Exercise 7

7.1 Reading

7.1.1 LogP: Towards a Realistic Model of Parallel Computation

The paper addresses the lack of a general model for parallel machine models. According to the authors the models used are either to Simplistic, the authors use the PRAM model as an example, or overly specific so as to only apply to few select actual hardware devices.

As an alternative LogP model is introduced which is supposed to act as general model for parallel machines. It uses four key parameters to characterize a parallel machine: the number of processors, communication bandwidth, communication latency, and communication overhead. Since parallel machines employ vastly different strategies for problems LogP is high-level enough to incorporate factors of e.g. the interconnection network without using implementation details like topology or routing. Additionally to describing the LogP model the paper also addresses how to use the model to develop efficient parallel algorithms, primarily at the example of FFT and LU decomposition.

As also mentioned in the paper the LogP model also serves as a more concrete version of the BSP model proposed by Valiant which neglects for example synchronization. This leads to algorithms being designed to optimal up to constant factors. As these may be large LogP is designed to also take these into account. These seem like reasonable observations and we therefore accept the paper.

7.1.2 Roofline: An Insightful Visual Performance Model for Multicore Architectures

To address the increasing variety in microprocessors, the Roofline model was introduced to offer cross-platform understandable performance guidelines using bound and bottleneck analysis. It ties together floating-point performance, operational intensity and memory performance in a 2D-graph. After explaining the function of the Roofline model, the paper demonstrates its utility on four diverse multicore computers, then optimizing four floating-point kernels taken from the Seven Dwarfs before clearing up some fallacies.

The Roofline model sets an upper bound on performance depending on the kernel's operational intensity, thus showing if performance of the kernel is compute-bound or memory-bound and, by adding ceilings, which optimizations may be necessary.
Finally, the ridge point is a better predictor for performance than clock rate or peak performance and indicates when a computer is imbalanced.

The Roofline model is still used today to check for a kernel's peak performance, although one has to especially consider to choose the right metric. Therefore, we accept the given paper.

7.2 n-Body Problem — Partitioning/Communication Design

7.2.1 Memory Layout

```
std::random_device rd;  //Will be used to obtain a seed for the random number engine
std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
std::uniform_real_distribution<> mass_distrib(1e-10, 1e10);
```

```
std::uniform_real_distribution<> space_distrib(-1e4, 1e4); // start within 25km of eachother

struct Body {
  union {
    double raw[4];
    struct __attribute__((__packed__)) {
      double m;
      double pos[3];
    };
  };

  Body() {
    m = mass_distrib(gen);
    for (int i=0; i < 3; i++) {
      pos[i] = space_distrib(gen);
    }
  }
}

// to send we just use the raw data
MPI_Send(&b, 8, ...)
```

- this structure will result in contiguous arrays of bytes which can be sent by their `double raw[4]` representation
- velocities will not be sent
- `MPI_Gather()` will then collect blocks of mulitple bodies per rank

7.2.2 Partitioning

- the amount of bodies should be a multiple of the available ranks, resulting in equally large messages
- each rank will handle a number of bodies
- the initial positions and masses are scattered over the ranks
- ranks are sequential over nodes (per-slot-mapping) resulting in the least amount of node-to node communication due to circular messaging ($\frac{4}{16}$ of all communication for 16 ranks over 4 nodes)

7.2.3 Communication

- the communication will happen in a circular fashion
- for each arriving packet a copy will be saved inside of a buffer
- the now buffered msg can be forwarded to the next rank
- while waiting for the arrival of the next msg, the received data can be used to update the Forces on each body the specific rank is responsible for

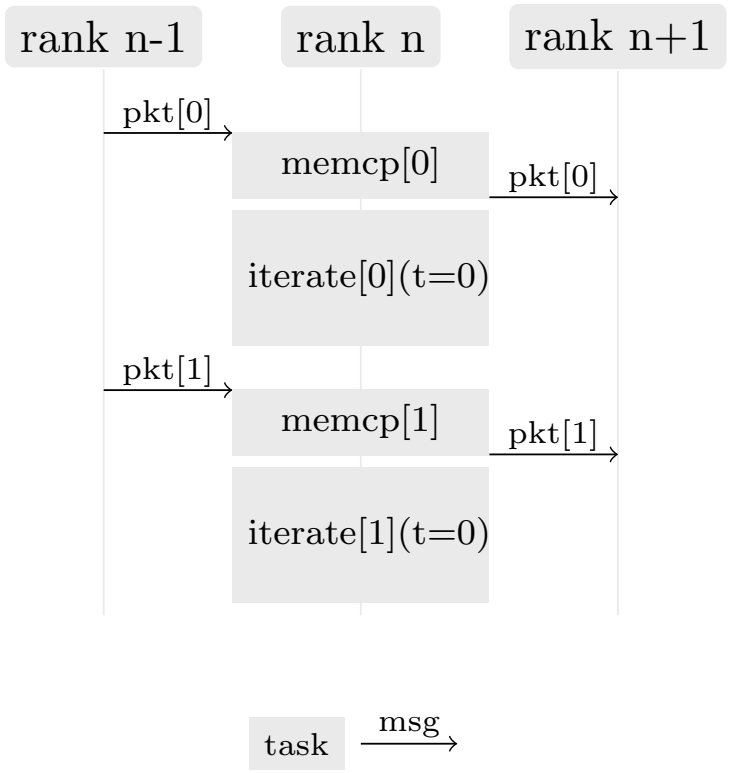

Figure 1: Overalp utilization during ring-wise message passing

Figure 1 shows this sequence for a rank n:

- here rank n starts out with a running `MPI_Irecv()` and waits for `pkt[0]`
- after copying the received data to a buffer, rank n calls `MPI_Isend()` on `pkt[0]` and starts another `MPI_Irecv()`
- after iterating over the received data, rank n will wait again for `pkt[1]`