The following are a summary of relevant sparse linear solvers available from PETSc (I included what I thought we will use based on what I learned from the thesis).
In short, the types of matrices are:

- sparse matrices
- block sparse matrices
- sequential sparse matrices, based on compressed sparse row format
- sequential block sparse matrices, based on block sparse compressed row format

1 Krylov Methods: Conjugate Gradients (KSPCG)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPCG.html`
The Preconditioned Conjugate Gradient (PCG) iterative method.

- Requires both the matrix and preconditioner to be symmetric positive (or negative) (semi) definite.
- Only left preconditioning is supported.
- Parallel and complex.
- For complex numbers there are two different CG methods, one for Hermitian symmetric matrices and one for non-Hermitian symmetric matrices.

2 Krylov Methods: GMRES (KSPGMRES)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPGMRES.html`
Left and right preconditioning are supported, but not symmetric preconditioning.

3 CG for Least Squares (KSPCGLS)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPCGLS.html` Conjugate Gradient method for Least-Squares problems

- Supports non-square (rectangular) matrices.
- Parallel and complex.

4 Preconditioner: Jacobi (PCJACOBI)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCJACOBI.html` Diagonal scaling preconditioning.
Parallel and complex.
Matrix Types:

1. MATAIJ = "aij" - A matrix type to be used for sparse matrices.
2. MATBAIJ = "baij" - A matrix type to be used for block sparse matrices.
3. MATSBAIJ = "sbaij" - A matrix type to be used for symmetric block sparse matrices.
4. MATDENSE = "dense" - A matrix type to be used for dense matrices.

5 Preconditioner: Point Block Jacobi (PCPBJACOBI)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCPBJACOBI.html`
Uses dense LU factorization with partial pivoting to invert the blocks; if a zero pivot is detected a PETSc error is generated.
Parallel and complex.
Matrix Types:

1. AIJ
2. BAIJ

6 Preconditioner: Block Jacobi (PCBJACOBI)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCBJACOBI.html`
Use block Jacobi preconditioning, each block is (approximately) solved with its own KSP object.
Each processor can have one or more blocks, or a single block can be shared by several processes. Defaults to one block per processor.
Parallel and complex.
Matrix Types:

1. AIJ
2. BAIJ
3. SBAIJ

7 Preconditioner: ILU (PCILU)

`http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCILU.html`

For BAIJ matrices this implements a point block ILU.
Parallel and complex.
Matrix Types:

1. MATSEQAIJ = "seqaij" - A matrix type to be used for sequential sparse matrices, based on compressed sparse row format.
2. MATSEQBAIJ = "seqbaij" - A matrix type to be used for sequential block sparse matrices, based on block sparse compressed row format.

8 Direct Solvers: Cholesky (PCCHOLESKY)

http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCCHOLESKY.html
Uses a direct solver, based on Cholesky factorization, as a preconditioner.
Usually this will compute an "exact" solution in one iteration.
Matrix Types:

1. seqaij
2. seqbaij

9 Matrix Collection

Set of widely used set of sparse matrix benchmarks collected from a wide range of applications. Link: https://sparse.tamu.edu/

10 Free Linear Algebra Software

http://www.netlib.org/utk/people/JackDongarra/la-sw.html
http://www.netlib.org/lapack/#_presentation

CG using Jacobi preconditioner:
https://www.npmjs.com/package/conjugate-gradient

Iterative methods library:
https://math.nist.gov/iml++/

11 CG

Github repository: High Performance Computing Conjugate Gradients: The original Mantevo miniapp
https://github.com/Mantevo/HPCCG

- Generates a 27-point finite difference matrix with a user-prescribed sub-block size on each processor
- Code compiles with MPI support and can be run on one or more processors
- Input: nx, ny, nz are the number of nodes in the x, y and z dimension respectively on a each processor. The global grid dimensions will be nx, ny and numproc * nz. In other words, the domains are stacked in the z direction.

12 B-CG

Paper: An Implementation of Block Conjugate Gradient Algorithm on CPU-GPU Processors
Link to paper:
`https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7017966`