FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Scalable AnyDSL Molecular Dynamics application with MPI

Scaling the Molecular Dynamics implementation in AnyDSL

Rafael Ravedutti Lucio Machado
Chair of Computer Science 10, Friedrich-Alexander University of Erlangen-Nuremberg

March 19, 2019

Motivation

Compare AnyDSL scalable application with other state-of-the-art technologies in order to explore benefits in code writing and performance.

Outline

AnyDSL

Molecular Dynamics

Proposal

Experimental Results

AnyDSL

AnyDSL

Framework for development of domain-specific libraries

- Higher-order functions
- Thorin
- Impala
- Partial evaluation

AnyDSL

```
fn main() {
  let img = load("dragon.png");
  let blurred = gaussian_blur(img);
}
```

AnyDSL

```
fn gaussian_blur(field: Field) -> Field {
  let stencil: Stencil = { /* ... */ };
  let mut out: Field   = { /* ... */ };

  for x, y in @iterate(out) {
    out.data(x, y) = apply_stencil(x, y, field, stencil);
  }

  out
}
```

AnyDSL

```
fn iterate(field: Field, body: fn(int, int) -> ()) -> () {
  let grid  = (field.cols, field.rows, 1);
  let block = (128, 1, 1);

  with nvvm(grid, block) {
      let x = nvvm_tid_x() + nvvm_ntid_x() * nvvm_ctaid_x();
      let y = nvvm_tid_y() + nvvm_ntid_y() * nvvm_ctaid_y();
      body(x, y);
  }
}
```

Molecular Dynamics

Molecular Dynamics

Pair-wise interaction of particles simulation implemented in AnyDSL

- Cells of particles (bounding boxes)
- Neighborlists
- Cluster of particles
- Target CPU with vectorization instructions and GPU

Molecular Dynamics

Steps

1. Initialize grid
2. Initialize clusters
3. Build neighbor lists
4. Compute forces and update particles (for 20 timesteps)
5. Redistribute particles and go back to item 2

Molecular Dynamics

Steps

1. Initialize grid
2. Initialize clusters
3. Build neighbor lists
4. Compute forces and update particles (for 20 timesteps)
5. Redistribute particles and go back to item 2

Molecular Dynamics

Steps

1. Initialize grid
2. Initialize clusters
3. Build neighbor lists
4. Compute forces and update particles (for 20 timesteps)
5. Redistribute particles and go back to item 2

Molecular Dynamics

Steps

1. Initialize grid
2. Initialize clusters
3. Build neighbor lists
4. Compute forces and update particles (for 20 timesteps)
5. Redistribute particles and go back to item 2

Molecular Dynamics

Steps

1. Initialize grid
2. Initialize clusters
3. Build neighbor lists
4. Compute forces and update particles (for 20 timesteps)
5. Redistribute particles and go back to item 2

Proposal

Proposal

Goals

- Scalable version of the application
- First: scale application on homogeneous clusters
- In the future: heterogeneous clusters (both CPU and GPU nodes)
- Compare scalable implementation with other state-of-the-art versions

Proposal

Steps

- Domain partitioning
- Communication pattern
- Synchronization of cells (every timestep)
- Particle exchange (after redistribution)

Proposal

Domain partitioning

- Define configuration of nodes
- Split domain accordingly
- Define current node bounding box
- Include ghost layer cells
- **get_sync_timesteps()**

Domain partitioning

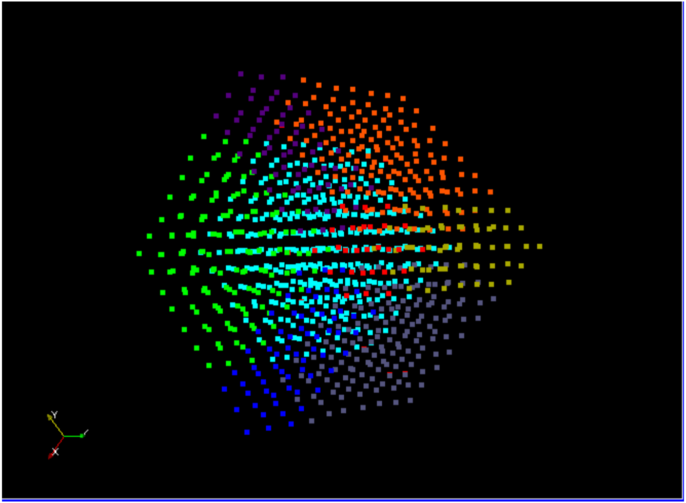
```
fn get_node_config(
  world_size: i32,
  rank: i32,
  xcells: i32,
  ycells: i32,
  zcells: i32) -> [i32 * 3] {

  let mut gx = 1, gy = 1, gz = 1;
  let mut min_missing_factor = xcells * ycells * zcells;

  for i in range(1, world_size) {
    if(world_size % i == 0) {
      let rem_yz = world_size / i;

      for j in range(1, rem_yz) {
        if(rem_yz % j == 0) {
          let k = rem_yz / j;
          let missing_factor = xcells % i + ycells % j + zcells % k;

          if(min_missing_factor > missing_factor) {
            gx = i;
            gy = j;
            gz = k;
            min_missing_factor = missing_factor;
          }
        }
      }
    }
  }

  [gx, gy, gz]
}
```

