Concepts and Models of Parallel and Data-centric Programming

Distributed Shared Memory

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Outline

0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. **Distributed Shared Memory**
7. Parallel Algorithms
8. Parallel I/O
9. MapReduce
10. Apache Spark

a. PGAS Foundations
b. DASH Overview
c. Distributed Data Structured
d. DASH Algorithms
e. Tasking

DASH Algorithms

DASH Algorithms (1)

- There are a few DASH equivalents for STL algorithms, e.g., dash::**fill**, dash::**for_each**, etc.

- Example: Set all elements in the range to ‘val’

```
dash::GlobIter<T> dash::fill(GlobIter<T> begin,
                             GlobIter<T> end, T val);
```

- Implementation:
  - Perform a projection of global range to local range
  - Apply STL algorithm (e.g., std::fill) on local range
  - Combine results when needed (e.g., dash::min_element)

DASH Algorithms (2)

- Examples
  - dash::fill — arr[i] <- val
  - dash::generate — arr[i] <- func()
  - dash::for_each — func(arr[i])
  - dash::transform — arr2[i] = func(arr1[i])
  - dash::accumulate — sum(arr[i]) (0<=i<=n-1)
  - dash::min_element — min(arr[i]) (0<=i<=n-1)
  - dash::max_element — min(arr[i]) (0<=i<=n-1)

DASH Algorithms (3)

- Example: Find the min. element in a distributed array

```
dash::Array<int> arr(100, dash::BLOCKED);

for( auto i=0; i<arr.lsize(); i++ ) {
  arr.local[i]=rand()%100;
}
arr.barrier();

auto min = dash::min_element(arr.begin(),
                             arr.end());

if( dash::myid()==0 ) {
  cout<<"Minimum: "<<(int)*min<<endl;
}
```

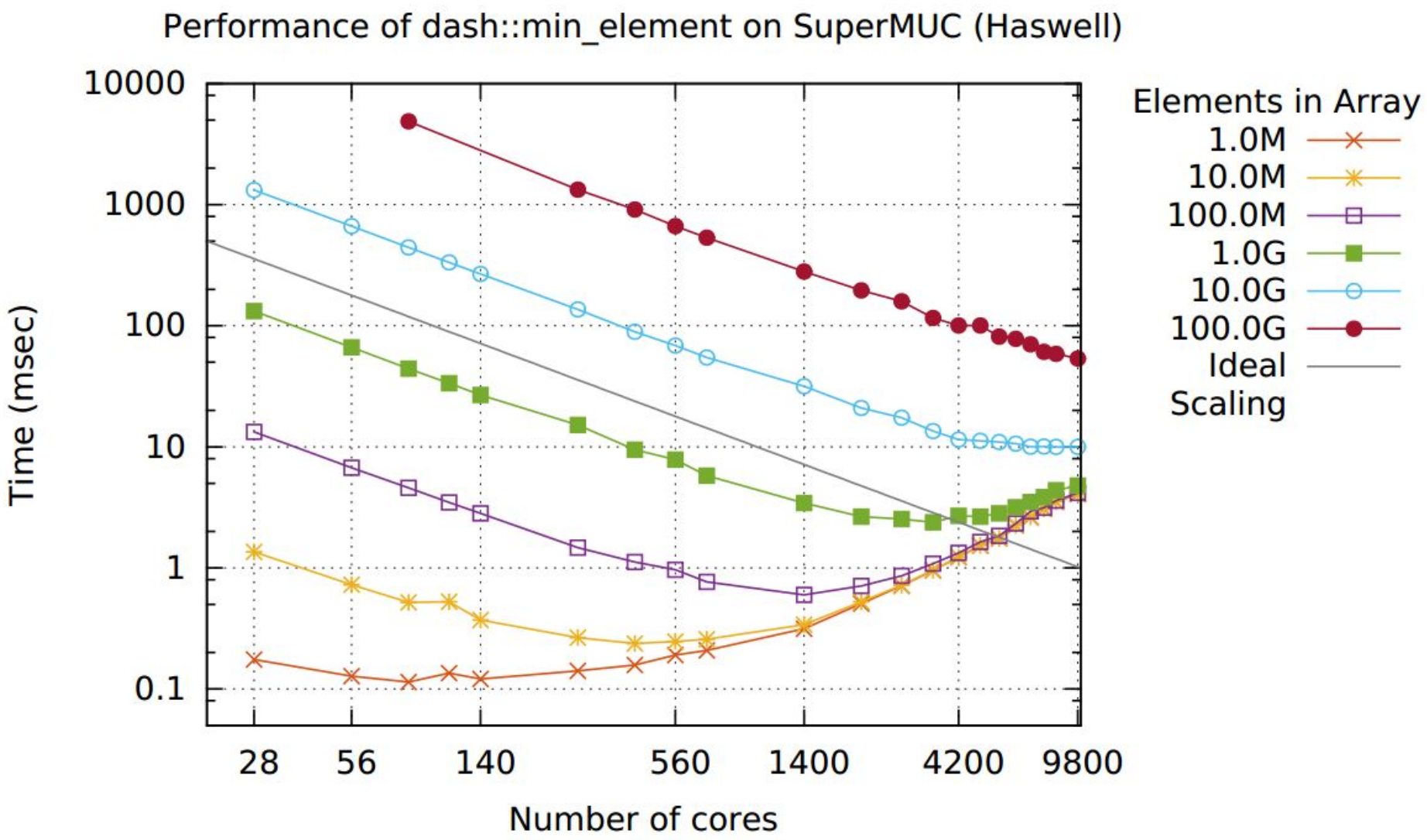
Collective call, returns global pointer To min. element

