ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# HPC Lab for CSE

FS 2024
Dr. R. Käppeli,
Dr. A. Schell, M. R. Herde,
C. Humbel, A. Mazzoleni,
N. A. Siegenheim

# Project 6

**Due date**: Monday 27 May 2024, 23:59 (midnight).

In this project, we will look at the graph partition problem in context of domain decomposition and popular HPC software libraries and tools widely used in computational science and engineering.
You may do this project in groups of students (max four or five). In fact, we prefer that you do so.

## 1 Graph Partitioning with Julia: Load balancing for HPC [50 points]

Partitioning a computational problem into sub-problems that can be solved efficiently in parallel is a recurring task in HPC. In order to be efficient, the sub-problems should be of near equal size, to ensure load balancing, and the required communication between the sub-problems should be minimized. We have already encountered this problem in a few instances during the course, mainly in the form of domain decomposition. However, the domains were mostly simple and rather square. In this sub-project, we consider more complex grids or meshes. We abstract domain decomposition as a graph partition problem.

The purpose of this sub-project is to familiarize yourself with some elementary graph partitioning algorithms and to implement them in the Julia language. We compare the resulting partitions with the METIS[1] graph partitioning software [18, 19, 20]. METIS was developed by Karypis and Kumar and is probably the most widely used graph partitioning software. In terms of computational efficiency, numerical experiments have demonstrated that graphs with several millions of vertices can be partitioned to 256 parts in a few seconds on current generation workstations and laptops using METIS. We will interface Julia and METIS through the Julia wrapper `Metis.jl` which allows us to use the functions `METIS_PartGraphRecursive` and `METIS_PartGraphKway` with the same arguments and argument order described in the Metis manual.

We refer to Elsner [13] (see the course web page) for a comprehensive introduction to graph partitioning. For (much) more on graph partitioning, we refer to Bichot and Siarry [6]. Many other practical applications and the recent advances in the field of graph partitioning can be found in Buluç et al. [9] and references therein.

### 1.1 Graph partitioning: Generalities

Consider the mesh with 10 cells in Fig. 1a. Assuming that for the computation only cells sharing an edge have to exchange data[2], this mesh can be represented by the dependency graph shown in Fig. 1b. To decompose the mesh in two parts suitable for parallel processing, one tries to partition the mesh into two sub-meshes with the same number of cells and with a minimum number of edges between. This is equivalent to partition the graph in Fig. 1b into two complementary sub-graphs with equal number of vertices and a minimum number of edges between the two sub-graphs. In other words, we partition the graph in two equal parts by cutting the least possible number of edges.

For the simple example in Fig. 1, the solution is obvious. However, the solution is less obvious in general for larger meshes. In the following, we formalize the graph partitioning problem in a suitable manner for the scope of this sub-project and present three bisection algorithms, where bisection restricts the problem to the partition into two sub-graphs. Partitioning into a power of two partitions $p = 2^l$ can then be achieved recursively.

Let $\mathcal{G} = (V, E)$ be an undirected graph with $n$ vertices[3] $V = \{v_1, \ldots, v_n\}$ connected by edges $E = \{e_{i,j} \mid \text{edge between } v_i \text{ and } v_j\}$. The goal is to bisect the graph $\mathcal{G}$ into two complementary sub-graphs

[1] https://github.com/KarypisLab/METIS
[2] This is a common situation in finite volume or discontinuous Galerkin methods.
[3] Depending on the context, vertices may also be called nodes or points.

$\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, that is $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, of near equal size, $|V_1| \approx |V_2|$, such that the number of edges between the parts,

$$\text{cut}(V_1, V_2) = |\{e_{i,j} \in E \mid v_i \in V_1 \text{ and } v_j \in V_2\}|, \tag{1}$$

is minimized. The latter quantity is also known as the cut-size, edge-cut or cost of the partition.

