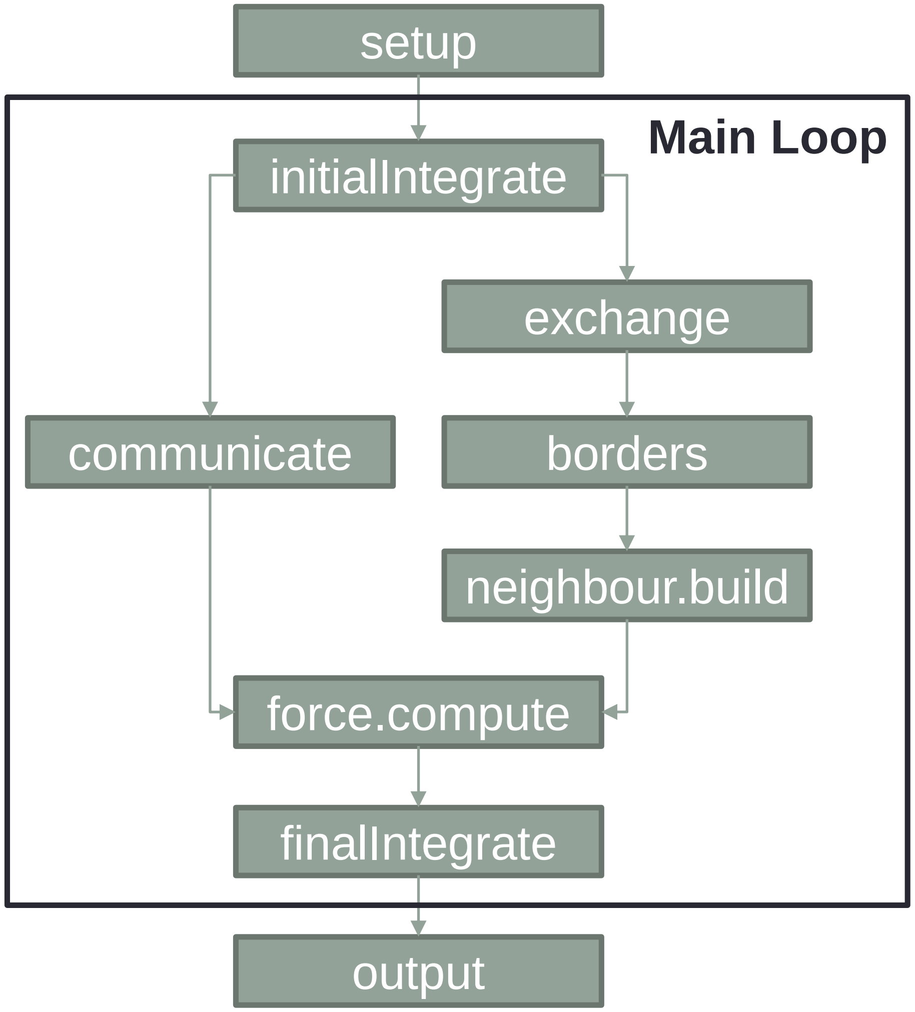
MANTEVO: MINIMD

Preparing for tasks and TAMPI

Overview

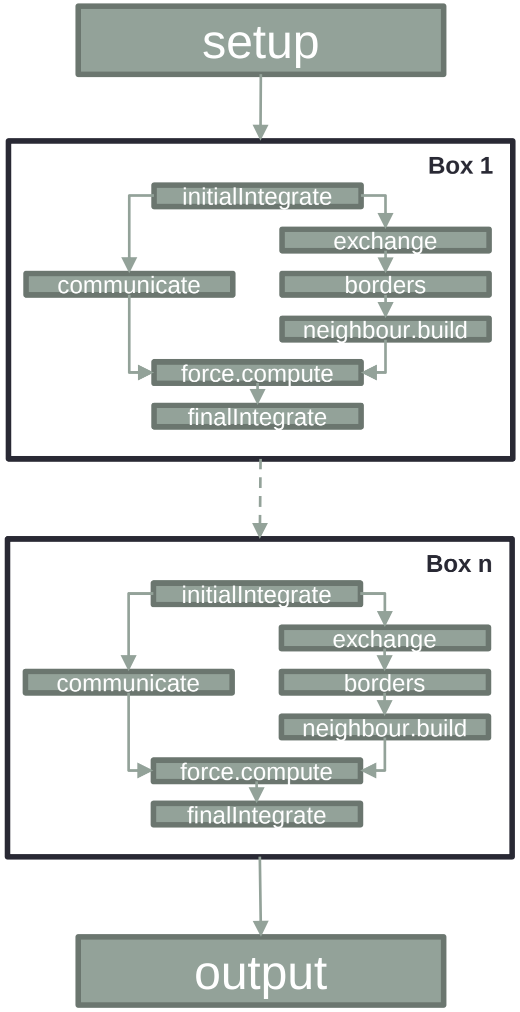
- Molecular dynamics code designed to mimic LAMMPS
- MPI+OpenMP
- ~5000 lines of C++
  - Simple classes, no advanced C++ features (complex templating, etc.)
  - Builds against TAMPI
- Uses neighbour lists for force calculation. Contrasts with cell lists employed by CoMD

