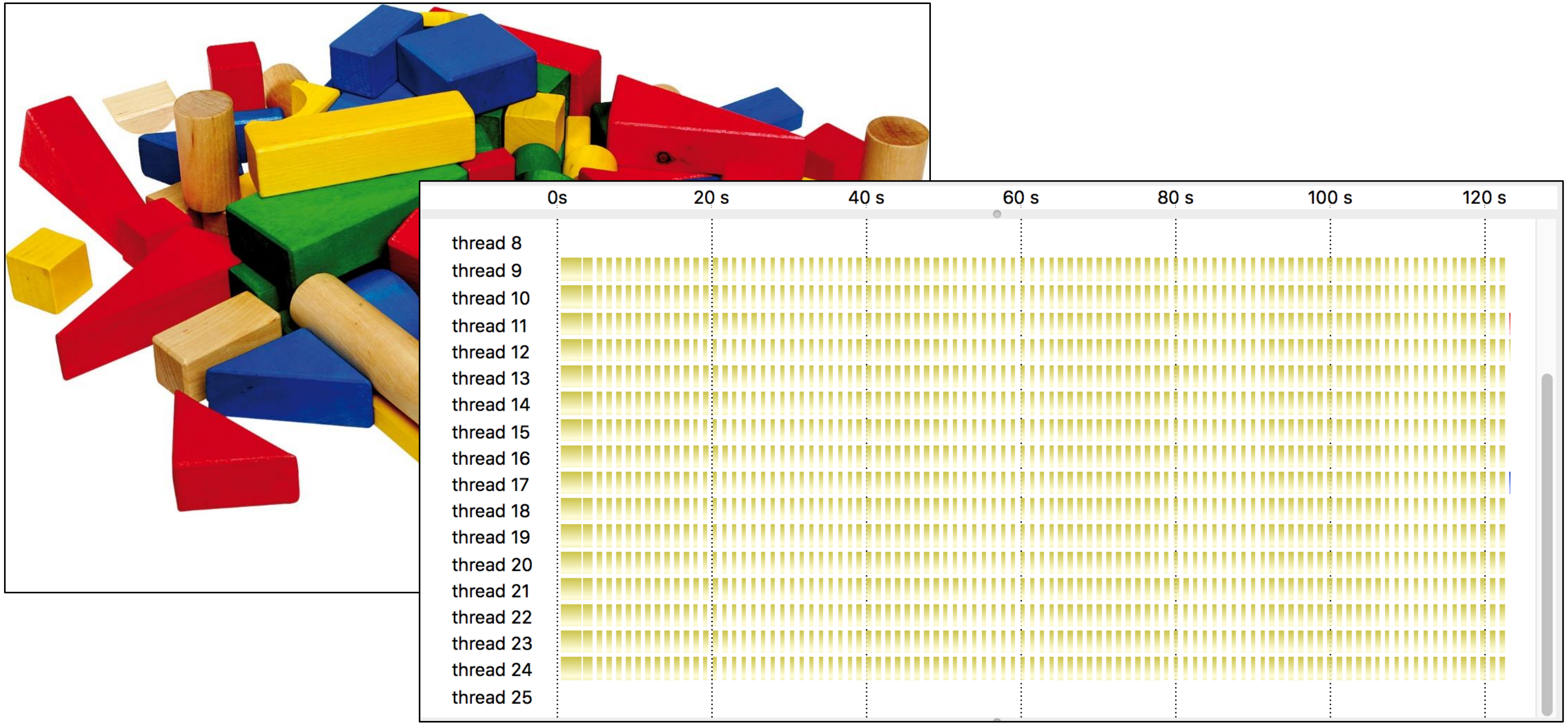
HPX
A C++ Library for Parallelism and Concurrency

Hartmut Kaiser (hkaiser@cct.lsu.edu)

WAMTA, February 15, 2023

Todays Application Problems



HPX

The C++ Standards Library for Concurrency and Parallelism

https://github.com/STEllAR-GROUP/hpx

STE||AR GROUP

HPX – A Distributed Asynchronous Many-task Runtime System

- At it’s heart, HPX is a very efficient threading implementation
- Several functional layers are implemented on top:
  - C++ standards-conforming API exposing everything related to parallelism and concurrency
  - Full set of C++17/C++20/C++23 (parallel) algorithms
    - One of the first full openly available implementations
    - Extensions:
      - Asynchronous execution of algorithms
      - Parallel range based algorithms
      - Auto vectorization execution polices `unseq/par_unseq`
      - Explicit vectorization execution policies `simd/par_simd`

STE||AR GROUP

HPX – A Distributed Asynchronous Many-task Runtime System

- At it's heart, HPX is a very efficient threading implementation
- Several functional layers are implemented on top:
  - Uniform integration of your Kokkos, CUDA, HIP, and SYCL (oneAPI) kernels
  - Full set of senders/receivers (currently being discussed for standardization)
    - Implemented using C++17
  - Distributed operation
    - Extending the standard interfaces for use on tightly coupled clusters (super-computers)
    - Global address space, load balancing, uniform API for local and remote operations