We represent the dependency graph with the so-called adjacency matrix $A = (a_{ij})_{1 \le i,j \le n}$ whose elements are defined by

$$a_{ij} = \begin{cases} 1 & \text{there is an edge } e_{i,j} \text{ between } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that for an undirected graph the adjacency matrix is symmetric. The so-called degree of a vertex $\deg(v_i) = \sum_{j=1}^{n} a_{ij}$ is the number of edges that are connected to the vertex $v_i$. This is encoded in the degree matrix $D = \text{diag}(d_1, \ldots, d_n)$. With the adjacency and degree matrix, we can form the Laplacian matrix

$$L = D - A. \tag{3}$$

The latter will play an essential role in the spectral bisection algorithm presented below.

As an example, let us consider again the mesh and corresponding dependency graph in Fig. 1. The adjacency and degree matrices are given by

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and the Laplacian matrix

$$L = D - A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 3 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}.$$

This can easily be formalized to an arbitrary number of partitions $p$ and weighted vertices and edges. See Elsner [13] for details.

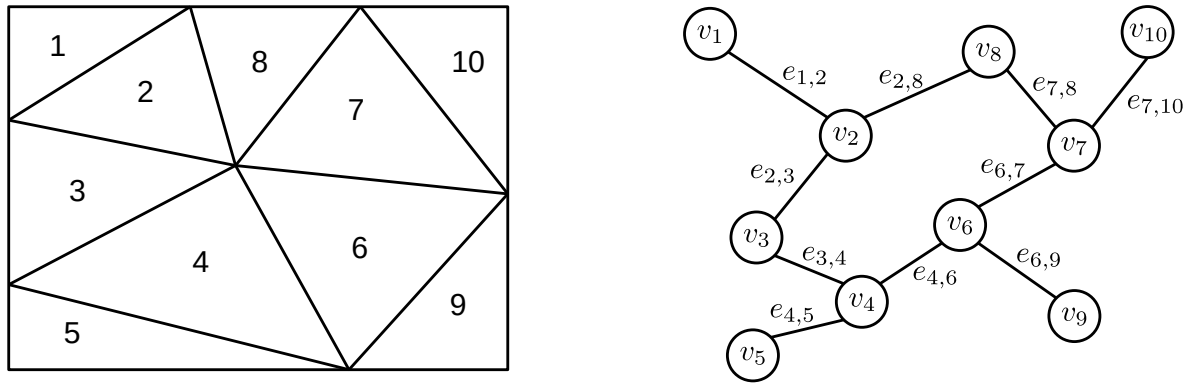
## 1.2 Graph partitioning: Simple algorithms

### 1.2.1 Partitioning with geometric information: Coordinate and inertial bisection

We start with two bisection algorithms that assume that the geometric layout of the graph is known. This is for instance the case in our application of domain decomposition of a given mesh.

#### Coordinate bisection

Coordinate bisection simply consists of finding the hyperplane orthogonal to a coordinate axis that divides the graph vertices in two (almost) equal parts with the smallest edge-cut. This is achieved by computing the median $\overline{x}_i$ over each coordinate axis $x_i$ (e.i., $x_1 \equiv x$ and $x_2 \equiv y$), that is $\overline{x}_i$ separates all vertices of the graph in two with half having $x_i$ coordinates less and half larger than $\overline{x}_i$. Then one computes the edge-cut for each of the coordinate axes and bisects along the one with the smallest edge-cut. This algorithm is summarized in Algorithm 1 for two dimensions.




(a) (b)

Figure 1: Mesh with 10 (simplex/triangle) cells (left) and corresponding dependency graph (right).

Coordinate bisection is a very simple bisection algorithm. However, the quality of the resulting partition may depend strongly on the coordinate systems (e.g., a simple coordinate axes rotation may lead to very different results). We refer to Elsner [13] for further information.

