PHYS52015 Coursework - Part2

Ciaran Reed
Core Ib: Introduction to Scientific and High-Performance Computing

This paper presents the parallelisation of a two-dimensional reaction–diffusion system using MPI. Weak scaling performance was evaluated across multiple process counts, yielding an estimated serial fraction of $f = 0.59(2)$. The results demonstrate good parallel efficiency, with the evaluation of *dxdt* accounting for the majority of the computational cost.

I. Introduction

A serial codebase for simulating the reaction-diffusion system was provided, containing *init*, *dxdt*, *step*, *norm*, and *main*. The system is evaluated over a 2D plane of dimensions M=1024, N=512. MPI was used to decompose the domain into local subregions for parallel execution. Performance was assessed via weak scaling speedup $S(p) = t(n, 1)/t(n, p)$ and efficiency $E(p) = S(p)/p$. Under Gustafson's Law, assuming a serial fraction $0 < f < 1$, the expected speedup is $S(p) = f + (1 - f)p$.

II. Methods

Several strategies exist for decomposing the two-dimensional domain, either along one axis or both. In all cases, neighbouring subdomains exchange boundary data via ghost cells, with the decomposition determining communication volume and memory overhead.

A one-dimensional decomposition forming either a $p \times 1$ or $1 \times p$ process grid requires communication only between neighbouring ranks, with $2p - 2$ messages per time step. Local arrays are sized $[M/p+2] \times [N]$ or $[M] \times [N/p+2]$, where the additional elements store ghost cells.

Since C/C++ arrays are stored in row-major order, decomposing along the first (slow) axis results in contiguous memory layouts, improving cache utilisation and simplifying communication. Splitting along the second axis would require strided communication or intermediate buffers. Therefore, a $p \times 1$ decomposition was selected.

A different approach is to decompose the domain along both axes, forming an $m \times n$ process grid where $mn = p$. In this case, local arrays are of size $[M/m+2] \times [N/n+2]$, which is larger than in the one-dimensional decomposition due to the presence of ghost cells in both dimensions. Diagonally adjacent ranks do not share boundaries and therefore do not exchange data. Each step there are $(2m - 2)n$ or $(2n - 2)m$ messages of size N/n or M/m for horizontal and vertical boundaries respectively; a significantly larger number of smaller messages compared to splitting along a single axis. Since MPI communication performance is often limited by message latency rather than bandwidth for small messages, this approach is likely to exhibit poorer performance.

Several options exist for implementing halo exchanges, including both blocking and non-blocking communication. Communication for the u and v fields can be overlapped, making non-blocking calls a valid option. Alternatively, provided that every send operation has a corresponding receive in the opposite direction, MPI_Sendrecv can be used. This approach is straightforward to implement and avoids deadlocks and data races while remaining performant; therefore, it was selected for this implementation. If u and/or v vary slowly compared to the time step, halo exchanges could be performed every n steps to reduce communication overhead. This would introduce deviations from the reference solution, and was therefore not used.

Difficulties arise when attempting to accurately compute the solution norm, which requires a global reduction operation. Several approaches are possible, including MPI reduction routines, gathering local sums, or gathering entire local arrays onto one rank and performing the summation serially. Using MPI reductions is simple to implement and generally performant, as they exploit available parallelism. However, due to the non-associativity of floating-point addition, the order in which values are summed can vary between runs, leading to small discrepancies in the computed norm when using different numbers of processes. To ensure deterministic and numerically identical results, local arrays were gathered onto a single rank using `MPI_Gather` and `MPI_Gatherv`, and the summation was performed serially. While this approach limits scalability, it was chosen to prioritise numerical accuracy.

III. Results and Discussion

Weak scaling analysis yielded an overall serial fraction of $f = 0.59(2)$. Individual functions were profiled, with *step* including halo exchanges. Measured serial fractions were $init : f = 0.21(8)$, $step : f = 0.66(1)$, $dxdt : f = 0.17(2)$, and $norm : f = 1$. The norm routine exhibits slowdowns with increasing process counts, consistent with its fully serial implementation.

