High-**P**erformance **C**omputing 20**24**

PDEs (Heat, Poisson’s & Laplace’s Eq.) and Mathematical Software

Aryan Eftekhari, Olaf Schenk | **U**niversità della **S**vizzera **i**taliana (USI) | **I**nstitute of **C**omputing (CI)

Motivation

PDEs – Heat, Poisson’s & Laplace’s Equation

Poisson’s, **Laplace’s**, and **Heat** equations are fundamental for modeling essential physical phenomena.

Their relationship can be enlightening …

Heat, Poisson's & Laplace's Equation
Definitions

Heat Equation
Describes the diffusion of a quantity (possibly heat) in a given region over time.

$$\boxed{\frac{\partial u}{\partial t} = \Delta u}$$

Poisson's Equation
Describes potential fields influenced by a given source (**source term**).

$$\boxed{\Delta u = \boxed{f}}$$

Written out as ...

$$\frac{\partial u}{\partial t} = \boxed{\alpha}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

A positive coefficient called "thermal diffusivity".

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = \boxed{f(x, y, z)}$$

*If **source term** is equal to zero, the equation is called "**Laplace's Equation**".*

Heat, Poisson’s & Laplace’s Equation
We have seen this before …

Heat Equation: Can be viewed as a special case of *Fisher’s Equation*.

Fisher's Equation (reminder):

Used to describe biological populations:
spatial diffusion with **reaction/growth**.

$$\frac{\partial s}{\partial t} = \delta \Delta s + \rho s(1 - s)$$

Heat Equation

$$\boxed{\frac{\partial s}{\partial t} = \delta \Delta s} + \cancel{\rho s(1 - s)}$$

* Notice that “S” is function of space, and time!

Ok but what what about *Poisson’s*, and *Laplace’s Equations* … how are they related?

Heat, Poisson’s & Laplace’s Equation

Steady-State of Heat Equation

Consider the Heat Equation with **source term**

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \Delta u(x,y,t) + \boxed{f(x,y)}$$

What will the “*steady-state*” solution (i.e. $t \to \infty$) look like ?

Heat, Poisson's & Laplace's Equation

Steady-State of Heat Equation

Poisson's Equation: Steady-state solution of "Heat Equation with a source".

The **steady-state** solution will follow:

$$\boxed{\frac{\partial u(x,y,t)}{\partial t}} = \alpha \Delta u(x,y,t) + f(x,y) = \boxed{0} \Rightarrow \Delta u(x,y,t) = -\frac{1}{\alpha} f(x,y)$$

In steady-state there is **no change induced by time**:

$$\boxed{\Delta u(x,y,\cancel{t}) = -\frac{1}{\alpha} f(x,y)}$$

Heat, Poisson's & Laplace's Equation
Steady-State of Heat Equation w. Zero Source Term

Laplace's Equation: Steady-state solution of "Heat Equation without a source".

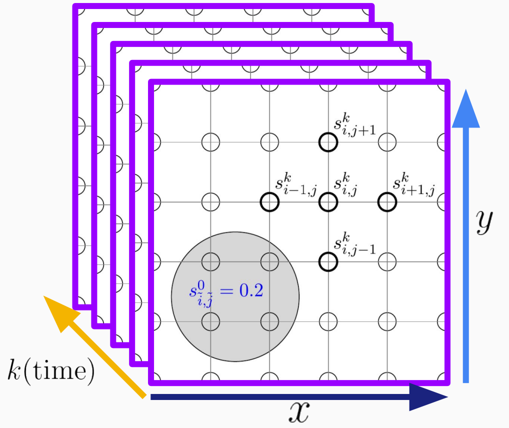
Consider the Heat Equation with **no heat source**:

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \Delta u(x,y,t) + \cancel{f(x,y)}$$

In steady-state there is **no change induced by time**:

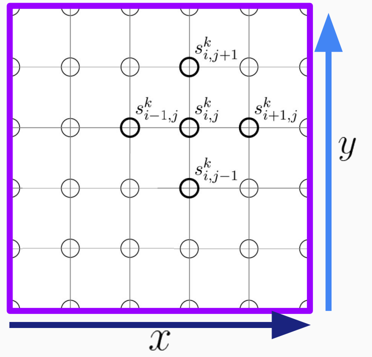
$$\boxed{\Delta u(x,y,\cancel{t}) = 0}$$

Solution Method
Basic Steps



Time-Dependent (e.g., Heat, Fisher's Eq.)

1. **Formulation & Representation:** Model the domain, **initial condition**, and **boundaries conditions**.
2. **Discretization Space & Time:** Convert the continuous problem into a set of discrete equations.
3. **Solve Space & Time:** Solution over domain per time-step.