STE||AR GROUP

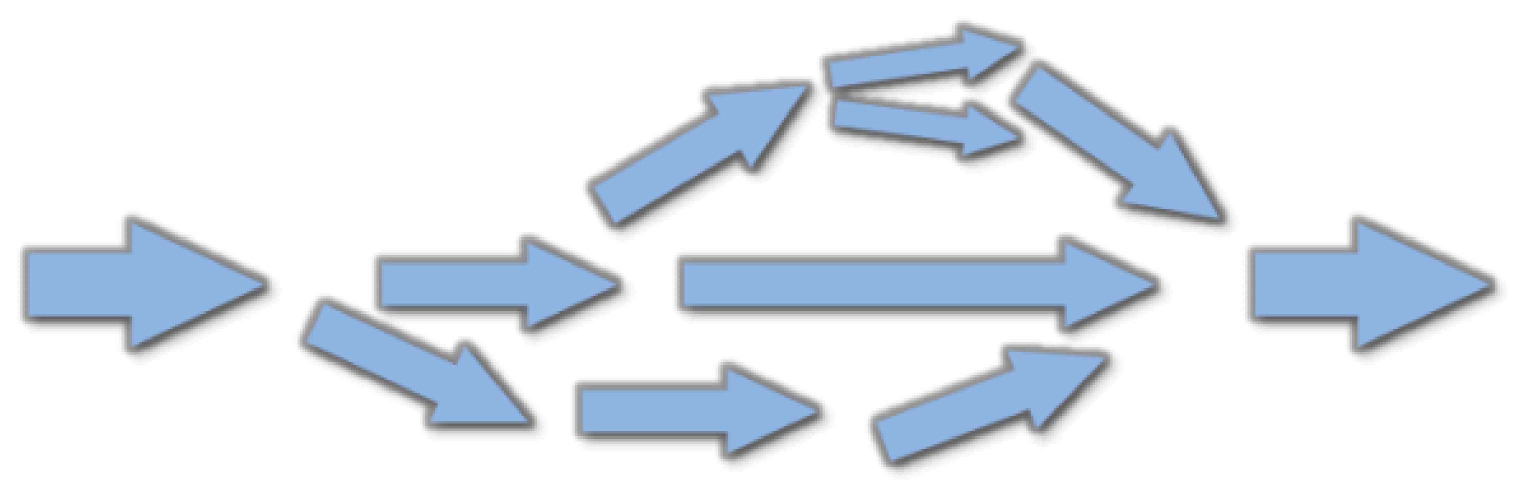
HPX – The C++ Standards Library for Concurrency and Parallelism

- Exposes a coherent and uniform, standards-oriented API
  - Ease of programming of parallel, distributed, and heterogeneous applications.
    - Enables to write fully asynchronous code using hundreds of millions of threads.
    - Provides unified syntax and semantics for local and remote operations.
- Enables using the Asynchronous C++ Standard Programming Model
  - Emergent auto-parallelization, intrinsic hiding of latencies,

HPX – The API

- As close as possible to C++17/20/23 standard library, where appropriate, for instance
  - std::thread, std::jthread
  - std::mutex
  - std::future
  - std::async
  - std::for_each(par, …), etc.
  - std::experimental::task_block
  - std::latch, std::barrier
  - std:: experimental::for_loop
  - std::bind
  - std::function
  - std::any
  - std::cout
  - hpx::thread (C++11), hpx::jthread (C++20)
  - hpx::mutex
  - hpx::future (including N4538, 'Concurrency TS')
  - hpx::async (including N3632)
  - hpx::for_each (N4507, C++17/20/23)
  - hpx::experimental::task_block (N4411)
  - hpx::latch, hpx::barrier (C++20)
  - hpx::experimental::for_loop
  - hpx::bind
  - hpx::function
  - hpx::any (C++20)
  - hpx::cout

The Future of Computation



STE||AR GROUP

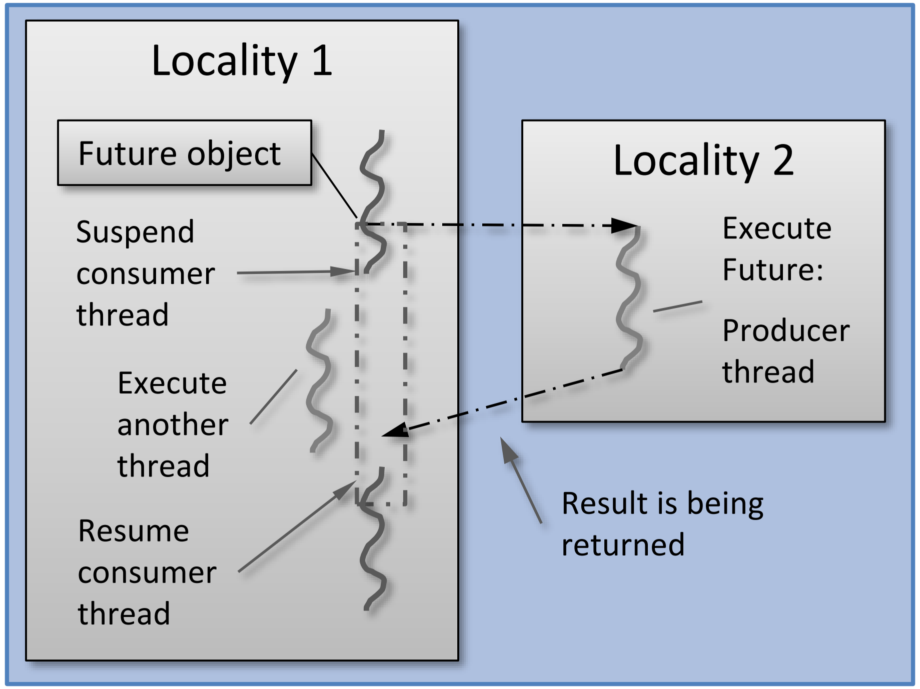
What is a (the) Future?