Speedup and efficiency results for the full program are shown in Table I. The decline in efficiency at higher process counts is primarily due to the serial norm computation, which introduces memory and communication overheads on rank 0 that scale with p, imposing a hard scalability limit. Furthermore, increasing p reduces local subdomain sizes, increasing the relative cost of ghost cells and shrinking communication messages, which lowers the computation-to-communication ratio and degrades performance due to latency-dominated MPI communication..

TABLE I: Weak scaling results for the final program, parenthesis indicate error

	Number of cores / Problem size					
Metric	2	4	8	16	32	64
Speedup	1.907(8)	4.0(3)	5.16(9)	9.5(4)	18.1(3)	32.1(2)
Efficiency	0.954(4)	0.99(6)	0.65(1)	0.59(2)	0.57(1)	0.50(2)

IV. Conclusion

By decomposing the domain into a $p \times 1$ process grid along the slow axis, significant parallel speedups were achieved while maintaining efficient memory access patterns. To guarantee numerical reproducibility of the solution norm, a serial summation strategy was adopted, trading scalability for accuracy.

V. Image Appendix

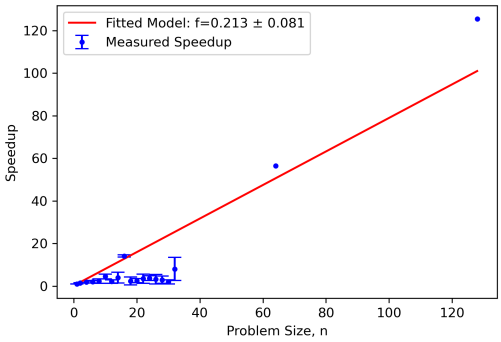

FIG. 1: Weak scaling performance of *init*.

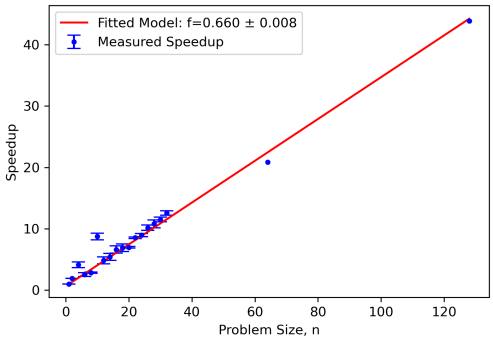

FIG. 2: Weak scaling performance of *step*.

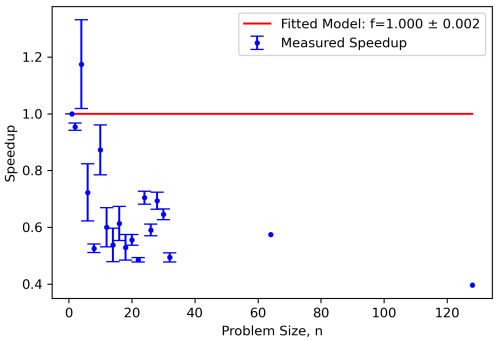

FIG. 3: Weak scaling performance of *norm*.

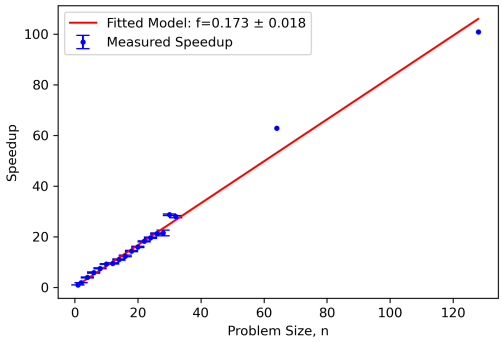

FIG. 4: Weak scaling performance of *dxdt*.

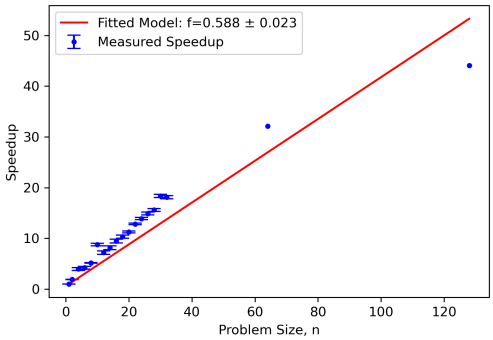

FIG. 5: Weak performance of the final combined program.