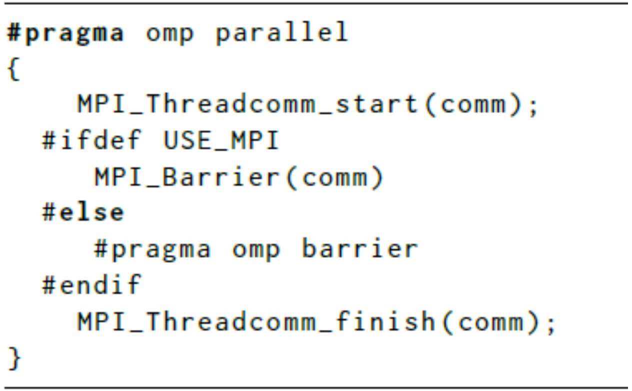
MPI on threads *VS*
MPI on processes

Can threadcomm replace flat-MPI?

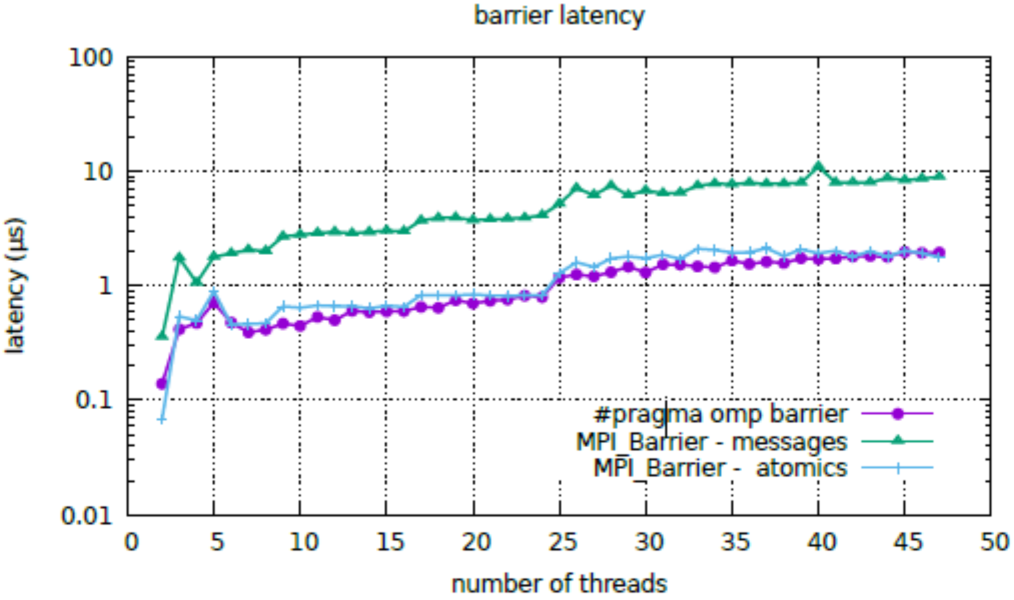


- Only technical difference
- No fundamental difference
- See paper for detailed discussions