Time-Independent (e.g., Poisson's and Laplace's Eq.)

1. **Formulation & Representation:** Model the domain and **boundaries conditions**.
2. **Discretization Space:** Convert the continuous problem into a set of discrete equations.
3. **Solve Space:** Solution over domain.

Solution Method

Concept of Linearity

Fisher's Equation (Nonlinear)

$$\frac{1}{\tau}(s^k_{i,j} - s^{k-1}_{i,j}) = \underbrace{\frac{\delta}{h^2}\left(-4s^k_{i,j} + s^k_{i+1,j} + s^k_{i-1,j} + s^k_{i,j+1} + s^k_{i,j-1}\right)}_{\textbf{linear}} + \underbrace{\boxed{\rho s^k_{i,j}(1 - s^k_{i,j})}}_{\textbf{nonlinear}}$$

The solution is than the <u>root</u> of:

$$f(\mathbf{s}^k|\mathbf{s}^{k-1}, \mathbf{A}, c_1, c_2) := \mathbf{s}^k - \mathbf{s}^{k-1} - c_1\mathbf{A}\mathbf{s}^k - c_2\mathbf{s}^k \cdot (1 - \mathbf{s}^k)$$

Use Newton's Method → <u>multiple linear solves</u>

How would this change for Poisson's and Laplace's Equation?

Solution Method
Concept of Linearity

Poisson's Equation (Linear):

For example consider a forcing function $\sin(x, y)$ which is discretized to $\mathbf{b} = [\sin(x_0, y_0), \sin(x_1, y_0), \ldots, \sin(x_{N-1}, y_{N-1})]$.

The solution is than the root of:

$$\Delta u(x, y) = \sin(x, y) \Rightarrow \; f(\mathbf{u}|\mathbf{A}, c) = \boxed{\mathbf{A}\mathbf{u} - c\mathbf{b}}$$

Or simply single linear solve

… and Laplace's Equation?

Solution Method

Pseudo Algorithm

Nonlinear w. time stepping

```
Input u_initail_value, K, iter_max, eps

// Initial Conditions
u_last ← u_initail_value
u      ← u_last

// Time loop
For k = 1 to K
     // Newton loop
     For iter=1 to iter_max
          // Linear Solve
          update ← lin_solve(J(u|u_last),f(u|u_last))
          u ← u - update
          // Convergence Check
          If norm(update)<eps
          break
          Endif
     Endfor
     // Swap Solution
     u_last ← u
Endfor

Return u
```

Linear wo. time stepping

```
Input ~~u_initail_value, K, iter_max, eps~~

// Initial Conditions
~~u_last ← u_initail_value~~
~~u      ← u_last~~

~~// Time loop~~
~~For k = 1 to K~~
     ~~// Newton loop~~
     ~~For iter=1 to iter_max~~
          // Linear Solve
          u ← lin_solve(A,b)
          ~~u ← u - update~~
          ~~// Convergence Check~~
          ~~If norm(update)<eps~~
          ~~break~~
          ~~Endif~~
     ~~Endfor~~
     ~~// Swap Solution~~
     ~~u_last ← u~~
~~Endfor~~

Return u
```

Solution Method
Linear Solve

Linear solve is central to the solution of both **linear** and **nonlinear** PDEs.

Direct Methods: Solve matrices in fixed steps with notable stability, especially for well-conditioned systems.

Utilize matrix factorizations (e.g., **LU**, **Cholesky**, **QR**) to reduce the problem to simpler triangular systems, which can then be rapidly solved through back-substitution.

Iterative Methods: Memory-efficient with *adjustable accuracy,* though they demand careful considerations for stability. For example:

1. Conjugate Gradient (CG): An iterative method for symmetric positive-definite systems of linear equations.

2. **Generalized Minimal Residual (GMRES)**: An iterative method for indefinite nonsymmetric system of linear equations.

In the examples we have seen, **CG** is very much applicable.

For more on CG, see "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain"

Toolkits
PETSc

Steps: (**i**) assemble matrices/vectors , (**ii**) solve and (**iii**) maybe repeat

… a toolkit would really help!

Portable, **E**xtensible **T**oolkit for **S**cientific **C**omputation: Scalable (parallel) solution of scientific applications modeled by partial differential equations (PDEs).