Identify local range and call std::min_element

Get the min. element value and print it

- Features
  - Still works when using CYCLIC or any other distribution
  - Still works when using a range other than [begin, end)

Performance of dash::min_element() (int)



Asynchronous Communication

- Realized via two mechanisms
  - Async. copy operation (dash::copy_async())
  - .async proxy object on DASH containers

```
// async. copy of global range to local memory
auto fut = dash::copy_async(block.begin(), block.end(),
                            local_ptr);
...
fut.wait();
```

Tasking

Tasking

- Task := a logically discrete section of computational work
  - Typically a set of instructions
  - May contain local / private data
  - A task-parallel program consists of multiple tasks running on multiple processors
- In Distributed Memory: a task is represented by a MPI process
- In Shared Memory:
  - Threading: implicit tasks
  - Tasking: Distribution of work into (explicit) tasks which are executed by threads
- In Shared Memory: a task is very similar to a future without a return value

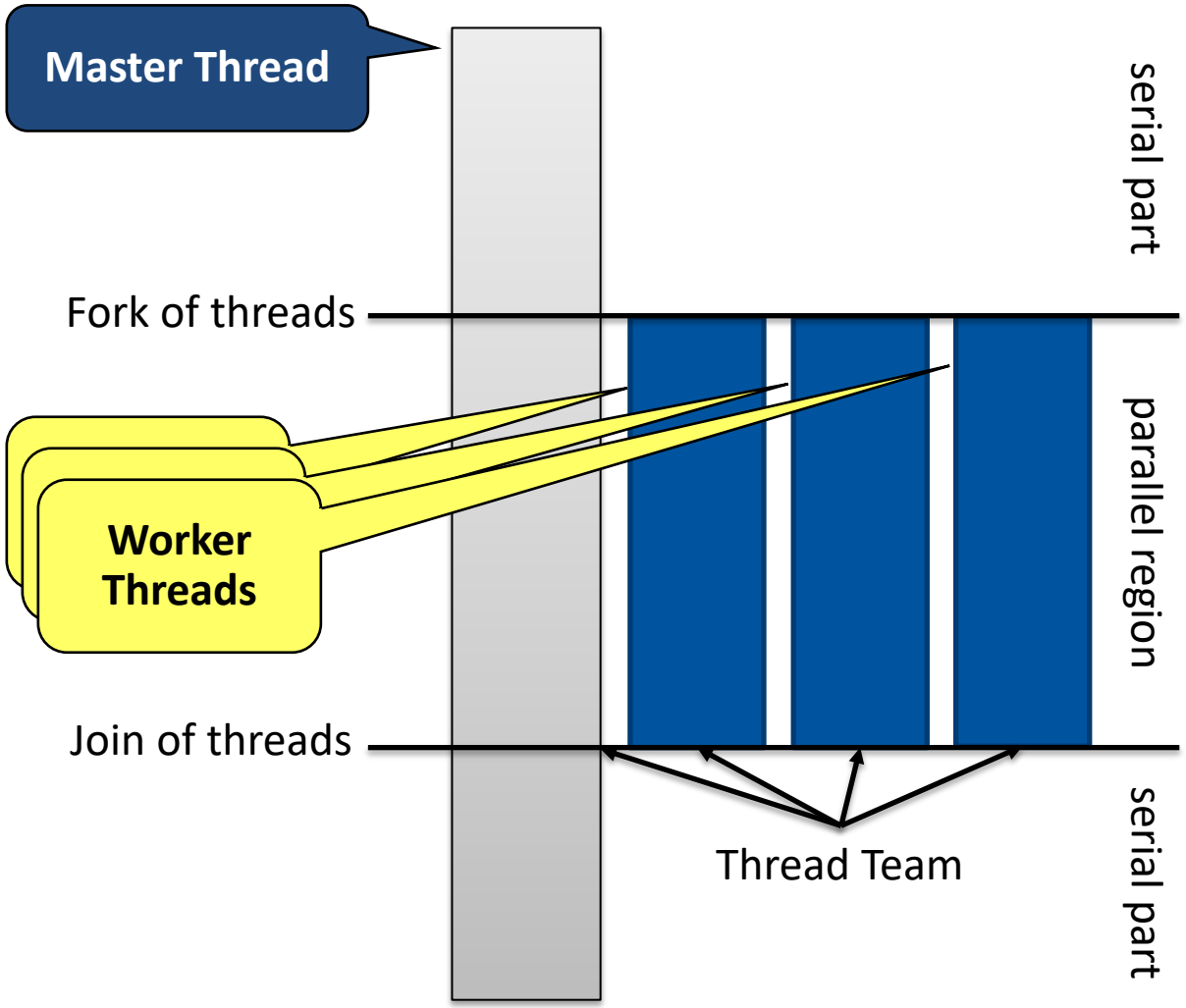
Task-loops / 1

- Distribution of loop iteration space over threads

```
dash::Matrix<double> matrix{N, N};

#pragma omp parallel for
for (auto iter = matrix.lbegin(); iter != matrix.lend(); ++iter)
{
  *iter *= 2;
}
```

OpenMP: fork-join execution model

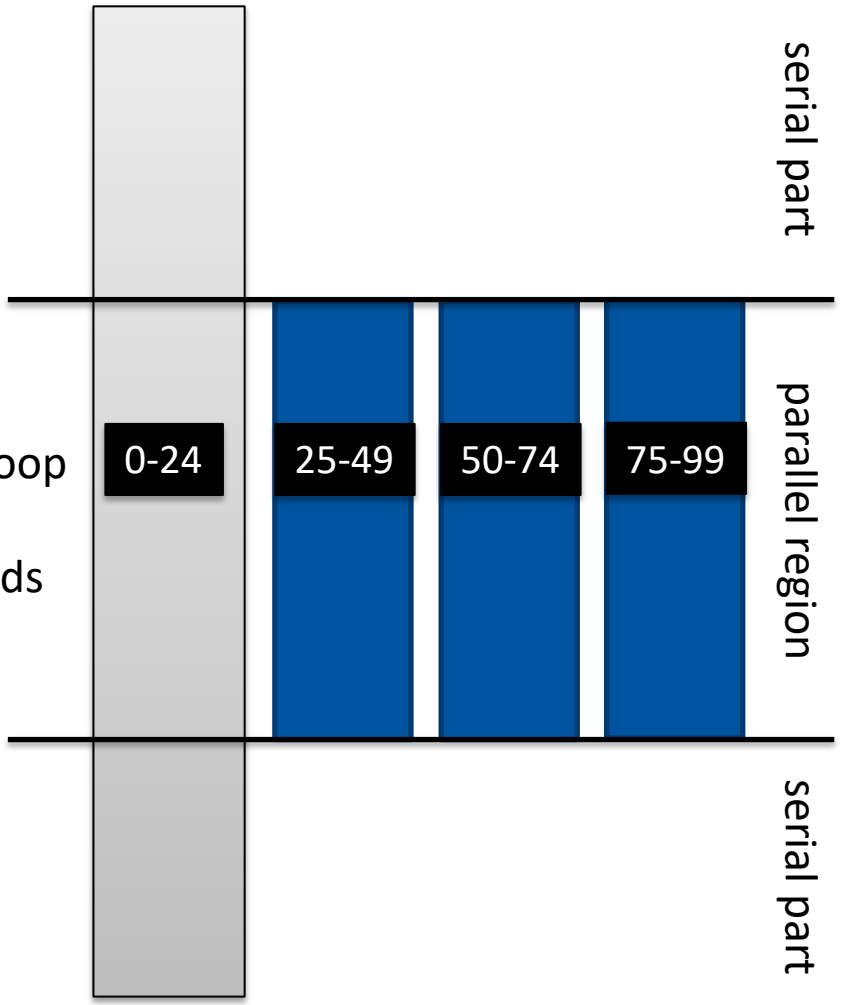
```
.
.
.
#pragma omp parallel
{
.
.
.
.
}
.
.
.
```