Domain partitioning

```
fn @get_node_bounding_box(
  world_size: i32,
  rank: i32,
  cell_spacing: f64,
  aabb: AABB) -> AABB {

  let mut xmin, xmax, ... : f64;

  let xcells = math.floor((aabb.max(0) - aabb.min(0)) / cell_spacing);
  let xlength = (xcells / node_dims(0)) * cell_spacing;
  let rank_index = unflat_index(rank, ...);

  xmin = aabb.min(0) + xlength * rank_index(0);
  xmax = aabb.min(0) + xlength * (rank_index(0) + 1);

  if(rank_index(0) > 0) {
    xmin -= get_sync_timesteps() * cell_spacing;
  }

  if(rank_index(0) < node_dims(0) - 1) {
    xmax += get_sync_timesteps() * cell_spacing;
  }

  /* y and z are analogous to x */

  AABB {
    min: [xmin, ymin, zmin],
    max: [xmax, ymax, zmax]
  }
}
```

Proposal

Domain partitioning



Proposal

Communication pattern

- Higher-order function for iteration
- Easy to write and change with AnyDSL

Communication pattern

```
fn communication_nodes(
  world_size: i32,
  rank: i32,
  grid: Grid,
  body: fn(i32, [i32 * 3], [i32 * 3], [i32 * 3], [i32 * 3]) -> ()) -> () {

  ...

  if(rank_index(0) < node_dims(0)) {
    @@body(
      flat_index([rank_index(0) + 1, rank_index(1), rank_index(2)], ...),
      send_begin1, send_end1, recv_begin1, recv_end1 // regions to communicate
    );
  }

  if(rank_index(0) > 0) {
    @@body(
      flat_index([rank_index(0) - 1, rank_index(1), rank_index(2)], ...),
      send_begin2, send_end2, recv_begin2, recv_end2 // regions to communicate
    );
  }

  ... /* y and z are analogous to x */
}
```

Proposal

Synchronization of cells

- Update positions, velocity and forces of particles
- One-step communication
- Every **get_sync_timesteps()** timesteps

Synchronization of cells

```
fn synchronize_ghost_layer_cells(
  grid: &mut Grid,
  accelerator_grid: AcceleratorGrid,
  world_size: i32,
  world_rank: i32) -> () {

  ... /* Transfer data from accelerator to CPU */

  for exchange_rank, send_begin, send_end, recv_begin, recv_end in
      communication_nodes(world_size, world_rank, *grid) {

    pack_ghost_layer_cells(..., send_begin, send_end);

    mpih.irecv(...);
    mpih.send(...);
    mpih.wait(...);

    unpack_ghost_layer_cells(..., recv_begin, recv_end);
  }

  ... /* Transfer data from CPU to accelerator */
}
```

Proposal

Particle exchange

- Exchange redistributed cells
- May be a two-step communication ($N' > N + N/2$)
- Every 20 timesteps (redistribution)

Particle exchange

```
fn exchange_ghost_layer_particles(
  grid: &mut Grid,
  world_size: i32,
  world_rank: i32) -> () {

  ...

  for exchange_rank, send_begin, send_end, recv_begin, recv_end in
      communication_nodes(world_size, world_rank, *grid) {
    ...

    pack_ghost_layer_particles(..., &mut rmng_send_ptcs);
    mpih.irecv(...);
    mpih.send(...);
    mpih.wait(...);
    unpack_ghost_layer_particles(..., &mut rmng_recv_ptcs);

    if(rmng_recv_ptcs > 0) {
      mpih.irecv(...);
    }

    if(rmng_send_ptcs > 0) {
      pack_ghost_layer_particles(...);
      mpih.send(...);
    }

    if(rmng_recv_ptcs > 0) {
      mpih.wait(...);
      unpack_ghost_layer_particles(...);
    }
  }
}
```