Supports MPI and GPU acceleration through CUDA, HIP, Kokkos, or OpenCL, as well as hybrid MPI-GPU parallelism

Other frameworks exist such as Trilinos

Toolkits

PETSc Code Structure

```
int main(int argc, char **argv){

// Boilerplate code for initial setup
PetscCall(KSPCreate(PETSC_COMM_WORLD, &ksp));
PetscCall(KSPSetComputeRHS(ksp, ComputeRHS, &user));
PetscCall(KSPSetComputeOperators(ksp, ComputeMatrix, NULL));
PetscCall(KSPSetDM(ksp, da));
PetscCall(KSPSetFromOptions(ksp));
PetscCall(KSPSetUp(ksp));
...
}

PetscErrorCode ComputeMatrix(KSP ksp, Mat A, Mat P, void *ctx) {
 DM             da;
 DMDALocalInfo info;

 // Retrieve the distributed array, grid information, and global grid dimensions
 PetscCall(KSPGetDM(ksp, &da));
 PetscCall(DMDAGetLocalInfo(da, &info));
 ...
}

PetscErrorCode ComputeRHS(KSP ksp, Vec b, void *ctx) {
 DM             da;
 DMDALocalInfo info;

 // Retrieve the distributed array, grid information, and global grid dimensions
 PetscCall(KSPGetDM(ksp, &da));
 PetscCall(DMDAGetLocalInfo(da, &info));
 ...
}
```

Many tutorials available …
Use them !

Università della Svizzera italiana
Institute of Computing CI

High-Performance Computing, Fall 2024
Lecturer: Dr. A. Eftekhari, Prof. O. Schenk
Assistants: M. Lechekhab, D. Vega,
D. Santarsiero, J. Palumbo, I. Ecevit

Project 7
HPC Mathematical Software for Extreme-Scale Science

Due date: 20 December 2024 at 23:59 (See iCorsi for updates)

In this project, we provide you with a non-exhaustive list of HPC software toolkits and libraries for CSE that you may find useful in your CSE studies and beyond. The task of this project is then to solve a simple boundary value problem (Poisson equation with Dirichlet boundary condition) with Python and the PETSc toolkit.

1 High-Level Languages for Numerical Computations

Using numerical methods and reference implementations from popular textbooks is often not sufficient for the development of serious software for large-scale applications in computational science. In general, state-of-the-art algorithms are not covered by these books and neither are the corresponding implementations suited to achieve a reasonable performance on multicores. Software packages like Mathematica [28], Maple [20], Matlab [19] provide high-level programming languages that can support the initial development of new numerical methods and the rapid prototype implementation. However, these standard packages are often not sufficient as HPC kernels, and neither are they intended for the realization of scientific software that has to deal with large-scale parallel applications on an HPC platform.

2 HPC software frameworks

2.1 Non-exhaustive overview of software toolkits and libraries for HPC-CSE

In the following, we describe various well-known high-performance computing mathematical software libraries that have influenced computational science and engineering (CSE). All parallel software toolkits can be used as an underlying layer in computational science to build successful parallel applications on multi- and many-core architectures. You may also find interesting the article by Rüde et al. [23] on research and education in CSE. [1] Enabling parallel technologies in computational science must also ensure that a successful high-performance software library is fully and freely available for use throughout the scientific computing community.

Parallel Dense Matrix Software Libraries

BLAS The Basic Linear Algebra Subprogramm (BLAS) [10] specifies a set of computational routines for linear algebra operations such as vector and matrix operations. BLAS is a well-known example of a software layer which represents a de facto standard in the field of HPC. All hardware vendors, such as Intel on multicores, or NVIDIA on manycores provide these BLAS routines as building blocks to ensure efficient scientific software portability.

LAPACK The Linear Algebra Package (LAPACK) [2] is a software library for matrix operations such as solving systems of linear equations, least-squares problems, and eigenvalue or singular value problems. Almost all LAPACK routines make use of BLAS to perform efficient computations — LAPACK acts as an additional parallel software layer to the underlying BLAS. The motivation for the development of BLAS and LAPACK was to provide efficient and portable implementations for solving dense systems of linear equations and least-squares problems and it was initiated in the 1970s. Influenced by the changes in the hardware architectures (e.g., vector computers, cache-based processors to parallel multiprocessing architectures), it changed from the original specification to, finally, BLAS Level-3 [9] for matrix-matrix operations.

[1] Some of these software packages, e.g., IPOPT, METIS, PETSc, Trilinos, or PARDISO have been selected as important tools for future HPC applications.

ScaLAPACK Scalable LAPACK (ScaLAPACK) [6] is an extension of LAPACK that is targeted to multiprocessing computers with a distributed-memory hierarchy. Fundamental building blocks of ScaLAPACK are the parallel BLAS (PBLAS) and the Basic Linear Algebra Communication Subprograms (BLACS). PBLAS is a distributed-memory version of BLAS, and BLACS is a library for handling interprocessor communication. The design policy of ScaLAPACK was to have the ScaLAPACK routines similar to their LAPACK equivalents so that both libraries can be used as a computational software layer for application developers.