BARRIER

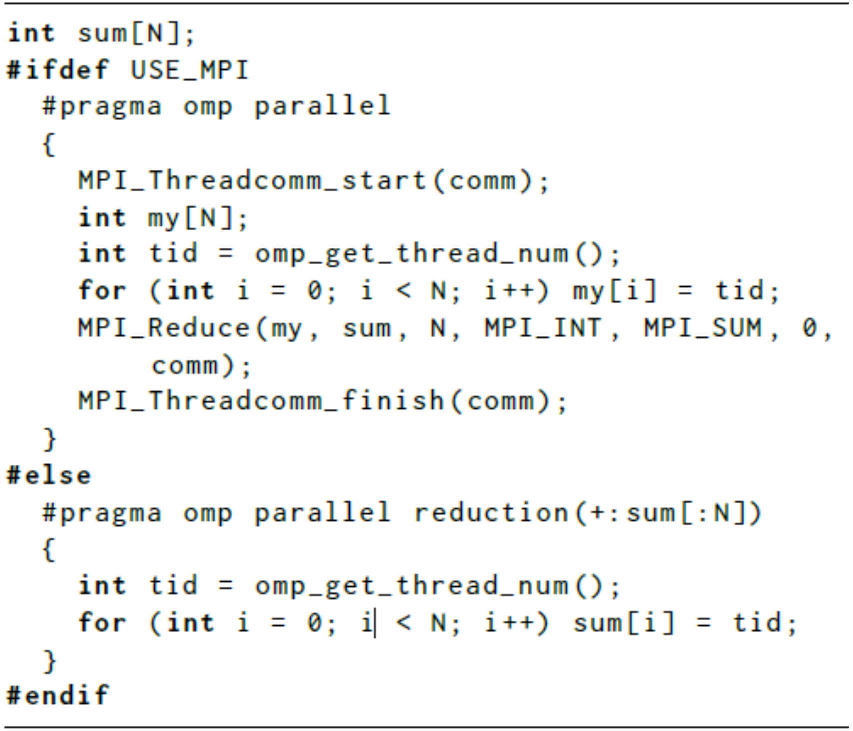


```
#pragma omp parallel
{
    MPI_Threadcomm_start(comm);
  #ifdef USE_MPI
     MPI_Barrier(comm)
  #else
     #pragma omp barrier
  #endif
    MPI_Threadcomm_finish(comm);
}
```

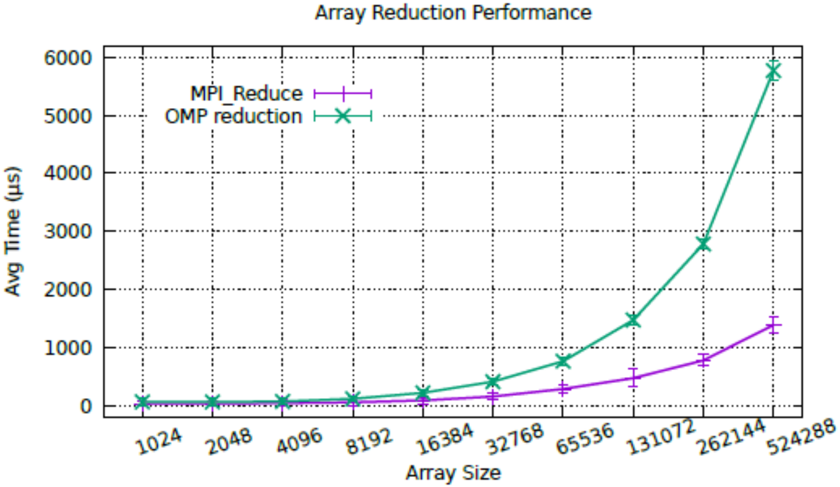


Are MPI's APIs useable for OpenMP?