OpenMP: worksharing

```
.
.
.
#pragma omp parallel
#pragma omp for
for (int i=0; i<100; i++){
.
.
.
.
}
.
.
.
```

distributes loop iterations across threads



Task-loops / 2

- Distribution of loop iteration space over threads

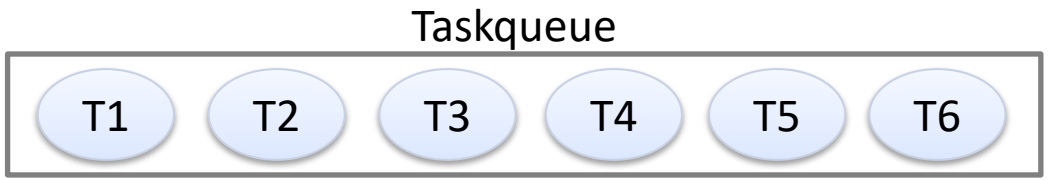
```
dash::Matrix<double> matrix{N, N};

#pragma omp parallel for
for (auto iter = matrix.lbegin(); iter != matrix.lend(); ++iter)
{
  *iter *= 2;
}
```

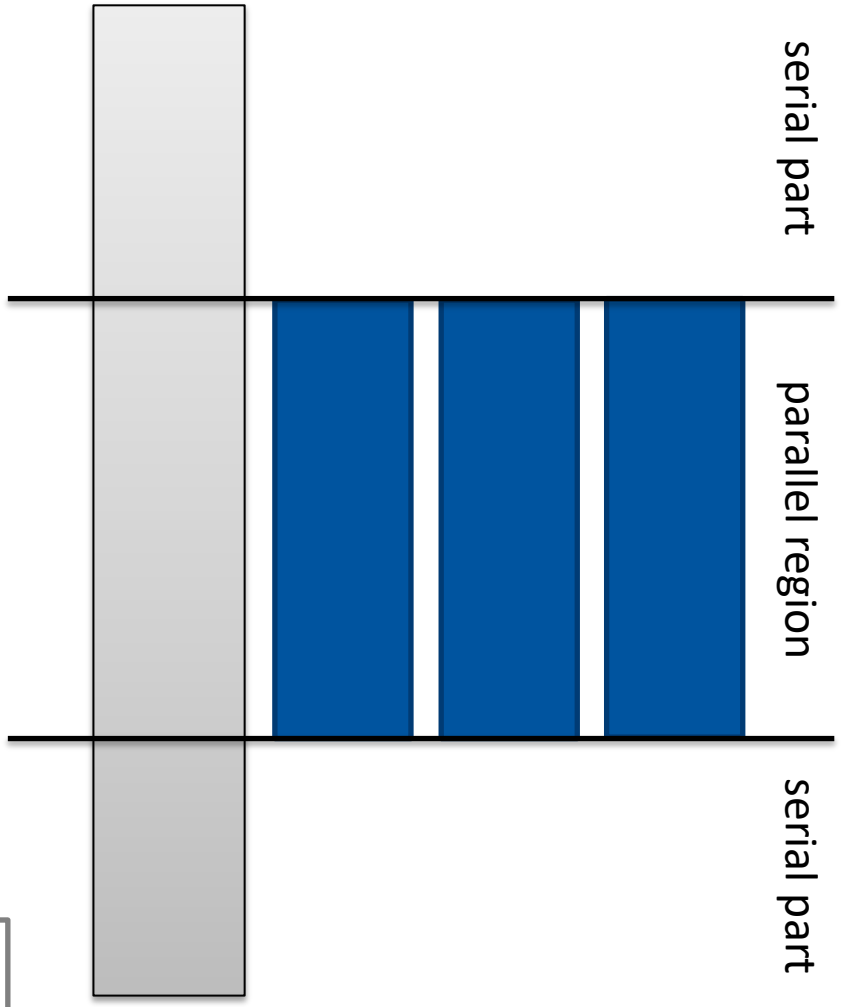
- Partitioning of loop iteration space into tasks

```
dash::taskloop(matrix.lbegin(), matrix.lend(),
  [&](auto begin, auto end){
  for (auto iter = begin; iter != end; ++iter)
  {
    *iter *= 2;
  }


});
```