SLATE The Software for Linear Algebra Targeting Exascale (SLATE) aims at being an extension of LAPACK for current and upcoming distributed high-performance systems, both accelerated CPU-GPU based and CPU based. It will serve as a replacement for ScaLAPACK, which after two decades cannot adequately be adapted for modern accelerated architectures. [12].

Parallel Sparse Matrix Software Toolkits

A number of applications give rise to large-scale sparse linear systems that are becoming exceedingly difficult to handle by standard numerical linear algebra techniques such as 3D wave propagation problems or PDE-constrained optimization problems. A number of highly efficient parallel software tools have been developed within the last fifteen years that can be used to tackle large-scale systems of linear equations or eigenvalue problems in parallel. These tools usually make heavy use of the BLAS and LAPACK routines, and all these parallel software tools now represent a de facto standard in the field of HPC. The following is a list of the most widely use of the tools:

MUMPS The Multifrontal Massively Parallel Solver (MUMPS) [1] is being developed at CERFACS, ENSEEIHT-IRIT, and INRIA. It is a package for solving systems of linear equations of the form $Ax = b$, where A is a square sparse matrix that can be either unsymmetric, symmetric positive definite, or general symmetric. MUMPS is a direct method based on a multifrontal direct factorization. It exploits both parallelism arising from sparsity in the matrix A and from dense factorizations kernels. MUMPS offers several built-in ordering algorithms to some external parallel ordering packages such as METIS [16]. The parallel version of MUMPS requires MPI for message passing and makes use of the BLAS, BLACS, and ScaLAPACK libraries.

SuperLU SuperLU [17, 18] is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high-performance machines. It has been developed at the Computer Science Department of the University of California, Berkeley and at the National Energy Research Scientific Computing Center (NERSC). The library is written in C and is callable from either C/ C++ or Fortran. The library routines will perform an LU decomposition with partial pivoting and triangular system solves through forward and back substitution. The matrix columns may be preordered (before factorization) either through library or user-supplied routines. SuperLU comes in three different flavors: SuperLU for sequential machines, SuperLU_MT for shared memory parallel machines, and SuperLU_DIST for highly parallel distributed-memory architectures. The parallel version of SuperLU also requires MPI for the message-passing layer and makes use of the BLAS, BLACS, and ScaLAPACK libraries.

PARDISO PARDISO [24] is a thread-safe, high-performance, robust, memory efficient, easy-to-use software for solving large sparse symmetric and nonsymmetric linear systems of equations on multicore processing architectures. The solver is also part of the Intel Math Kernel Library with interfaces to Matlab, C, C++, and Python.

HYPRE HYPRE [11] is a library of preconditioners and solvers using multigrid methods for solving large, sparse systems of linear equations on massively parallel computers.

FEAST The FEAST library package [13, 21] represents a unified framework for solving various families of eigenvalue problems and achieving accuracy, robustness, high-performance and scalability on parallel architectures. Its originality lies with a new transformative numerical approach to the traditional eigenvalue algorithm design - the FEAST algorithm. The FEAST algorithm is a general purpose eigenvalue solver which takes its inspiration from the density-matrix representation and contour integration technique in quantum mechanics. FEAST can be used for solving both standard and generalized forms of the Hermitian or non-Hermitian problems (linear or nonlinear), and it belongs to the family of contour integration eigensolvers. FEAST's main computational task consists of a numerical quadrature computation that involves solving independent linear systems along a complex contour, each with multiple right-hand sides. FEAST is both a comprehensive library package, and an easy to use software. It includes flexible reverse communication interfaces and ready to use driver interfaces for dense, banded, and sparse systems.

Software Toolkits for Parallel Large-Scale Nonlinear Optimization

IPOPT Interior Point OPTimizer (IPOPT; pronounced eye-pea-Opt) [27] is a software package for large-scale nonlinear optimization. IPOPT is a C++ open-source software package based on a Newton-based interior point algorithm with filter line-search method. It has been designed for equality constrained problems and treats upper and lower bounds for the variables with an interior penalty formulation as a sequence of barrier problems for a decreasing sequence of barrier parameters converging to zero. It can be shown that under mild regularity assumptions the solutions of the barrier problems will converge to the solution of the original problem. The IPOPT algorithm has been shown to be globally and superlinearly convergent under weaker assumptions than other barrier methods. IPOPT has been applied to thousands of test problems and applications and has been adopted by a widespread user community. A key advantage is its ability to use second derivative information efficiently. IPOPT makes use of PARDISO or MUMPS, so it can also require MPI for message passing and makes use of the BLAS, BLACS, and ScaLAPACK libraries. Carl Laird and Andreas Wächter are the developers of IPOPT. Wächter and Laird were awarded the 2011 J. H. Wilkinson Prize for Numerical Software for this development .[2]

