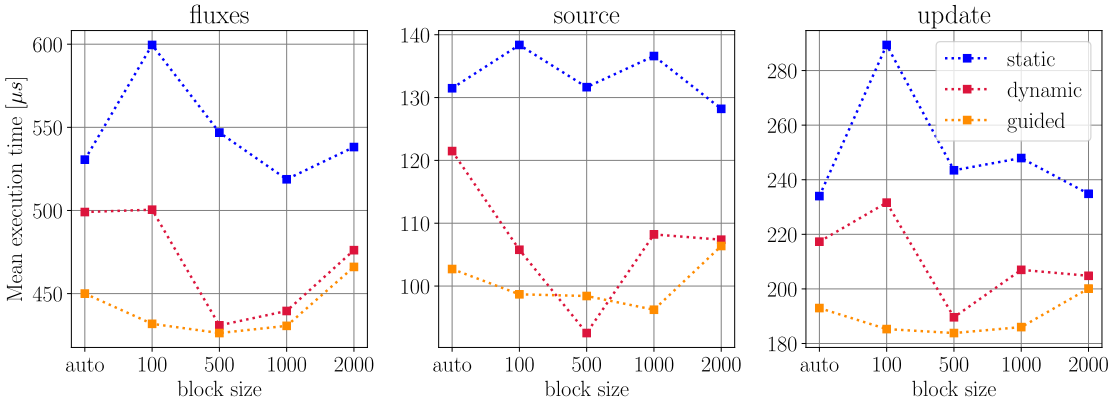

Figure 5.18 – Mean kernel execution time with each scheduling policy

volume time step likely benefit from dynamic scheduling. In contrast, updating the hydraulic variables is more uniform, mainly summing cell flux buffers that are zero for dry cells, making static scheduling more appropriate. To confirm this, we analyzed the mean execution time of each computational kernel during the Toce simulation (Figure 5.18).

We observe that guided scheduling is the most efficient for all steps with the default block size. When varying the block size, timings remain fairly stable, though chunks of 500 iterations appear to be a good choice. However, this value likely depends on the mesh studied, so we retain the default. Increasing the block size brings performance closer to that of guided scheduling but remains more variable. In contrast, the static policy suffers significantly from load imbalance and performs much worse than the other two.

5.5 Memory

5.5.1 Roofline analysis

To gain further insights into the bottlenecks in the program and identify what exactly should be optimized, we should first determine whether the program is *compute bound*, meaning the CPU reaches its floating point functional limits, or *memory bound*, meaning the CPU spends most of its cycles waiting on data loads and stores. This is mainly determined by the *data intensity* of the implemented algorithm, which refers to the ratio of floating point operations to the number of bytes moved. The *Roofline model* [77] relates data intensity to the number of floating point operations per second by showing the maximum achievable performance on the system, depending on the peak achievable bandwidth and the peak performance of the processor.