OpenMP: Tasking

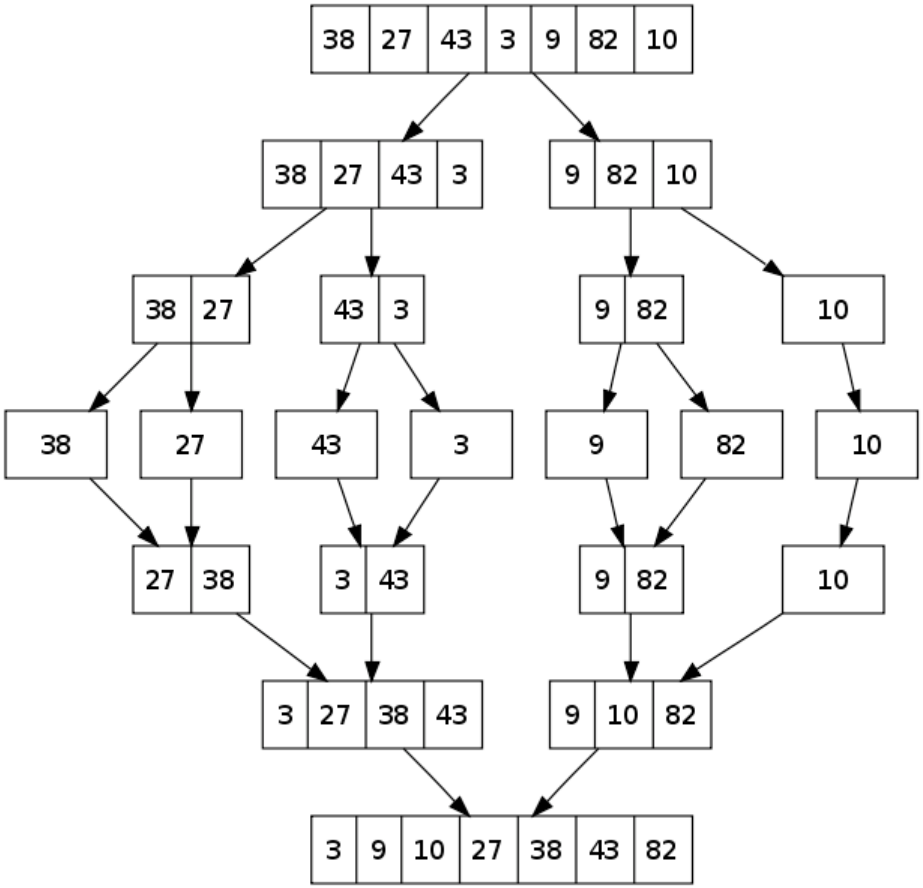
```
#pragma omp parallel
#pragma omp single
while (work()){
        #pragma omp task
        {
        …
        }
}// implicit barrier here
.
.
```