Software Toolkits for Parallel Partitioning, Load Balancing, and Data-Management Services

Parallel partitioning, load balancing, and data-management services are important enabling technologies for parallel computing. Their goal is to distribute work evenly to processors while creating decompositions that have low communication costs for applications. This distribution must be done statically as a first step in most parallel computations. For adaptive computations, where processor workloads change as computations proceed, dynamic load balancing may also be needed. Two such toolkits are, e.g., the following:

Zoltan The Zoltan toolkit [7] is a C++ open-source software collection of such data management services for parallel unstructured, adaptive, and dynamic applications, available as open-source software from the Sandia National Laboratories. The design goal is to simplify the load balancing, data movement, and unstructured communication that arise in dynamic applications such as adaptive finite element methods, particle methods, and multiphysics simulations.

METIS The METIS [14, 15, 16] is another alternative for parallel partitioning, load balancing, and data-management services. It can provide efficient parallel partitioning of graphs with sizes up to a billion vertices, distributed over a thousand processors. METIS includes routines that are especially suited for parallel AMR computations and large-scale numerical simulations. The algorithms implemented in METIS are based on the parallel multilevel k-way graph-partitioning, adaptive repartitioning, and parallel multi-constrained partitioning schemes.

Extreme-Scale Software Framework for Multiphysics Engineering and Scientific Problems

Trilinos The Trilinos Project [25] is a C++ open-source software package that represents an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multiphysics engineering and scientific problems. A unique design feature of Trilinos is its heavy focus on other similar packages within a computational software layer stack. Trilinos has been under development at Sandia National Laboratories since 2002, and it provides uniform access to accurate, robust, and efficient solvers and tools. It also facilitates more rapid development of new libraries by providing important core functionality and software engineering processes for developers.

PETSc The Portable, Extensible Toolkit for Scientific Computation (PETSc; pronounced PET-see; the S is silent) [5], is a suite of data structures and routines developed by Argonne National Laboratory for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the Message Passing Interface (MPI) standard for all message-passing communication. PETSc is probably the world's most widely used parallel numerical software library for PDEs and sparse matrix computations. The PETSc Core Development Group won the SIAM/ACM Prize in Computational Science and Engineering for 2015.[3] PETSc is intended for use in large-scale application projects; many ongoing computational science projects are built around the PETSc libraries. Its careful design allows advanced users to have detailed control over the solution process.

[2] https://www.coin-or.org/2011/04/07/ipopt-wins-the-wilkinson-prize-for-numerical-software/
[3] https://www.siam.org/programs-initiatives/prizes-awards/major-prizes-lectures/siamacm-prize-in-computational-science-and-engineering/

PETSc includes a large suite of parallel linear and nonlinear equation solvers that are easily used in application codes written in C, C++, Fortran, and now Python [8]. PETSc provides many of the mechanisms needed within parallel application code, such as simple parallel matrix and vector assembly routines that allow the overlap of communication and computation. In addition, PETSc includes support for parallel distributed arrays useful for finite-difference/volume/element methods.

MFEM MFEM [3] is an open-source and scalable C++ library for finite element methods that provides arbitrary high-order finite element meshes and spaces, supports a wide variety of discretization approaches, and focuses on ease of use, portability, and HPC. The goal of MFEM is to provide application scientists with access to state-of-the-art algorithms for high-level finite element meshes, discretizations, and linear solvers, while enabling researchers to quickly and easily develop and test new algorithms in very general, fully unstructured, high-level, parallel, and GPU-accelerated environments.

AMReX AMReX [29] is a C++ software framework that supports the development of block-structured adaptive mesh refinement (AMR) algorithms for solving multiphysics applications on current and new architectures.

2.2 The Poisson Equation with PETSc

Many books on programming languages start with a "Hello, World!" program. Readers are curious to know how fundamental tasks are expressed in the language, and printing a text to the screen can be such a task. In the world of finite-difference methods for PDEs and HPC, one of the most fundamental tasks is to solve the Poisson equation. Our counterpart to the classical "Hello, World!" program therefore is to solve the Poisson equation with Dirichlet boundary conditions

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{in} \quad \partial\Omega. \end{aligned} \tag{1}$$

Here, $u = u(x_1, x_2)$ is the unknown function, $f = f(x_1, x_2)$ is a prescribed source function, Δ is the Laplace operator, Ω is the spatial domain, and $\partial\Omega$ is the boundary of Ω. We denote by $\overline{\Omega}$ the corresponding closed set $\overline{\Omega} = \Omega \cup \partial\Omega$. The Poisson equation Eq. (3) arises in numerous physical contexts, including heat conduction, electrostatics, Newtonian gravity, diffusion of substances, twisting of elastic rods, non-viscous fluid flow, and water waves.