REDUCTION

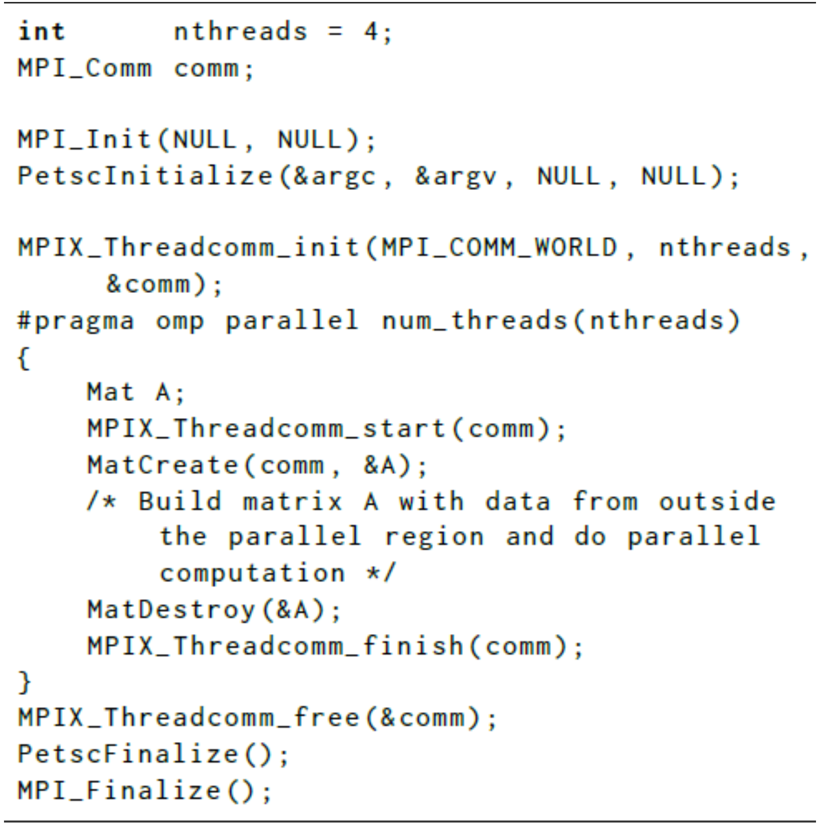


```
int sum[N];
#ifdef USE_MPI
  #pragma omp parallel
  {
    MPI_Threadcomm_start(comm);
    int my[N];
    int tid = omp_get_thread_num();
    for (int i = 0; i < N; i++) my[i] = tid;
    MPI_Reduce(my, sum, N, MPI_INT, MPI_SUM, 0,
        comm);
    MPI_Threadcomm_finish(comm);
  }
#else
  #pragma omp parallel reduction(+:sum[:N])
  {
    int tid = omp_get_thread_num();
    for (int i = 0; i < N; i++) sum[i] = tid;
  }
#endif
```



Are MPI's API good for OpenMP?