- Many ways to get hold of a (the) future, simplest way is to use (std) async:

```
int universal_answer() { return 42; }

void deep_thought()
{
    future<int> promised_answer = async(&universal_answer);

    // do other things for 7.5 million years

    cout << promised_answer.get() << endl;   // prints 42
}
```

What is a (the) future

- A future is an object representing a result which has not been calculated yet



- Enables transparent synchronization with producer
- Hides notion of dealing with threads
- Represents a data-dependency
- Makes asynchrony manageable
- Allows for composition of several asynchronous operations
- (Turns concurrency into parallelism)

Recursive Parallelism

STE||AR GROUP

Parallel Quicksort

```
template <typename RandomIter>
void quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > 1) {
        RandomIter pivot = partition(first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        quick_sort(first, pivot);
        quick_sort(pivot, last);
    }
}
```

STE||AR GROUP

Parallel Quicksort: Parallel

```
template <typename RandomIter>
void quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        RandomIter pivot = partition(par, first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        quick_sort(first, pivot);
        quick_sort(pivot, last);
    }
    else if (size > 1) {
        sort(seq, first, last);
    }
}
```

STE||AR GROUP

Parallel Quicksort: Futurized

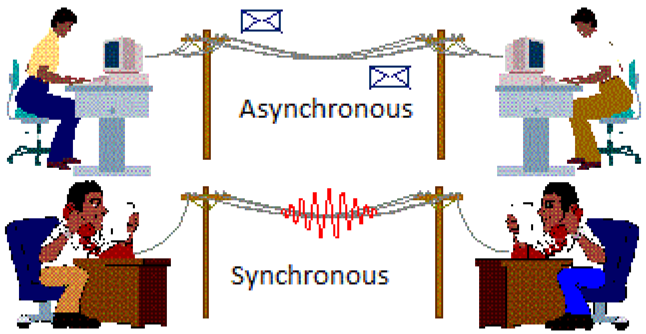
```
template <typename RandomIter>
future<void> quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        future<RandomIter> pivot = partition(par(task), first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        return pivot.then([=](auto pf) {
            auto pivot = pf.get();
            return when_all(quick_sort(first, pivot), quick_sort(pivot, last));
        });
    }
    else if (size > 1) {
        sort(seq, first, last);
    }
    return make_ready_future();
}
```

STE||AR GROUP

Parallel Quicksort: co_await

```
template <typename RandomIter>
future<void> quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        RandomIter pivot = co_await partition(par(task), first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        co_await when_all(
            quick_sort(first, pivot), quick_sort(pivot, last));
    }
    else if (size > 1) {
        sort(seq, first, last);
    }
}
```

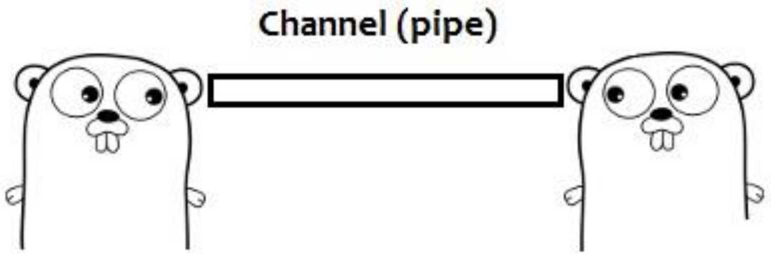
STE||AR GROUP



Asynchronous Communication

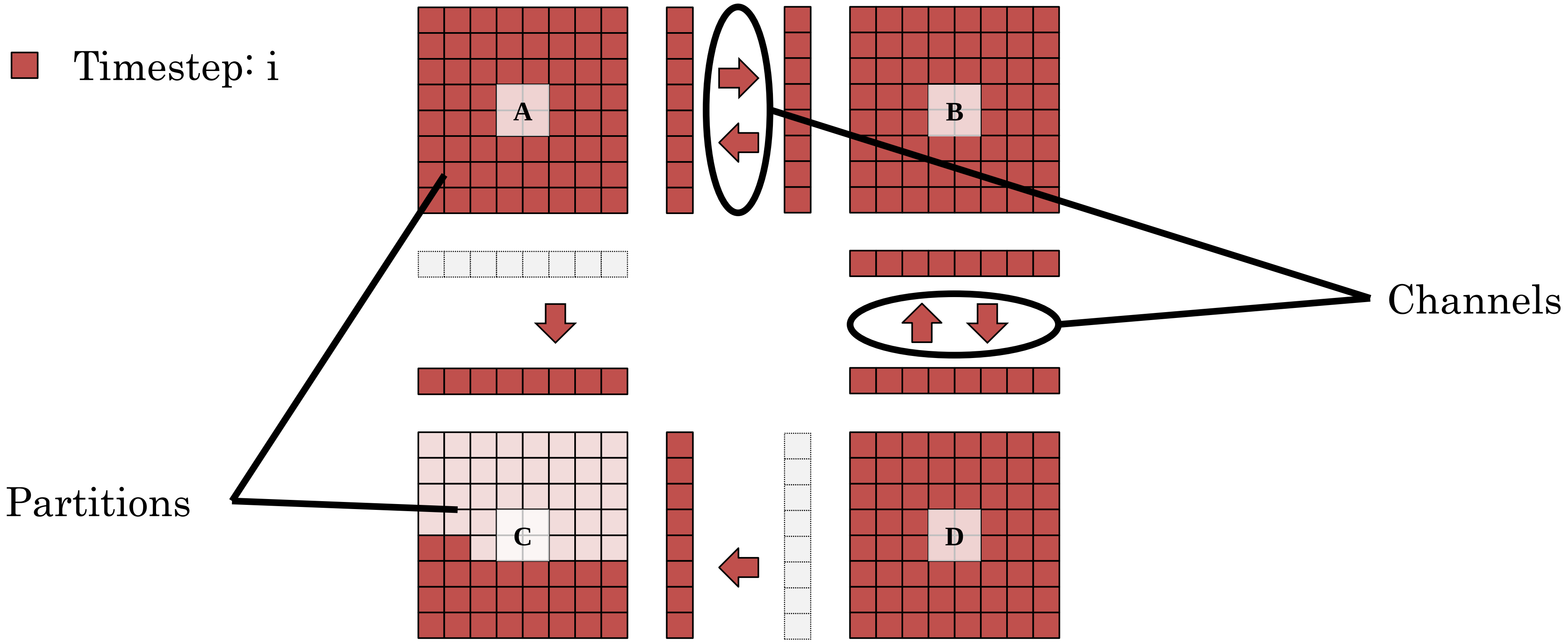
Example: Asynchronous Channels

- High level abstraction of communication operations
  - Perfect for asynchronous boundary exchange
  - Modelled after Go-channels
- Create on one thread, refer to it from another thread
  - Create on one locality, send over the wire, refer to it from another
  - Conceptually similar to bidirectional P2P (MPI) communicators
- Asynchronous in nature
  - `channel::get()` and `channel::set()` return futures