For simplicity, we restrict ourselves to two dimensions $\Omega \subset \mathbb{R}^2$. The Laplace operator in Cartesian coordinates is then given by

$$\Delta \equiv \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}. \tag{2}$$

Solving a boundary-value problem such as the Poisson equation using a second-order finite difference discretization, and deploying scalable mathematical algorithms and software tools for reliable parallel simulation consists of the following steps:

(i) Identify the computational domain (Ω), the PDE modeling the system under study, its boundary conditions, and source terms $f(x_1, x_2)$.

(ii) Discretize the domain with a regular grid (with potentially different number of grid points in the x and y directions) and the PDE with second-order centered finite-differences and reformulate the discrete problem into a linear system to solve.
Hint: See Ascher and Greif [4, Example 7.1] and/or Tveito and Winther [26, Sec. 7.5.1].

(iii) Select an appropriate mathematical software for extreme-scale computing. In this sub-project, we choose PETSc [4].

2.2.1 Environment setup

To get started on Rosa cluster, load the necessary modules and activate the conda environment as shown below:

[4]PETSc is pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable parallel solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs, as well as hybrid MPI-GPU parallelism.

```
[user@icslogin01 hello_petsc]$ source /apps/miniconda3/bin/activate
[user@icslogin01 hello_petsc]$ conda activate /apps/miniconda3/envs/petsc/
[user@icslogin01 hello_petsc]$ module load gcc openmpi
```

Next, you need to set the `PETSC_DIR` environment variable to indicate the full path of the PETSc home directory

```
[user@icslogin01 hello_petsc]$ export PETSC_DIR=/apps/miniconda3/envs/petsc
```

The available PETSc version on the Rosa cluster may not be the most recent release of PETSc, but this is not an issue for the current project. Now you should be able to compile and run the PETSc test program provided in the skeleton code directory `hello_petsc` as follows:

```
[user@icslogin01 hello_petsc]$ make
...
[user@icslogin01 hello_petsc]$ salloc -n 16 --nodes=4 --ntasks-per-node=4
↪  --reservation=hpc-tuesday
 # allocate 16 processes on 4 nodes with 4 processes each
...
[user@icslogin01 hello_petsc]$ mpirun ./hello_petsc
Hello world from rank 8 out of 16 on icsnode20
Hello world from rank 11 out of 16 on icsnode20
Hello world from rank 10 out of 16 on icsnode20
Hello world from rank 9 out of 16 on icsnode20
Hello world from rank 1 out of 16 on icsnode18
Hello world from rank 2 out of 16 on icsnode18
Hello world from rank 3 out of 16 on icsnode18
Hello world from rank 0 out of 16 on icsnode18
Hello world from rank 14 out of 16 on icsnode21
Hello world from rank 12 out of 16 on icsnode21
Hello world from rank 13 out of 16 on icsnode21
Hello world from rank 15 out of 16 on icsnode21
Hello world from rank 7 out of 16 on icsnode19
Hello world from rank 6 out of 16 on icsnode19
Hello world from rank 5 out of 16 on icsnode19
Hello world from rank 4 out of 16 on icsnode19
```

You are now all set with PETSc!

2.2.2 The project tasks

- Please read the paper [22] entitled "Challenges and Opportunities in CSE Research," in particular, section 2 on "Challenges and Opportunities in CSE Research," "CSE and High-Performance Computing," and "CSE Software".
- Your task is to solve with PETSc the following boundary value problem

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{in} \quad \partial\Omega \end{aligned} \tag{3}$$

  with constant source function $f = f(x_1, x_2) = 20$ in the unit square

$$\Omega = \{x = (x_1, x_2) \mid 0 < x_1, x_2 < 1\}. \tag{4}$$

  1. Discretize the boundary value problem with second-order centered finite-differences and reformulate the discrete problem into a linear system of equations to solve. Summarize your discretization in your report.
  2. Implement the solution to the boundary problem above in Python (Use `poisson_py.py` as a template) [25 points] . The solution should run correctly for grids with different number of grid points in the x and y dimensions. The Python implementation use three different solvers:

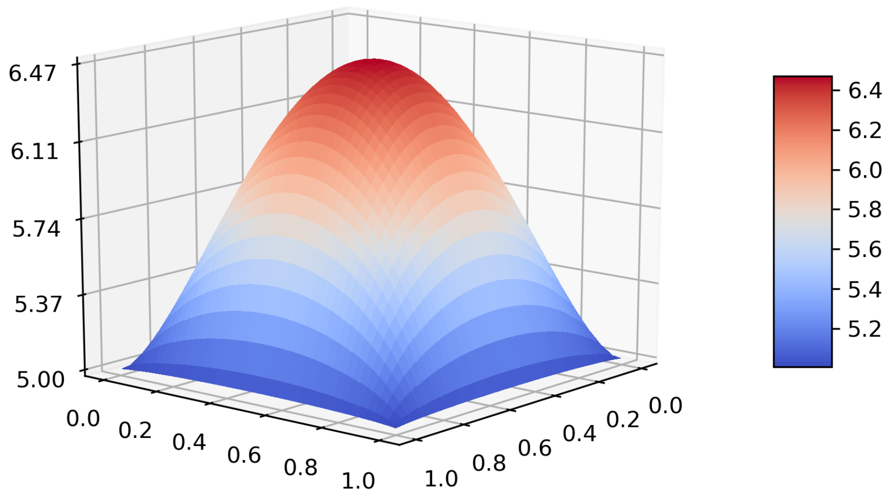


Figure 1: Solution of the Poisson equation from (3) using different f and boundary condition.

`sp_dir`, the Scipy Direct Sparse Solver (utilizing `COLAMD`[5]); `dn_dir`, the Numpy dense solver using the `LAPACK`[6]; and `sp_cg`, the Scipy Conjugate Gradient (CG) solver[7]. **Hint**: you might need to use the module py-pip to install python packages for this project (e.g. matplotlib).

3. Implement the solution to the boundary problem with PETSc (Use `poisson_petsc.c` as a template) [25 points]. The solution should run correctly for grids with different number of grid points in the x and y dimensions. **Hint**: Read the Getting Started section of the PETSc Manual. In particular, the section on developing an application program that uses PETSc. For instance, ex2 or ex29 could be of interest.

4. Validate and Visualize [10 points]: Run the test in `test_val` to validate the results (all solutions should have the same norm) and visualize your approximate solution $u(x_1, x_2)$ using `draw.py`. Ensure to include the PETSc visualization results in your report.

5. Performance Benchmark [15 points]: Run the test in `test_perf` for grid sizes $\{8, 16, \ldots, 512\}$ (equal in both x and y) to evaluate the *non-parallel* performance of the PETSc and Python implementations. Provide a *log-log* plot in your report comparing the runtimes versus the mesh grid sizes. Note that the PETSc implementation should be run with the Conjugate Gradient (CG) solver, configured using the `-ksp_type` flag. For more details, refer to the code in `poisson_petsc.c` and the PETSc documentation. *For all tests, if the runtime exceeds approximately* 10 *seconds, you can eliminate the execution of larger mesh sizes for that specific solver.*

6. Strong Scaling [10 points]: Run the test in `test_scal` for a grid size of 1024 to evaluate the parallel scalability of the PETSc implementation using $\{2, 4, 8, 16\}$ MPI processes. Provide a *log-log* plot of the total runtime versus the number of processes. Use the PETSc Conjugate Gradient. solver for this test.

3 Quality of the Report [15 Points]

Each project will have 100 points (out of 15 point will be given to the general written quality of the report).

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.spsolve.html
[6] https://numpy.org/doc/2.1/reference/generated/numpy.linalg.solve.html
[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html

Additional notes and submission details

Submit the source code files (together with your used `Makefile`) in an archive file (tar, zip, etc.) and summarize your results and the observations for all exercises by writing an extended Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to iCorsi.

- Your submission should be a gzipped tar archive, formatted like project_number_lastname_firstname.zip or project_number_lastname_firstname.tgz. It should contain:
  - All the source codes of your solutions.
  - Build files and scripts. If you have modified the provided build files or scripts, make sure they still build the sources an run correctly. We will use them to grade your submission.
  - project_number_lastname_firstname.pdf, your write-up with your name.
  - Follow the provided guidelines for the report.
- Submit your .tgz through iCorsi.

Code of Conduct and Policy

- Do not use or otherwise access any on-line source or service other than the iCorsi system for your submission. In particular, you may not consult sites such as GitHub Co-Pilot or ChatGPT.
- You must acknowledge any code you obtain from any source, including examples in the documentation or course material. Use code comments to acknowledge sources.
- Your code must compile with a standard-configuration C/C++ compiler.

Please follow these instructions and naming conventions. Failure to comply results in additional work for the TAs, which makes the TAs sad...

References

[1] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures. *ACM Transactions on Mathematical Software*, 45(1):2:1–2:26, February 2019. ISSN 0098-3500. doi: 10.1145/3242094. URL https://dl.acm.org/doi/10.1145/3242094.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, January 1999. ISBN 9780898719604. doi: 10.1137/1.9780898719604.

[3] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, jan 2021. doi: 10.1016/j.camwa.2020.06.009.