Tasking candidates

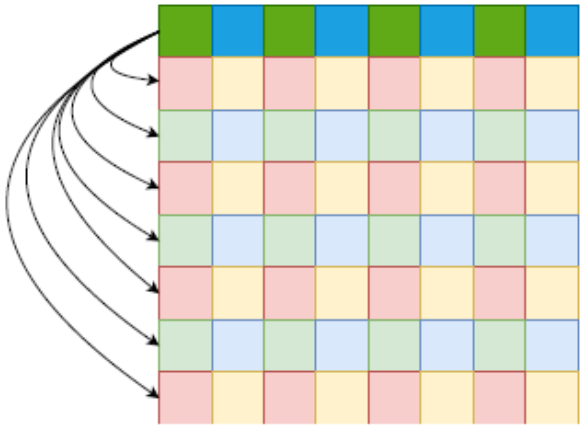
- Irregular / recursive algorithms, example: Merge-Sort

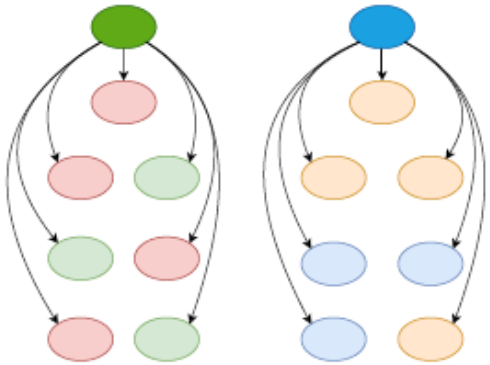


Challenge: Tasking in DSM

- DASH: Global Task Data Dependencies

```
dash::Matrix<double> matrix{N, N};


for (size_t j = 0; j < N; ++j) {
  if (matrix(0, j).is_local())

      matrix(0, j) = compute(j);

}

// wait for all blocks to be computed
dash::barrier();

for (size_t i = 1; i < N; ++i) {
  for (size_t j = 1; j < N; ++j) {
    if (matrix(i, j).is_local())

        apply(matrix(i, j),
              matrix(0, j));

  }
}
```

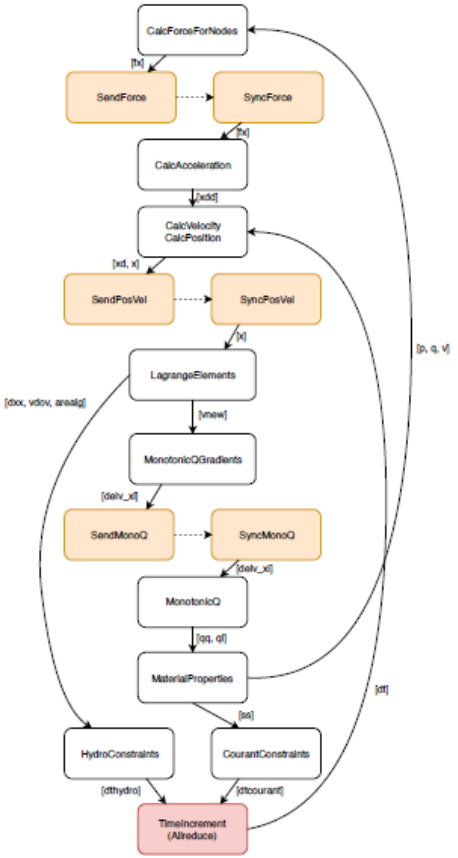




Potential of DSM / 1

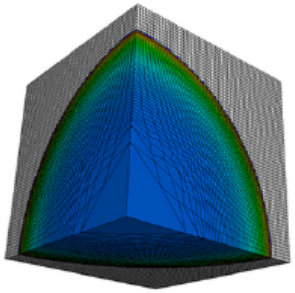
Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)

- 28pt stencil
- DoE CoDesign applications
- Abstraction levels: nodes, elements, regions