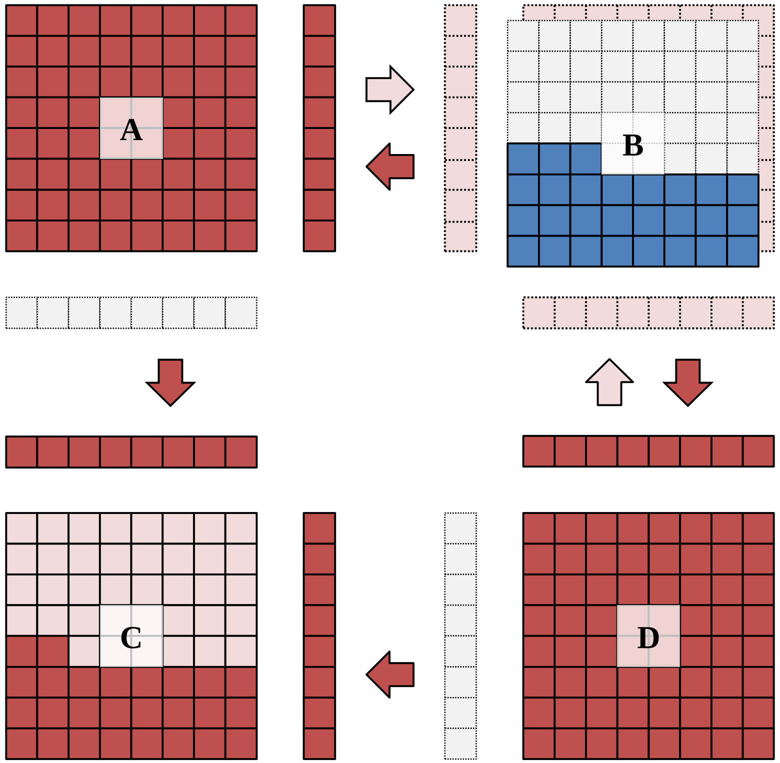
STE||AR GROUP

Futurized 2D Stencil: Timestep i



Futurized 2D Stencil: Timestep i+1

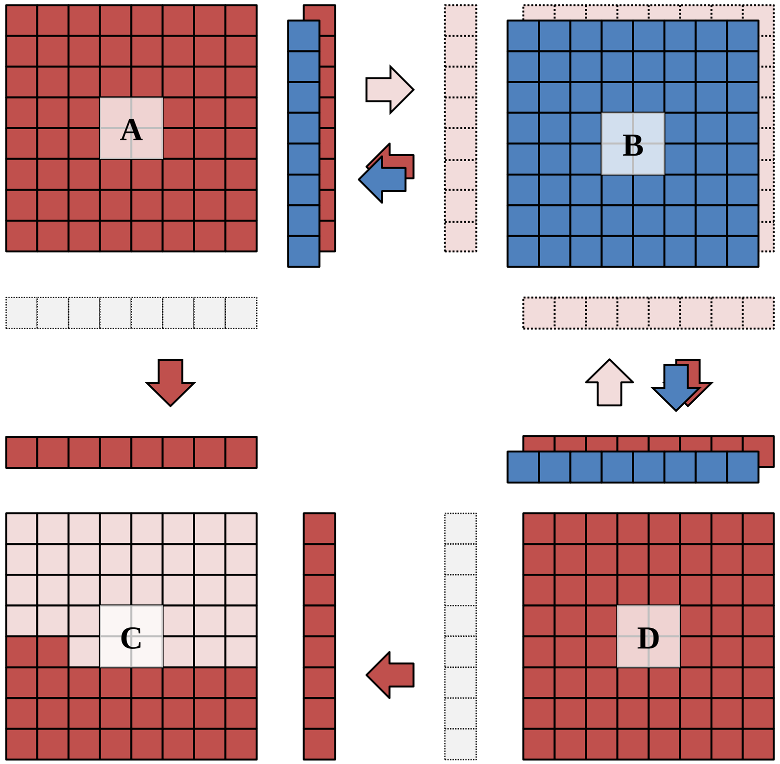
- Timestep: i
- Timestep: i+1



STE||AR GROUP

Futurized 2D Stencil

- Timestep: i
- Timestep: i+1



2D Stencil

- Partitions are distributed across machine
- More partitions per node (locality) than cores
  - Oversubscription
- Code equivalent regardless whether neighboring partition is on the same node
- Overlap of communication and computation
  - More parallelism (work) than compute resources (cores)

STE||AR GROUP

Futurized 2D Stencil: Main Loop

```
// execute this for each partition concurrently
hpx::future<void> simulate(std::size_t steps)
{
    for (size_t t = 0; t != steps; ++t)
    {
        co_await perform_one_time_step(t);
    }
    co_return;
}
```

STE||AR GROUP

One Timestep: Update Boundaries

```
future<void> upper_boundary(int t);  // same for other boundaries

future<void> perform_one_time_step(int t)
{
    // Update our boundaries from neighbors
    co_await when_all(upper_boundary(t), right_boundary(t),
        lower_boundary(t), left_boundary(t));

    // Apply stencil to partition
    co_await for_loop(par(task), min + 1, max - 1,
        [&](size_t idx) { /* apply stencil to each inner point */ });
}
```