**Algorithm 1** Coordinate bisection.

**Require:** $\mathcal{G}(V, E), P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the vertices
**Ensure:** A bisection of $\mathcal{G}$
1: **function** InertialBisection(graph $\mathcal{G}(V, E), P_i$)
2: For each coordinate axis, $x$ and $y$, find the median value $\overline{x}$ and $\overline{y}$ that divides the vertices in two equal parts.
3: Partition the vertices of the graph around median coordinate line having the smallest edge-cut.
4: **return** $V_1, V_2$ ▷ bisection of $\mathcal{G}$
5: **end function**

### Inertial bisection

Inertial bisection improves upon the axes dependence of coordinate bisection by choosing the dividing hyperplane orthogonal along a direction that runs through the center of mass of the vertices. In two dimensions, such a line $L$ is chosen such that the sum of squares of the distance of the vertices to the line is minimized. It is defined by a point $\overline{P} = (\overline{x}, \overline{y})$ and a vector $\mathbf{u} = [u_1, u_2]^T$ with $\|\mathbf{u}\|_2 = \sqrt{u_1^2 + u_2^2}$ such that $L = \{\overline{P} + \alpha\mathbf{u} \,|\, \alpha \in \mathbb{R}\}$ (see Fig. 2). We set

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \overline{y} = \frac{1}{n}\sum_{j=1}^{n} y_j, \tag{4}$$

as the center of mass lies on the line $L$. Finally, we need to compute $\mathbf{u}$ in order to minimize the sum of distances

$$\begin{aligned}\sum_{i=1}^{n} d_i^2 &= \sum_{i=1}^{n} (x_i - \overline{x})^2 + (y_i - \overline{y})^2 - (u_1(x_i - \overline{x}) + u_2(y_i - \overline{y}))^2 \\ &= u_2^2 \sum_{i=1}^{n} (x_i - \overline{x})^2 + u_1^2 \sum_{i=1}^{n} (y_i - \overline{y})^2 + 2u_2u_1 \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \\ &= \mathbf{u}^T \begin{bmatrix} S_{yy} & S_{xy} \\ S_{xy} & S_{xx} \end{bmatrix} \mathbf{u} = \mathbf{u}^T M \mathbf{u}, \end{aligned} \tag{5}$$

where $S_{xx}, S_{xy}, S_{yy}$ are the sums as defined in the previous line. The resulting matrix $M$ is symmetric, thus the minimum of Eq. (5) is achieved by choosing $\mathbf{u}$ to be the normalized eigenvector corresponding to the smallest eigenvalue of $M$. The procedure of bisecting a graph using inertial partitioning is summarized in Algorithm 2. We refer again to Elsner [13] and references therein for a thorough overview of the inertial bisection algorithm.

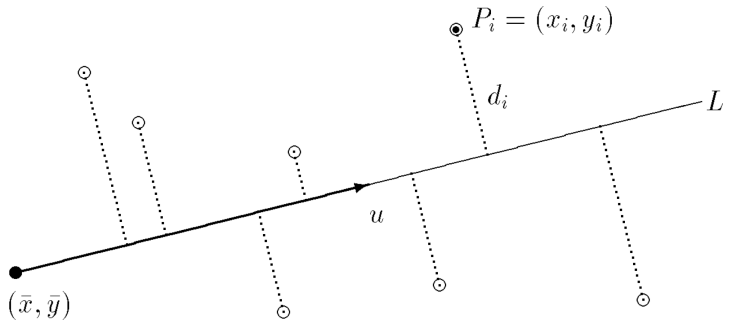



Figure 2: Illustration of inertial bisection in 2D (from Elsner [13]).

**Algorithm 2** Inertial bisection.

**Require:** $\mathcal{G}(V, E)$, $P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the vertices
**Ensure:** A bisection of $\mathcal{G}$
1: **function** InertialBisection(graph $\mathcal{G}(V, E)$, $P_i$)
2: Calculate the center of mass of the points ▷ acc. to (4)
3: Compute the eigenvector associated with the smallest eigenvalue of $\mathbf{M}$ ▷ $\mathbf{M}$ acc. to (5)
4: Partition the vertices of the graph around line $L$.
5: **return** $V_1, V_2$ ▷ bisection of $\mathcal{G}$
6: **end function**