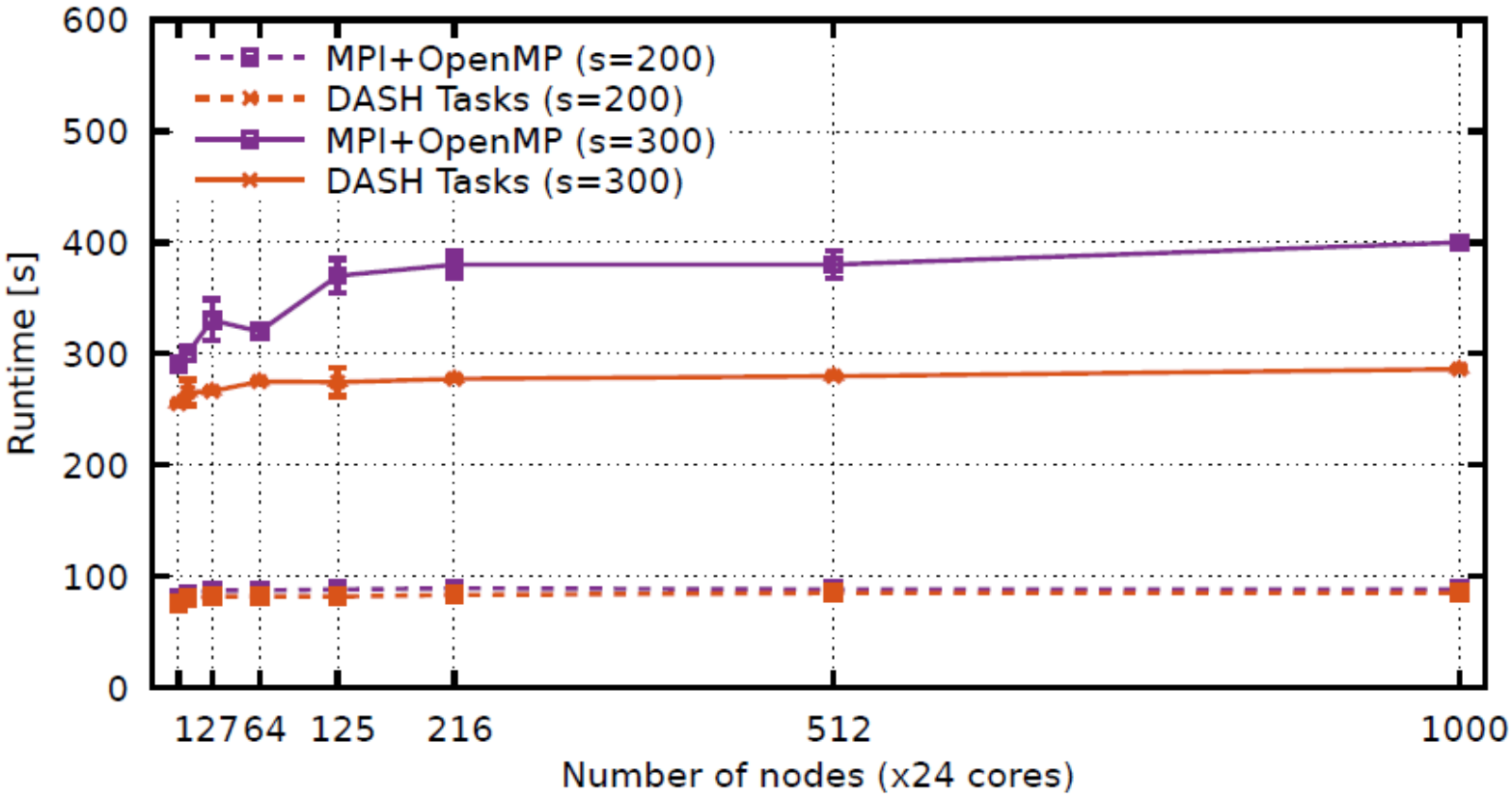
Porting Strategy

1. MPI Send/recv ⇒ `dash::copy`*
2. `omp parallel for` ⇒ `dash::taskloop`
3. Local dependencies
4. Remote dependencies



Potential of DSM / 2

- Results: LULESH @ Cray XC40 @ HLRS, Stuttgart, Germany



Note: weak scaling