STE||AR GROUP

One Timestep: Interior

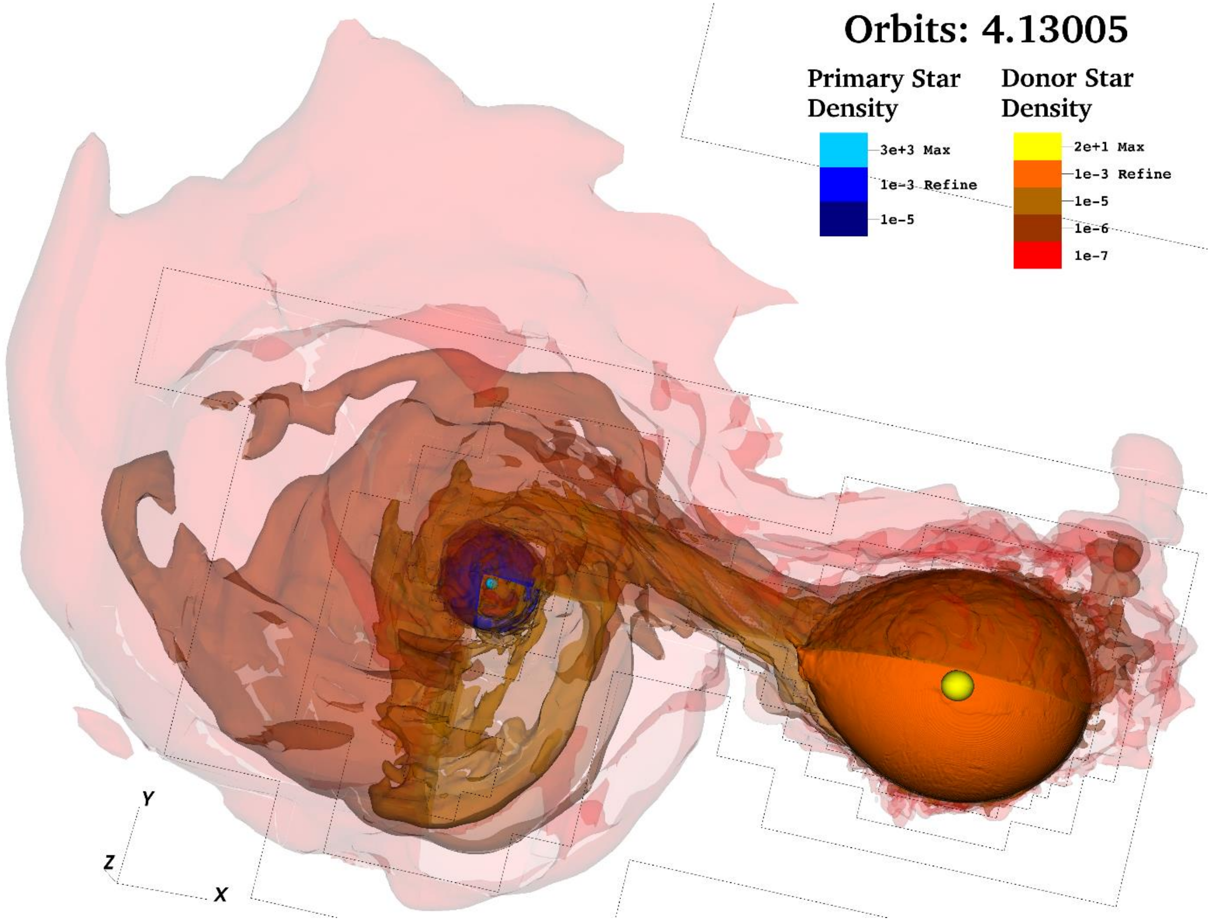
```
future<void> upper_boundary(int t)
{
    // Update upper boundary from upper neighbor
    vector<double> data = co_await channel_up_from.get(t);

    // process upper ghost-zone data using received data
    for_loop(seq, 1, size(data) - 1,
        [&](size_t idx) { /* apply stencil to each point in data */ });

    // send new ghost zone data to upper neighbor
    co_await channel_up_to.set(std::move(data), t + 1);
}
```