Experimental Results

Experimental Results

Cluster configuration

- 36-CPU High-Performance-Cluster
- 8 compute nodes
- 4 x Intel(R) Xeon(R) CPU E7-4830, 2.13 GHz 2.4GHz (max. turbo) (8 cores + SMT), SSE 4.1/4.2, 24 MB shared cache
- 256 GB RAM
- 2 x 300 GB SAS internal disks
- NVIDIA GeForce GTX 680
- QDR Infiniband network

Experimental Results

Cluster mapping

- MPI processes are mapped to different CPU cores in the same cluster node
- We bind processes to sockets for better memory usage

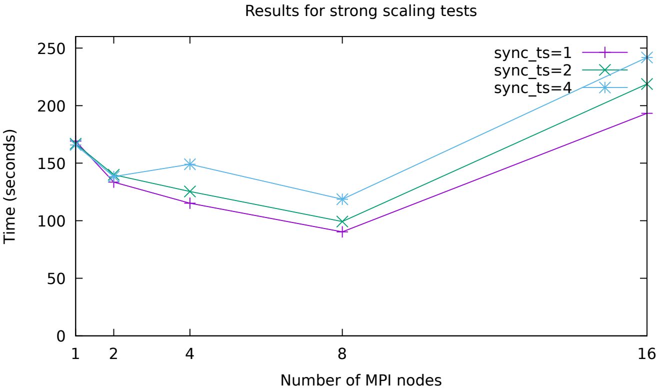
Experimental Results

Strong scaling

- Grid size: 128x128x128
- About 2 million particles

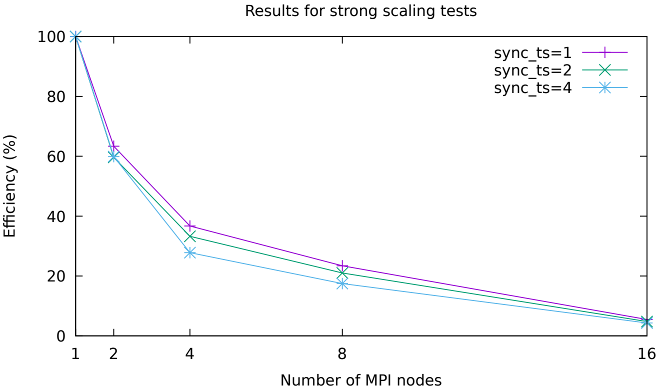
Proposal

Strong scaling time



Proposal

Strong scaling efficiency



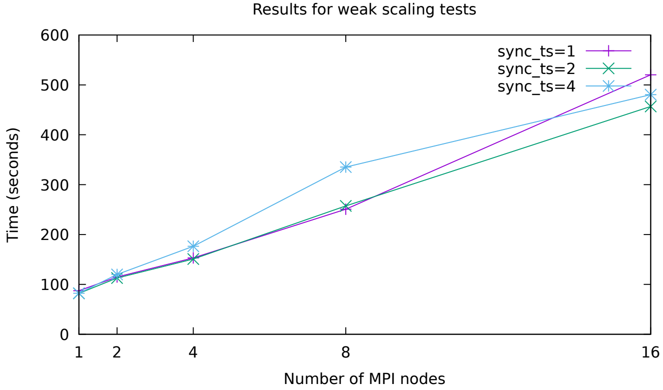
Experimental Results

Weak scaling

- **Problem: cannot use enough particles (too much memory)**
- Must fix this in order to get good weak scaling results

Experimental Results

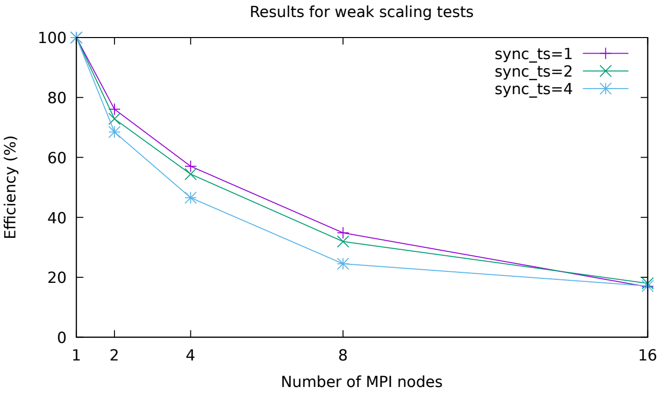
Weak scaling time



-

Experimental Results

Weak scaling efficiency



-

Future work

- Reduce memory usage by application
- Improve partitioning for heterogeneous clusters
- Compare with state-of-the-art applications

Thanks for listening.

Any questions?

References

References I

[1] J. Schmitt, H. Kostler, J. Eitzinger, and R. Membarth, “Unified code generation for the parallel computation of pairwise interactions using partial evaluation”, *17th International Symposium on Parallel and Distributed Computing*, 2018.