Distributed Preconditioned Conjugate Gradient (PCG) with GPU Offload

1. Project Overview

Solve a large SPD system $Ax = b$ arising from a 5-point or 7-point stencil (e.g., Poisson or anisotropic diffusion) using MPI + GPU. Treat the solver as minimizing $f(x) = \frac{1}{2}x^\top Ax - b^\top x$ to emphasize the optimization perspective.

Key Components

- **Matrix-free SpMV:** Implemented via stencil constants $c_1, \ldots, c_5$.
- **Parallelization:** Domain decomposition using MPI with ghost layers.
- **Acceleration:** GPU offload for computational kernels (HIP).
- **Preconditioning:** Start with Jacobi; optionally implement block-Jacobi or Chebyshev.

---

2. Preparation Steps

2.1 Model and Math

- Choose 2-D Poisson or anisotropic diffusion equation.
- Verify the matrix A is SPD.
- Define constants $c_1, \ldots, c_5$ for stencil weights.

2.2 Parallel Layout

- 1D or 2D Cartesian decomposition with ghost layers.
- Choose global sizes (N_x, N_y) large enough to justify MPI.

2.3 File Layout

```
/src
  mpi_domain.hpp/.cpp     # subdomain, halos, Cartesian coords
  pcg.hpp/.cpp            # CG/PCG driver (Krylov loop)
  kernels_hip.hpp/.cpp    # apply_A (stencil), axpy, scal, dot, copy
  timers.hpp              # GPU + MPI timing utilities
/scripts
  run_strong.sh, run_weak.sh, env_rocm.sh
/analysis
  analyze.py or Jupyter notebook for plots
```

2.4 Development Milestones

Day	Goal
1–2	Single-GPU CG (matrix-free `apply_A`) + CPU reference
3–4	Add MPI domain + halos (CPU compute)
5–6	GPU offload for compute kernels (HIP)
7–8	Jacobi preconditioner on GPU; timing & CSV output
9+	Optional: overlap comm/compute, pipelined CG

---

3. Implementation Details

3.1 GPU Kernels

- `apply_A(x, y)`: matrix-free 5-point stencil using $c_1 \dots c_5$.
- BLAS-1: `axpy`, `scal`, `dot`, `copy`.
- Optionally fuse kernels (e.g., `y = A*x + beta*y`).

3.2 MPI Integration

- **Halo exchange:** Pack boundary planes; `MPI_Irecv/Isend`; update ghosts.
- **Reductions:** GPU local reduce → host copy → `MPI_Allreduce`.
- **Overlap:** Compute interior stencil while halos in flight.

3.3 Preconditioner

- Start with **Jacobi** (scaling by diagonal c_1).
- Optionally add **Chebyshev** or **block-Jacobi**.

---

4. PCG Loop (Pseudocode)

```
r = b - A(x);
z = M^{-1} r;       // Jacobi
p = z;
rho = dot(r,z);

for k = 0..maxit:
    halo_exchange(p);
    Ap = A(p);
    alpha = rho / dot(p,Ap);
    x = x + alpha*p;
```

```
    r = r - alpha*Ap;
    if (norm(r)/norm0 < tol) break;
    z = M^{-1} r;
    rho_new = dot(r,z);
    beta = rho_new / rho;
    p = z + beta*p;
    rho = rho_new;
```

---

5. Evaluation Metrics

- **Convergence:** $\|r_k\|_2/\|r_0\|_2$ vs iterations.
- **Strong scaling:** Fixed global N ; time/iter & total time vs ranks.
- **Weak scaling:** Fixed per-rank problem size; near-flat time/iter = good.
- **Roofline analysis:** Report GB/s and arithmetic intensity for `apply_A`.
- **Time breakdown:** % time in SpMV, reductions, halos, preconditioner.

---

6. Common Pitfalls

- Boundary off-by-one errors at subdomain edges.
- Collective reduction overhead for large P.
- GPU-aware MPI performance may vary—test staged copies.

---

7. Recommended References

1. Hestenes & Stiefel (1952): *Methods of Conjugate Gradients for Solving Linear Systems*.
2. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. (free PDF).
3. Ghysels et al. (2014): *Pipelined Conjugate Gradient Method*.
4. Demmel et al. (2012): *Communication-Avoiding Krylov Subspace Methods*.
5. Bell & Garland (2009): *Implementing Sparse Matrix-Vector Multiplication on Throughput Processors*.
6. PETSc documentation (KSPCG, pipelined CG, preconditioning examples).
7. hypre/BoomerAMG GPU support papers (for future extensions).

---

8. Suggested Plots for Report

- Residual vs iteration count (CG vs PCG).
- Time-per-iteration vs ranks (strong scaling).
- Total runtime vs global N (weak scaling).
- Roofline model plot (bandwidth vs arithmetic intensity).
- Pie or bar chart: runtime distribution per kernel.

---

9. Stretch Goals

- **Pipelined CG** (reduces global reductions per iteration).
- **3-D stencil** and multi-GPU per node.
- **Overlapped communication** using MPI and HIP streams.
- **Geometric multigrid** as preconditioner (2-level or full V-cycle).

This document summarizes the end-to-end plan for a distributed, GPU-accelerated PCG solver suitable as a final project for APMA 2822B.