STE||AR GROUP

Recent Results

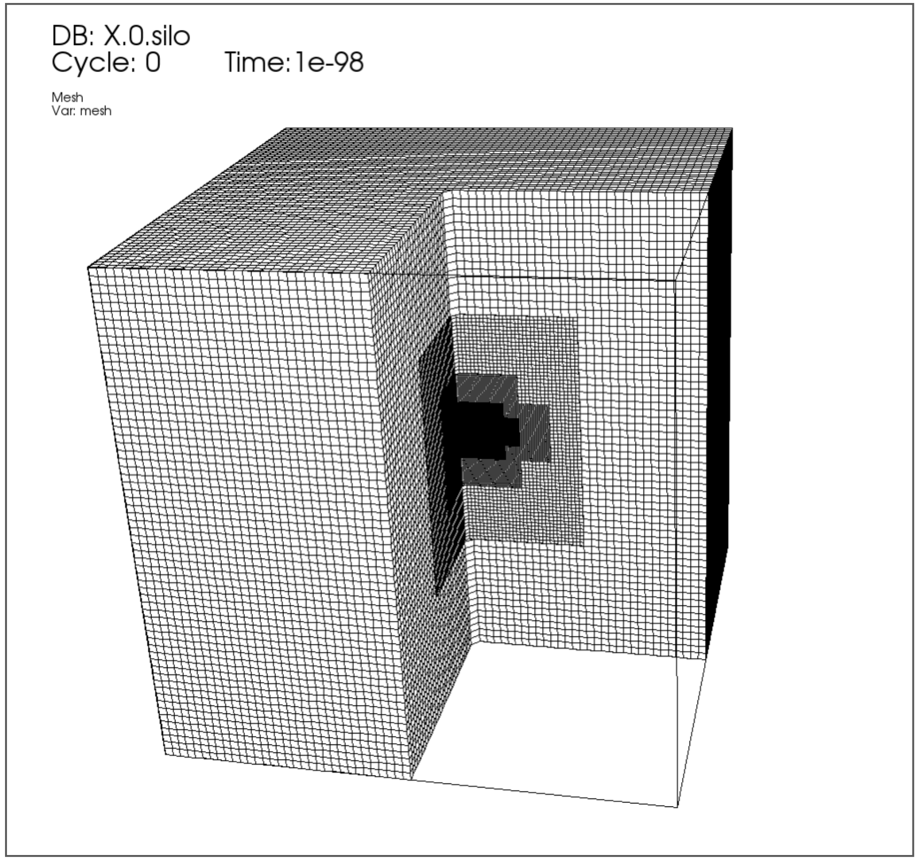
STE||AR GROUP

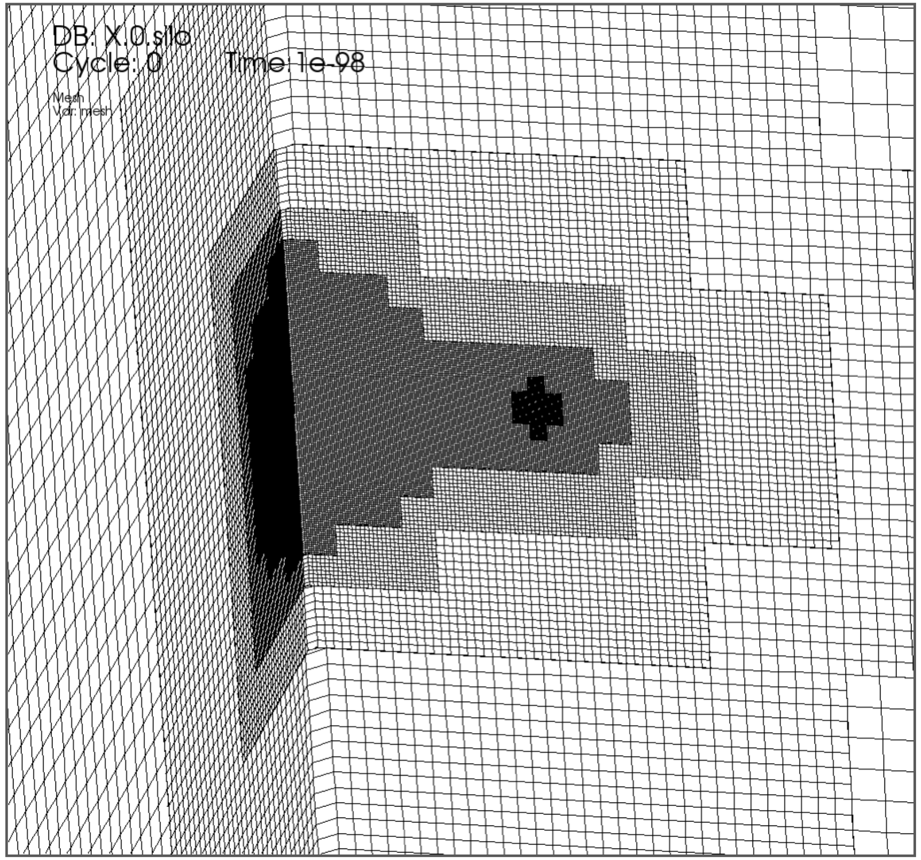
Merging White Dwarfs: OctoTiger



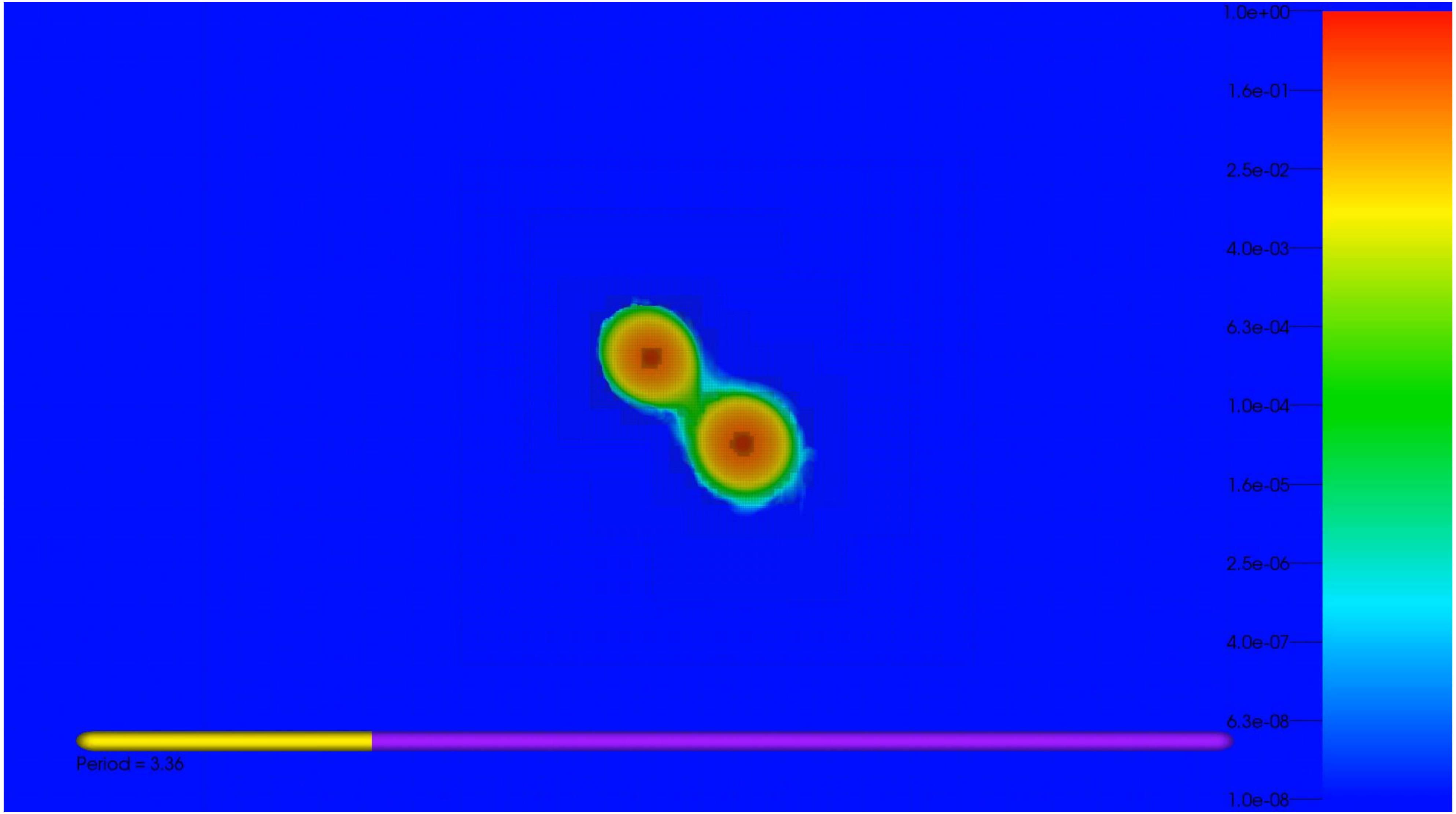