### 1.2.2 Partitioning without geometric information: Spectral bisection

Spectral bisection was initially considered the standard for solving graph partitioning problems. Unlike the previously introduced bisection algorithms, it does not use any geometric information associated with the graph to partition. A bisection is computed using the eigenvector associated with the second smallest eigenvalue of the Laplacian matrix $L$ of the graph. The graph Laplacian $L$ is a symmetric positive semi-definite matrix[4], with its smallest eigenvalue being $\lambda^{(1)} = 0$, and the associated eigenvector $\mathbf{u}^{(1)} = c\mathbf{1}$ being the constant one. The eigenvector $\mathbf{u}^{(2)}$ associated with the second smallest eigenvalue $\lambda^{(2)}$ is Fiedler's celebrated eigenvector, and is crucial for spectral graph partitioning. Each node $v_i$ of the graph is associated with one entry in $\mathbf{u}^{(2)}$. Thresholding the values of $\mathbf{u}^{(2)}$ around 0 results in two roughly balanced (equal sized) partitions, with minimum edge-cut, while thresholding around the median value of $\mathbf{u}^{(2)}$ produces two strictly balanced partitions. The procedure to compute a bisection using spectral partitioning is summarized in Algorithm 3. For a much more detailed description of the algorithm, we again refer to Elsner [13].

**Algorithm 3** Spectral bisection

**Require:** $\mathcal{G}(V, E)$,
**Ensure:** A bisection of $\mathcal{G}$
1: **function** SpectralBisection(graph $\mathcal{G}(V, E)$)
2: Form the graph Laplacian matrix $\mathbf{L}$
3: Calculate the second smallest eigenvalue $\lambda^{(2)}$ and its associated eigenvector $\mathbf{u}^{(2)}$.
4: Set 0 or the median of all components of $\mathbf{u}^{(2)}$ as threshold $\epsilon$.
5: Choose$V_1 := \{v_i \in V | u_i < \epsilon\}$, $V_2 := \{\{v_i \in V | u_i \geq \epsilon\}$.
6: **return** $V_1, V_2$ ▷ bisection of $\mathcal{G}$
7: **end function**

## 1.3 Graph partitioning: Recursive bisection

A simple and robust algorithm to create a $p = 2^l$ partition, with $l$ being an integer, is recursive bisection and it is presented in Algorithm 4. It uses the recursive function `Recursion` that takes as inputs the part $C'$ of the graph to partition, the number of parts $p'$ into which we will partition $C'$, and the index (an integer) of the first part of the partition of $C'$ into the final partitioning result $\mathcal{G}_p$.

[4] According to the spectral theorem of linear algebra the Laplacian matrix $L$ therefore possesses an orthogonal basis of eigenvectors $\mathbf{u}^{(i)}$ and corresponding real eigenvalues $\lambda^{(1)}$.

**Algorithm 4** Recursive bisection.

**Require:** $\mathcal{G}(V, E)$,
**Ensure:** A $p$-way partition of $\mathcal{G}$
1: $\mathcal{G}_p = \{C_1, \ldots, C_p\}$
2: $p = 2^l$ ▷ p is a power of 2
3: **function** RecursiveBisection(graph $\mathcal{G}(V, E)$, number of parts $p$)
4: **function** Recursion($C'$, $p'$, index)
5: **if** $p'$ is even **then** ▷ If $p'$ is even, a bisection is possible
6: $p' \leftarrow \frac{p'}{2}$
7: $(C'_1, C'_2) \leftarrow$ `bisection`$(C')$
8: Recursion($C'_1, p'$,index)
9: Recursion($C'_2, p'$,index+$p'$)
10: **else** ▷ No more bisection possible, $C'$ is in a partition of $\mathcal{G}$
11: $C_{index} \leftarrow C'$
12: **end if**
13: **end function**
14: Recursion$(C, p, 1)$
15: **return** $\mathcal{G}_p$ ▷ $\mathcal{G}_p$ is a partition of $\mathcal{G}$ in $p = 2^l$ parts
16: **end function**

## 1.4 Graph partitioning with Julia

The goal of this sub-project is to familiarize yourself with various algorithms for graph partitioning applied to domain decomposition. We have chosen the Julia language as our tool due to its high-level, dynamic capabilities, which are increasingly popular in HPC for some specific tasks. For more information about the Julia programming language, including package management and environment setup, please refer to the Julia documentation.