[4] Uri M. Ascher and Chen Greif. *A First Course in Numerical Methods*. Society for Industrial & Applied Mathematics (SIAM), June 2011. doi: 10.1137/9780898719987.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc Web page. https://petsc.org/, 2024. URL https://petsc.org/.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, January 1997. ISBN 9780898719642. doi: 10.1137/1.9780898719642.

[7] Erik G. Boman, Ümit V. Çatalyürek, Cédric Chevalier, and Karen D. Devine. The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering and Coloring. *Scientific Programming*, 20(2):129–150, 2012. ISSN 1058-9244. doi: 10.3233/SPR-2012-0342. URL https://www.hindawi.com/journals/sp/2012/713587/.

[8] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. ISSN 0309-1708. doi: 10.1016/j.advwatres.2011.04.013. New Computational Methods and Software Tools.

[9] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990. ISSN 0098-3500. doi: 10.1145/77626.79170. URL https://dl.acm.org/doi/10.1145/77626.79170.

[10] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1): 1–17, March 1988. ISSN 0098-3500. doi: 10.1145/42288.42291. URL https://dl.acm.org/doi/10.1145/42288.42291.

[11] Robert D. Falgout, Jim E. Jones, and Ulrike Meier Yang. The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners. In Are Magnus Bruaset and Aslak Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 267–294, Berlin, Heidelberg, 2006. Springer. ISBN 9783540316190. doi: 10.1007/3-540-31619-1_8.

[12] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. SLATE: design of a modern distributed and accelerated linear algebra library. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, pages 1–18, New York, NY, USA, November 2019. Association for Computing Machinery. ISBN 9781450362290. doi: 10.1145/3295500.3356223. URL https://dl.acm.org/doi/10.1145/3295500.3356223.

[13] Brendan Gavin, Agnieszka Międlar, and Eric Polizzi. FEAST eigensolver for nonlinear eigenvalue problems. *Journal of Computational Science*, 27:107–117, July 2018. ISSN 1877-7503. doi: 10.1016/j.jocs.2018.05.006. URL https://www.sciencedirect.com/science/article/pii/S1877750318302096.

[14] G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pages 28–28, November 1998. doi: 10.1109/SC.1998.10018. URL https://ieeexplore.ieee.org/document/1437315.

[15] George Karypis and Vipin Kumar. Multilevel$k$-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, January 1998. ISSN 0743-7315. doi: 10.1006/jpdc.1997.1404. URL https://www.sciencedirect.com/science/article/pii/S0743731597914040.

[16] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, January 1998. ISSN 1064-8275. doi: 10.1137/S1064827595287997. URL https://epubs.siam.org/doi/abs/10.1137/S1064827595287997.

[17] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, September 2005. ISSN 0098-3500. doi: 10.1145/1089014.1089017. URL https://dl.acm.org/doi/10.1145/1089014.1089017.

[18] X.S. Li, J.W. Demmel, J.R. Gilbert, iL. Grigori, M. Shao, and I. Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, September 1999. URL https://portal.nersc.gov/project/sparse/superlu/ug.pdf. https://portal.nersc.gov/project/sparse/superlu/ug.pdf Last update: June 2018.

[19] MATLAB. *Version Release 2019b*. The MathWorks Inc., Natick, Massachusetts, 2019.

[20] M.B. Monagan, K.M. Heal and Labahn K.O. Geddes, S.M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 12 Advanced Programming Guide*. Maplesoft, Waterloo, Ontario, Canada, 2008.

[21] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79(11):115112, March 2009. doi: 10.1103/PhysRevB.79.115112. URL https://link.aps.org/doi/10.1103/PhysRevB.79.115112.

[22] Ulrich Rüde, Karen Willcox, Lois Curfman McInnes, and Hans De Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, 2018. doi: 10.1137/16M1096840. URL https://doi.org/10.1137/16M1096840.

[23] Ulrich Rüde, Karen Willcox, Lois Curfman McInnes, and Hans De Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, jan 2018. doi: 10.1137/16m1096840.

[24] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 23:158–179, 2006. ISSN 1068-9613. URL https://eudml.org/doc/127439.

[25] The Trilinos Project Team. *The Trilinos Project Website.*

[26] Aslak Tveito and Ragnar Winther. *Introduction to Partial Differential Equations.* Springer-Verlag, 2005. doi: 10.1007/b138016.

[27] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, March 2006. ISSN 1436-4646. doi: 10.1007/s10107-004-0559-y. URL https://doi.org/10.1007/s10107-004-0559-y.

[28] S. Wolfram. *The Mathematica Book 5.* Cambridge University Press, 2008.

[29] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications*, 35(6):508–526, jun 2021. doi: 10.1177/10943420211022811.