STE||AR GROUP

Adaptive Mesh Refinement



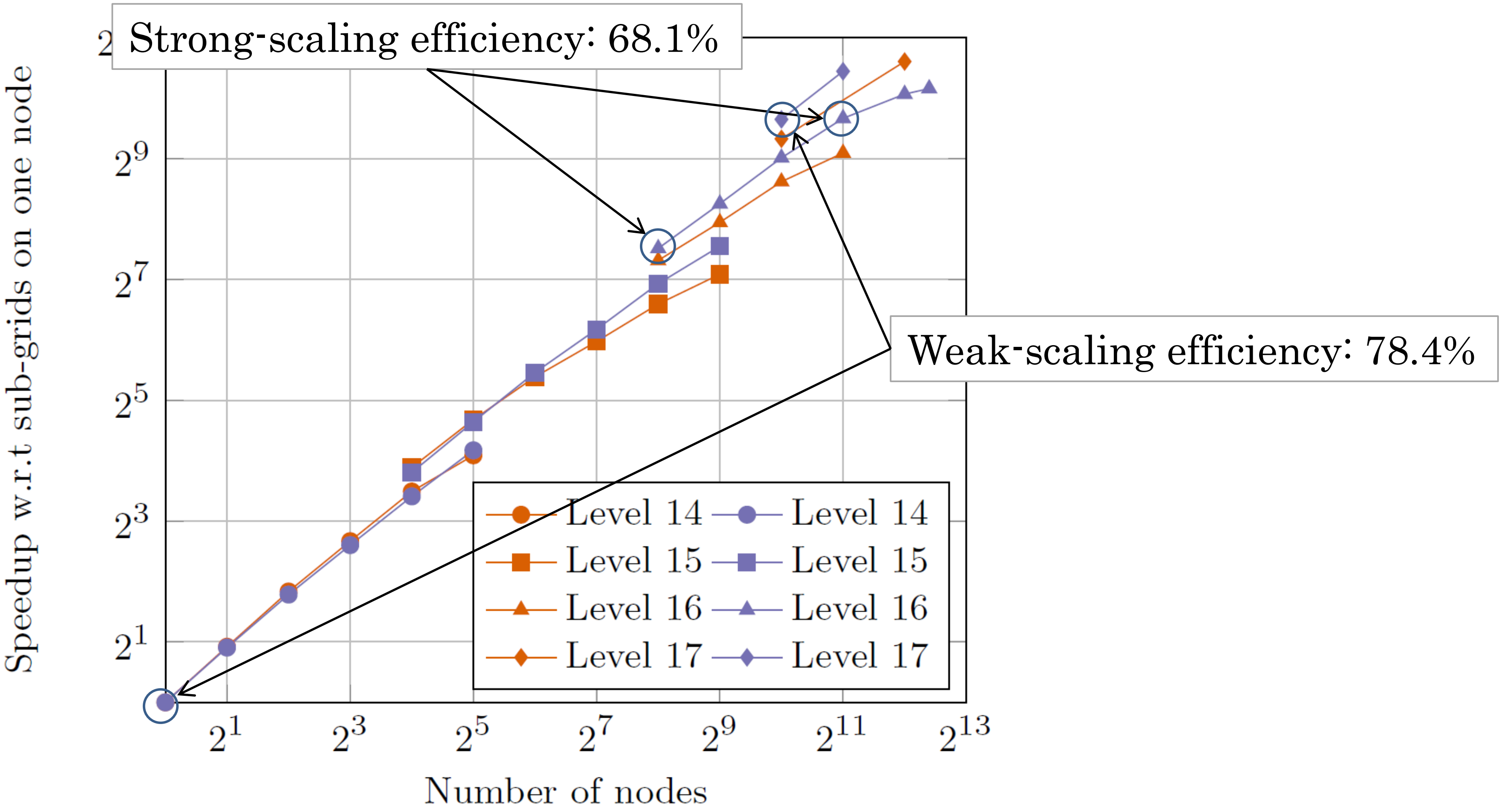


STE||AR GROUP


1.0e+00
1.6e-01
2.5e-02
4.0e-03
6.3e-04
1.0e-04
1.6e-05
2.5e-06
4.0e-07
6.3e-08
1.0e-08
Period = 3.36