### 1.4.1 Environment setup

The following commands log in to Euler and set up the Julia environment:

```
[user@local]$ ssh -Y euler.ethz.ch
[user@eu-login-43]$ module load gcc julia eth_proxy

The following have been reloaded with a version change:
  1) gcc/4.8.5 => gcc/11.4.0

[user@eu-login-43]$ export
↪  JULIA_DEPOT_PATH="$HOME/.julia:/cluster/work/math/hpclab/julia:$JULIA_DEPOT_PATH"
[user@eu-login-43]$ srun --time=4:00:00 --mem-per-cpu=8G --x11 --pty bash
srun: job 58201149 queued and waiting for resources
srun: job 58201149 has been allocated resources
[user@eu-a2p-284]$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.2 (2024-03-01)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 1 + 1
2

julia>
```

The first command enables X11 forwarding, the second loads the necessary modules, the third sets the Julia depot path (where we have already installed the necessary packages), the fourth submits an

interactive job to the queue (1 CPU, 8 GB of memory and X11 forwarding enabled for 4 hours) and the final command launches a Julia interactive session (often referred to as "REPL" for Read-Eval-Print-Loop). For the duration of the project, we recommend adding the second and third commands to your `.bashrc` configuration script (located in `$HOME/.bashrc`).

Before starting to work on the sub-project tasks, you need to activate the provided project package. Go into the directory `graph_part` (containing the `Project.toml` and `Manifest.toml`). In there, start an interactive Julia session and enter the package manager `Pkg` by pressing the `]` key to activate the package:

```
julia> ]
(@v1.10) pkg> activate .
```

Once the environment set up, you may run the `main.jl` script in the REPL

```
julia> include("main.jl")
```

This will create several PDFs of the mesh/graph in `airfoil1.mat` displayed in Fig. 3 and report the partitions' edge-cut[5]. You can display the PDFs with

```
[user@eu-a2p-284]$ display airfoil1.pdf # airfoil1_coordinate.pdf
                                        # or airfoil1_metis.pdf
```

To run code in a file non-interactively, you can

```
[user@eu-a2p-284]$ julia --project=. ./main.jl
```

in the provided Julia project directory.

Alternatively, you can also install Julia and the packages on your own machine. First, you need to install Julia. You can either use an appropriate package manager, pre-compiled binaries or compile directly from source. The last option is probably more involved. Please follow the instructions available at https://julialang.org/downloads/. Once Julia is installed, launch an interactive Julia REPL session and either activate and instantiate the project environment[6]

```
julia> ]
(@v1.10) pkg> activate "/path/to/julia/project/"
(@v1.10) pkg> instantiate
```

or manually install the packages into your `MyOwnProject` with the Julia package manager

```
julia> ]
(@v1.10) pkg> activate "MyOwnProject"
(@v1.10) pkg> add Arpack CairoMakie Colors Graphs LinearAlgebra MAT Metis
↪  PrettyTables Random SGtSNEpi SparseArrays Statistics
```

### 1.4.2 The sub-project tasks

Complete the following tasks

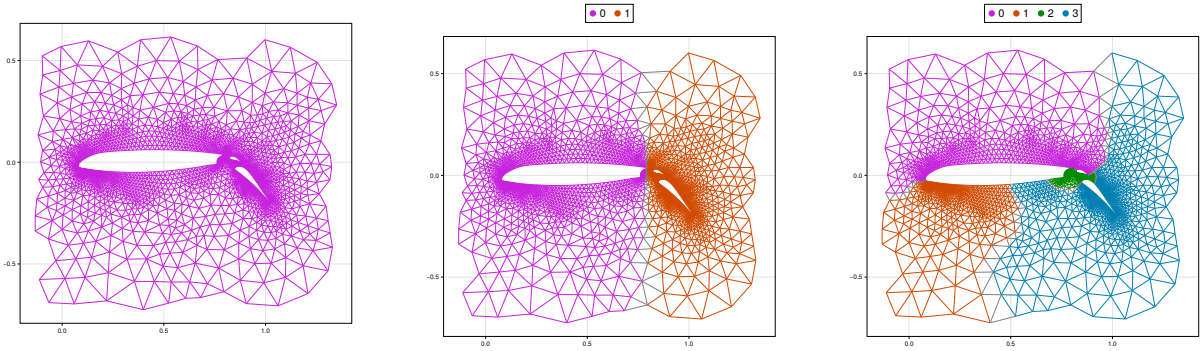
(1) Familiarize yourself with the provided skeleton code.