Code Summary



- setup creates data structures, determines process grid, calculates slab boundaries and sets PBCs
- Most timesteps: only update atom positions via communicate function
- Every n timesteps (controlled by input file): perform more expensive re-neighbouring operation and exchange ownership of atoms
- Communication done in X, Y, then Z dimensions. No explicit corner messages

Preparing for Tasks: Multiple Boxes Per Process

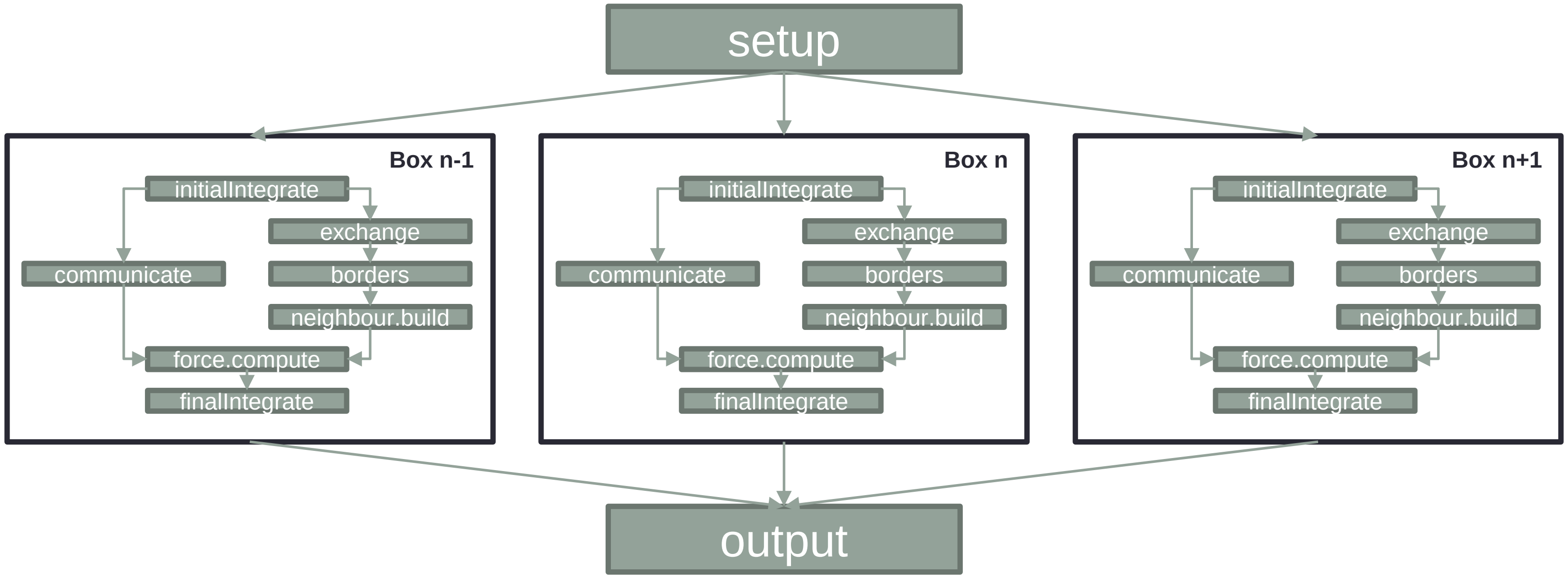


- Split single process box into multiple boxes per process
- Use single "column" of boxes:
    - 3 messages per off-process neighbour, one per box layer: same (box_id), lower (box_id-1) and upper (box_id+1)
    - 2 messages between internal boxes on same process (optimised to a memory copy).
- 26 messages sent in total: 3 off-process * 8 neighbours + 2 internal

Implementation Overview

- Number of boxes per process controlled via new input file parameter
- Each iteration of main loop broken into smaller loops over each box
- Send and receive buffers duplicated per communication:
  - 8 buffers per neighbour * 3 layers * 3 functions that perform MPI communication * 2 separate send/recv + 4 internal = 148 total buffers per box
- PBCs performed relative to periphery boxes rather than processes
- Communication now explicit with corners:
  - Atoms now marked for sending "up front" rather than after each swap in a dimension
  - Increased number of swaps per round of communication: swaps now in X, Y, Z, "left corner" and "right corner" dimensions and can be performed in any order
- Tags and duplicate communicators used to ensure messages are delivered to correct box on a process

Introducing Tasks



Neighbouring boxes run in parallel with task dependencies between relevant operations