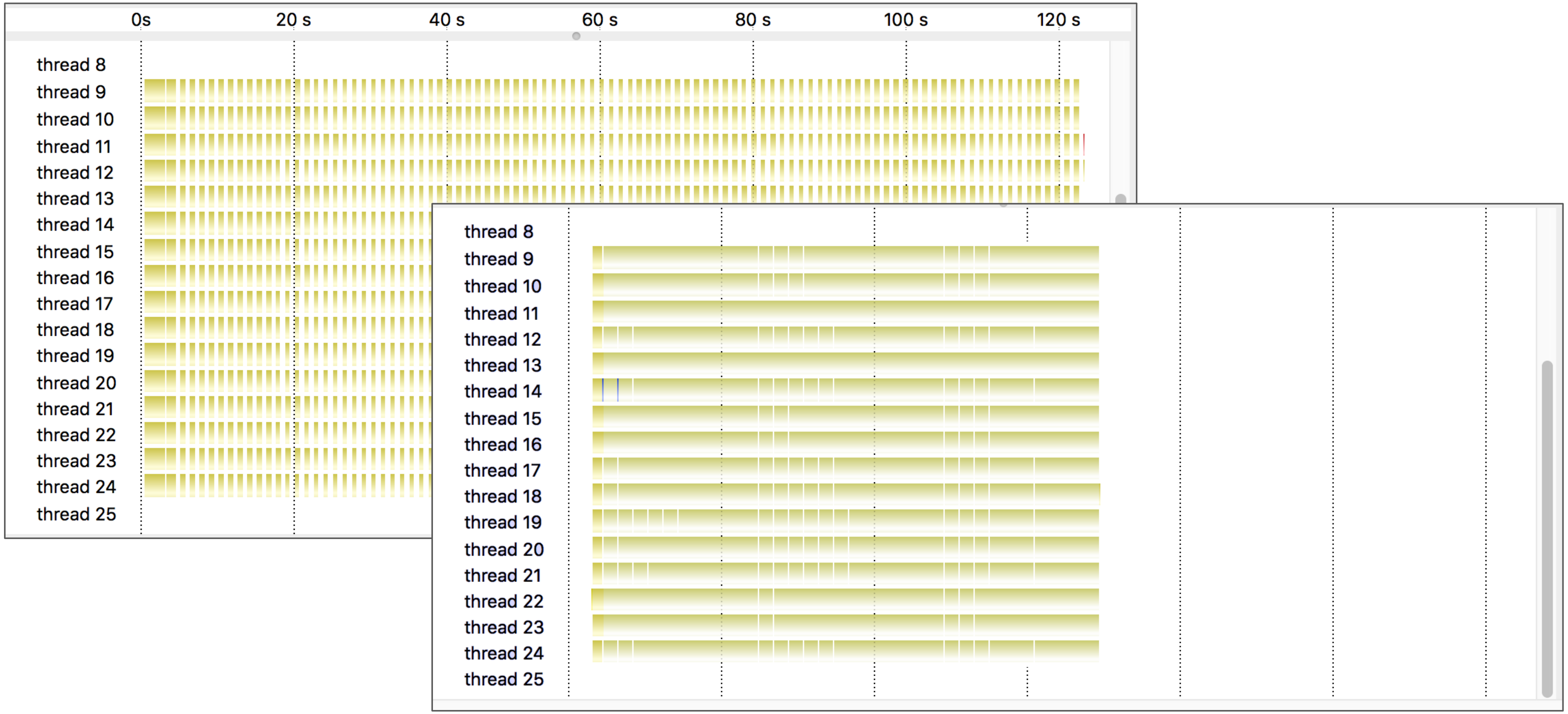
STE||AR GROUP

Adaptive Mesh Refinement



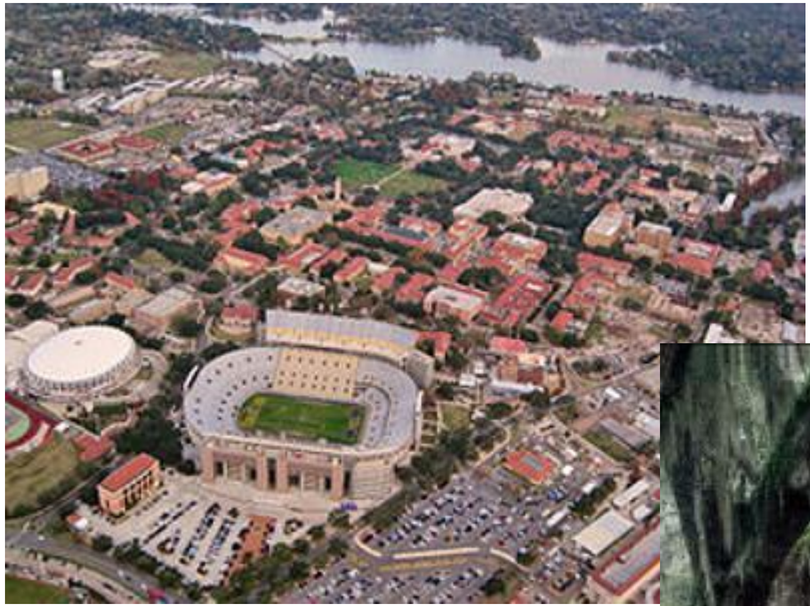
STE||AR GROUP

The Solution to the Application Problem

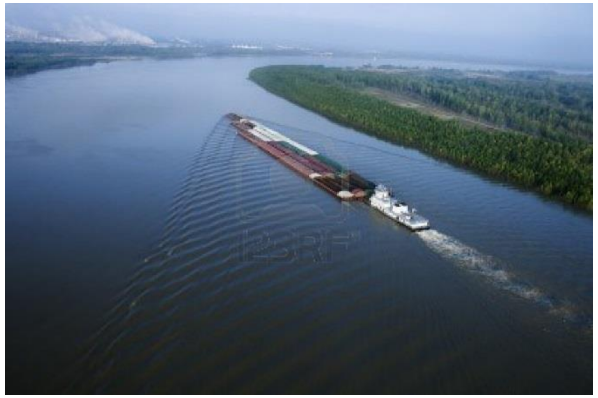


The Solution to the Application Problems

STE||AR GROUP



LSU CENTER FOR COMPUTATION & TECHNOLOGY



STE||AR GROUP