(2) Implement graph partitioning algorithms **[30 points]**.

- Implement the inertial bisection algorithm. Use the provided skeleton code `part_inertial.jl`. **Hint**: In the skeleton code, we provide an implementation of coordinate bisection.
- Implement the spectral bisection algorithm. Use the provided skeleton code `part_spectral.jl`.

[5] You may also see a warning from the `Makie.jl` package. This is unfortunate, but can be safely ignored.

[6] This might give a warning if your installed Julia version is different from the one on Euler. One solution is to manually install the packages or to use the Julia installer and version multiplexer `juliaup` (https://github.com/JuliaLang/juliaup).



(a) (b) (c)

Figure 3: The mesh of an airfoil: original (left), partitioned in two with coordinate bisection (middle) and four with METIS (right).

- Report the bisection edge-cut for all toy meshes that are loaded in the function `bench_bisection.jl` by running the script `main_bench.jl`. Qualitatively discuss your results briefly. Do you observe any surprising, or sub-optimal, partitions? For illustration, please also include figures of the partitions (i.e., similar to Fig. 3).

For the eigenvalue computations, use the `eigs` function from the `Arpack` package. Please visit Arpack.jl[7] for more information.

(3) Implement the recursive bisection algorithm **[20 points]**.

- Implement the recursive bisection algorithm in `recursive_bisection.jl`.
- Report the bisection edge-cut for all toy meshes that are loaded in the function `bench_recursive.jl` by running the script `main_bench.jl`. Qualitatively discuss your results briefly. Do you observe any surprising, or sub-optimal, partitions? For illustration, please also include figures of the partitions (i.e., similar to Fig. 3).

# 2 HPC software frameworks [50 points]

In this sub-project, we provide you with an 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 sub-project is then to solve a simple boundary value problem (Poisson equation with Dirichlet boundary condition) using the PETSc toolkit.

## 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. [24] on research and education in CSE.

### Parallel Dense Matrix Software Libraries

**BLAS** The Basic Linear Algebra Subprogramm (BLAS) [12] 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 —

[7] https://arpack.julialinearalgebra.org/latest/

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 [11] for matrix-matrix operations.

**ScaLAPACK** Scalable LAPACK (ScaLAPACK) [7] 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. [15].

### 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 [20]. The parallel version of MUMPS requires MPI for message passing and makes use of the BLAS, BLACS, and ScaLAPACK libraries.

**SuperLU** SuperLU [21, 22] 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 [25] 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 [14] 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 [16, 23] 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) [28] 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 .[8]

#### 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 [8] 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 [18, 19, 20] 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 [26] 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

[8] https://www.coin-or.org/2011/04/07/ipopt-wins-the-wilkinson-prize-for-numerical-software/

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.[9] 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. 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 [10]. 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 [50 points]

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

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{in} \quad \partial\Omega. \end{aligned} \tag{6}$$

Here, $u = u(x_1, x_2)$ is the unknown function, $f = f(x_1, x_2)$ is a prescribed source function, $\Delta$ is the Laplace operator, $\Omega$ is the spatial domain, and $\partial\Omega$ is the boundary of $\Omega$. We denote by $\overline{\Omega}$ the corresponding closed set $\overline{\Omega} = \Omega \cup \partial\Omega$. The Poisson equation Eq. (8) arises in numerous physical contexts, including heat conduction, electrostatics, Newtonian gravity, diffusion of substances, twisting of elastic rods, inviscid 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

$$\Delta \equiv \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}. \tag{7}$$

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 ($\Omega$), 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 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 [27, Sec. 7.5.1].