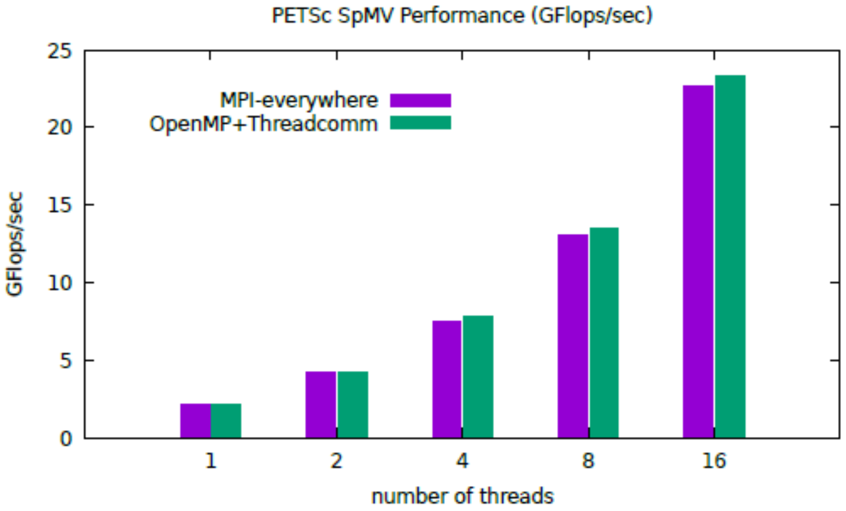
USING PETSC WITH THREADCOMM

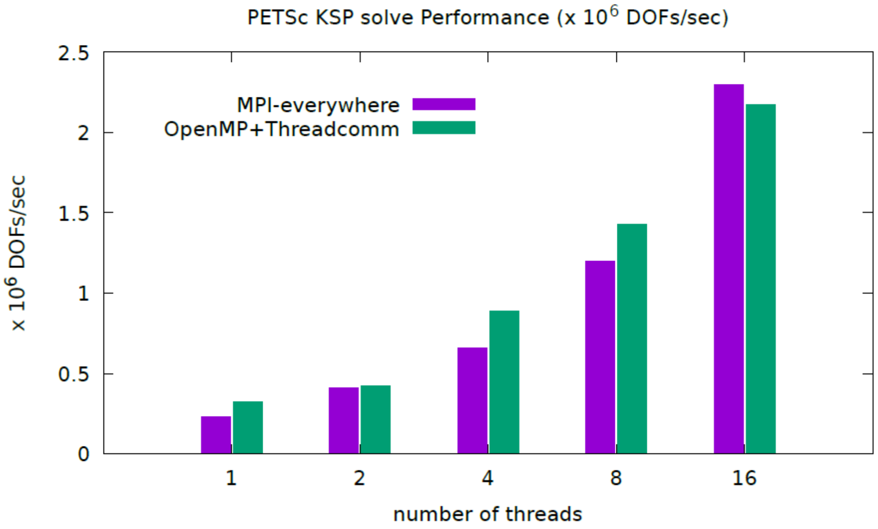


```
int      nthreads = 4;
MPI_Comm comm;

MPI_Init(NULL, NULL);
PetscInitialize(&argc, &argv, NULL, NULL);

MPIX_Threadcomm_init(MPI_COMM_WORLD, nthreads,
     &comm);
#pragma omp parallel num_threads(nthreads)
{
    Mat A;
    MPIX_Threadcomm_start(comm);
    MatCreate(comm, &A);
    /* Build matrix A with data from outside
        the parallel region and do parallel
        computation */
    MatDestroy(&A);
    MPIX_Threadcomm_finish(comm);
}
MPIX_Threadcomm_free(&comm);
PetscFinalize();
MPI_Finalize();
```

- PETSc is not thread-safe
  - Use thread-local storage
  - Global init, then read-only
  - Logging and debugging
    - Need mutexes
    - Need threadcomm-aware
- The lessons apply to all MPI-only applications
- The changes required by adaptation are minimal

PETSC+THREADCOMM PERFORMANCE





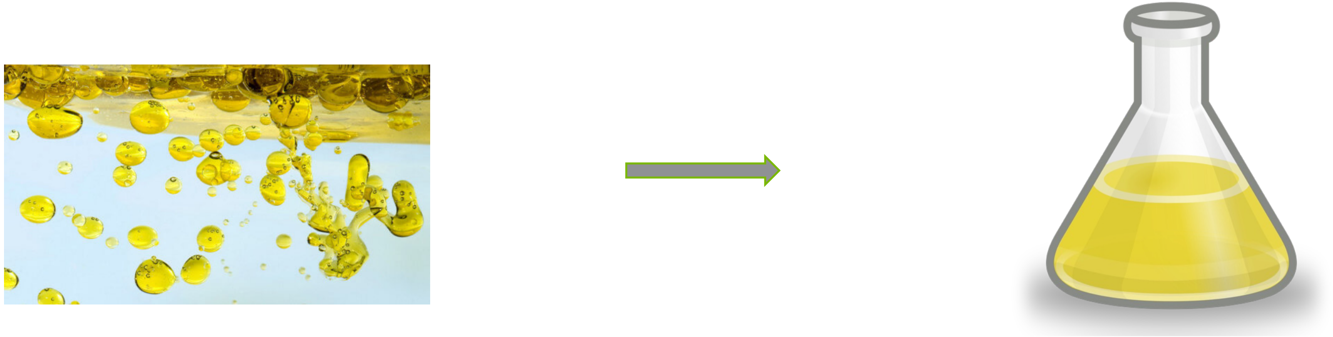
U.S. DEPARTMENT OF ENERGY

Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY

SUMMARY

- MPI + Threads is a compromise like mixing oil in water
- MPI x Threads is a solution makes MPI and OpenMP work together
- New proposal, MPIX Threadcomm, to enable MPI x Threads
- Thread communicator will be available in MPICH-4.2, to be released this year



Q & A