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

[9] https://www.siam.org/prizes-recognition/major-prizes-lectures/detail/siam-acm-prize-in-computational-science-and-engine

### 2.2.1 Environment setup

To get started on Euler cluster, you need to load the necessary modules

```
[user@eu-login-43]$ module load gcc openmpi petsc

The following have been reloaded with a version change:
  1) gcc/4.8.5 => gcc/11.4.0
```

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

```
[user@eu-login-43]$ export petsc_dir=$petsc_root
```

`PETSC_ROOT` is set by loading the PETSc module. The available version on the Euler cluster may not be the most recent release of PETSc, but this is not an issue for the current sub-project. Now you should be able to compile run the PETSc test program provided in the skeleton code directory `hello_petsc` as follows

```
[user@eu-login-43]$ make
...
[user@eu-login-43]$ salloc -n 8 --nodes=4 --ntasks-per-node=2 # allocate 8 processes
↪  on 4 nodes with 2 processes each
...
salloc: Nodes eu-a2p-[277,281-282,284] are ready for job
[user@eu-login-43]$ mpirun ./hello_petsc
Hello world from rank 4 out of 8 on eu-a2p-282
Hello world from rank 6 out of 8 on eu-a2p-284
Hello world from rank 2 out of 8 on eu-a2p-281
Hello world from rank 5 out of 8 on eu-a2p-282
Hello world from rank 7 out of 8 on eu-a2p-284
Hello world from rank 3 out of 8 on eu-a2p-281
Hello world from rank 0 out of 8 on eu-a2p-277
Hello world from rank 1 out of 8 on eu-a2p-277
```

You are now all set with PETSc!

### 2.2.2 The sub-project tasks

Your task is to solve with PETSc the following boundary value problem

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{in} \quad \partial\Omega \end{aligned} \tag{8}$$

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

$$\Omega = \{x = (x_1, x_2) \mid 0 < x_1, x_2 < 1\}\,. \tag{9}$$

1. Discretize the boundary value problem with second-order centered finite-differences and reformulate the discrete problem into a linear linear system to solve. Summarize your discretization in your report.

2. Solve the boundary problem with PETSc. Summarize the solvers (preconditioners, ...) your implementation uses in your report.
   **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.

3. Visualize your approximate solution $u(x_1, x_2)$.

4. Perform a strong scaling analysis. Choose a few representative resolution.

## Additional notes and submission details

Submit **all the source code files** together with your used build files (e.g., `Makefile`(s)) and other scripts (e.g., batch job scripts) in an archive file (tar or zip) and summarize your results and observations for all sections by writing a detailed LaTeX report. Use the LaTeX template from the webpage and upload the report as a PDF to Moodle.

- Your submission should be a tar or zip archive, formatted like `project_number_lastname_firstname.zip/tgz`. It must 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 report with your name.
  - Follow the provided guidelines for the report.
- Submit your archive file through Moodle.

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] Charles-Edmond Bichot and Patrick Siarry, editors. *Graph Partitioning*. Wiley, feb 2013. doi: 10.1002/9781118601181.

[7] 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.

[8] 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/.

[9] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent Advances in Graph Partitioning. Lecture Notes in Computer Science, pages 117–158. Springer International Publishing, Cham, 2016. ISBN 9783319494876. doi: 10.1007/978-3-319-49487-6_4. URL https://doi.org/10.1007/978-3-319-49487-6_4.

[10] 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.

[11] 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.

[12] 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.

[13] Ulrich Elsner. Graph partitioning - a survey. Technical report, Technische Universität Chemnitz, September 2005. URL https://nbn-resolving.org/urn:nbn:de:swb:ch1-200501047.

[14] 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.

[15] 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.

[16] 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.

[17] G. Karypis and V. Kumar. Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs. In *Supercomputing '96:Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 35–35, January 1996. doi: 10.1109/SUPERC.1996.183537.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

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

[28] 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.

[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.